SHARIF UNIVERSITY OF TECHNOLOGY
PHYSICS DEPARTMENT

Problem Set 11

Student name: *Ali Fayyaz - 98100967*

Course: *Computational Physcis - (Spring 2023)*
Due date: *June 16, 2023*

Exercise 11.1

Molecular Dynamics

Simulate a microcanonical ensemble of a system of 100 Argon atoms in a 2D space. The initial conditions are such that all atoms are in the left half of the container in a crystal formation with random velocities.

1. Obtain a trajectory of the system.
2. Plot the fraction of atoms in the left half of the container through time, as an indicator of equilibrium.
3. Anylize the conservation of energy in this system.
4. Find the auto-correlation of velocites and the equilibration time of the system.
5. Find the temperature and pressure of the system after equilibrium is reached. To find the temperature, use the kinetic energy of the particles, and to find the pressure, use Virial's theorem.
6. Find the Van der Waals constants a and b.
7. Scale the velocities and reduce the temperature to see the phase shift.

Answer. The Lennard-Jones potential reads:

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

The following reduced constants are used:

Energy	$\epsilon^* = 1$
Distance	$\sigma^* = 1$
Mass	$m^* = 1$
Boltzmann's Constant	$k_B^* = 1$
Temperature	$T^* = \frac{\epsilon}{k_B}$
Time	$t^* = \sqrt{\frac{m\sigma^2}{\epsilon}}$

A square shaped container of side length $L = 30$ (reduced) was simulated with $N = 100$ particles. The radius of cut-off for the L-J potential and the maximum velocity of the particles was set to be $r_{cutoff} = 2.5$ and $v_{max} = 5$ in reduced units, respectively.

A matrix *particles* of shape *(6, N)* was to contain all the data for the trajectory of particles, such that the 1^{st} row contained the x component of the positions of the particles, the 2^{nd} row the y component of the positions, the 3^{rd} and 4^{th} rows the x and y components of the velocities, and the 5^{th} and 6^{th} rows the x and y components of the accelerations, respectively.

The following functions were declared, all of which take as input the global variable *particles* and modify it as desired:

- **initialize_system():**

  This function initializes the system in such a manner as described in the question: a crystal structure in the left half of the container. It also gives the particles random (uniform distribution) velocities in random directions according to v_{max}. Then the average of all velocities is subtracted from all the particles to set the velocity of the Center of Mass equal to zero.

- **relative_distances():**

  Applies Minimum Image Convention and computes the pairwise distances between particles. In order to compute the pairwise distances, *for loops* are avoided using a numpy broadcasting trick. It involves using a 3D matrix where (i, j, k) shows the difference between the k'th coordinate of particle i and the k'th coordinate of particle j. Given periodic boundary conditions are used, care is taken to compute the minimum distance between one particle and the other particles, or the images of the other particles. Then, applying ternary conditions using *numpy.where()*, *NaN* is put inplace of distances smaller than σ and larger than r_{cutoff}. Finally, the 2D matrix of pairwise distances is returned.

- **acc():**

  The matrix of pairwise distances is taken as input and the matrix of the magnitude of accelerations is computed using the derivative of the L-J potential. After that, the angles between the particles are calculated to find the x and y components of accelerations, which are then put in the 5^{th} and 6^{th} rows of *particles*.

- **velocity_verlet():**

  First, periodic boundary conditions are applied to the postions of all particles. Then, using the half-step Velocity-Verlet algorithm, velocities, positions and accelerations in *particles* are updated. To avoid numerical instability, the velocities are scaled such that the maximum velocity never exceeds v_{max} as specified at the beginning of the simulation. Finally, the velocity of the center of mass is set to zero.

- **kinetic_energy():**

  A one-line function to compute the kinetic energy of the particles, based on their velocities, and return the corresponding values in an array.

- **potential_energy():**

  Using the function *relative_distances()*, computes the potential energy associated with each particle, based on the relation for the L-J potential.

- **press():**

  It takes the ralative distances of the particles, calculates the reduced temperature of the system using the average of the kinetic energy of the particles, then computes the pressure using Virial's theorem.

- **auto_correlation():**

  Takes an array and computes the Pearson Correlation Coefficient using *numpy.corrcoef()*, for different lengths of the array.

- **velocity_auto_correlation():**

  It uses a matrix, each column of which contians the evolution of the magnitude of the velocity of one particle. It calls the *velocity_verlet()* function *num_step* number of times (as specified in the input of the function), calculates the auto-correlation of velocities for each particle with the help of the previous function, then returns the average of all the auto-correlations along with the auto-correlation length, ξ.

- **trajectory():**

  As the name indicates, this function calls the *velocity_verlet()* function, a number of times, takes *snapshots* in different intervals, all as specified by the user, and records the data for *particles* in a 3D matrix, such that each *sheet* contians the positions, velocities and accelerations of all particles in a certain step of time.

- **take_snapshots():**

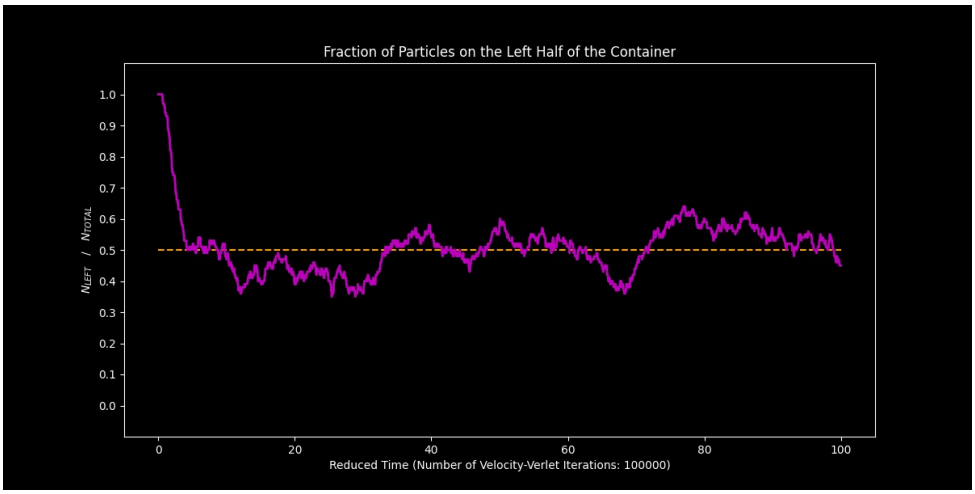
  This function is used to record data for kinetic energy, potential energy, temperature, pressure and the fraction of particles on the left half of the container, at specific intervals of time. This data will then be used for plotting, as necessary.

1. **Trajectory**

   The system was initialized and then run for 10,000 velocity-verlet steps. With a step size of $h = 0.001$, this corresponds to 10 units of reduced time. A *snapshot* was taken every 10 steps, equivalent to time intervals of 0.01 in reduced time units. The matrix obtained from the *trajectory()* function was then used, along with *pandas.DataFrame()* to write the x and y components of postitions, velocities and accelerations in 6 seperate csv files as the trajectory of the system.

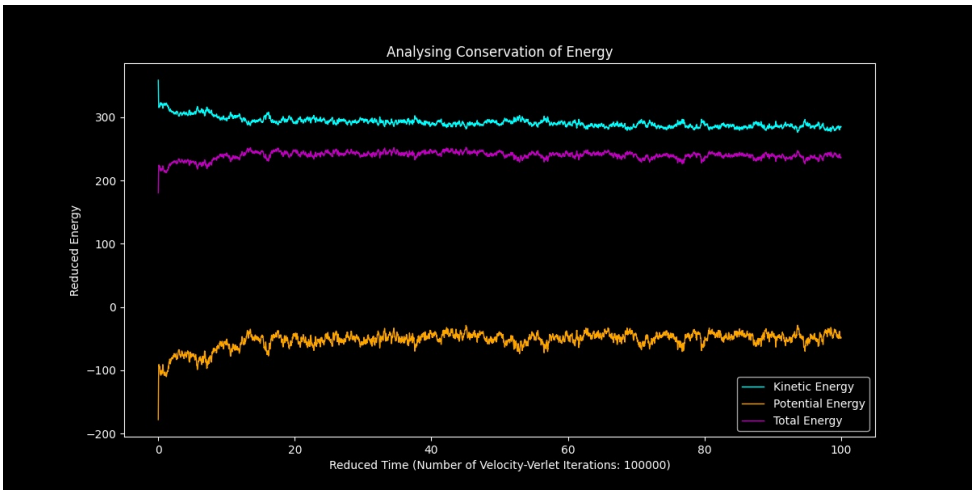
2. **Fraction of Atoms in the Left Half**

   The system was initialized. With 100,000 Velocity-Verlet steps, 10,000 snapshots were taken. It is seen from the graph that with $v_{max} = 5$, the system reaches equilibrium at time 10 in reduced units, corresponding to 10,000 Velocity-Verlet steps.



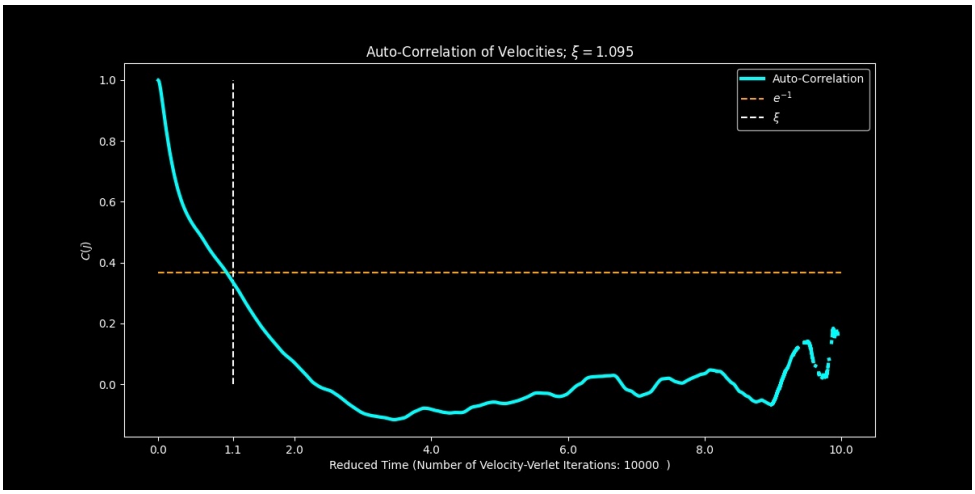
3. **Conservation of Energy**

   Again, with 100,000 Velocity-Verlet steps, the potential and kinetic energies were recorded every 10 steps, using the corresponding functions as explained earlier, and the total energy, obviously, was calculated as the sum of the two energies in every *snapshot*. The graph shows equilibrium is reached after 20 reduced time units, and the conservation of energy is apparent.



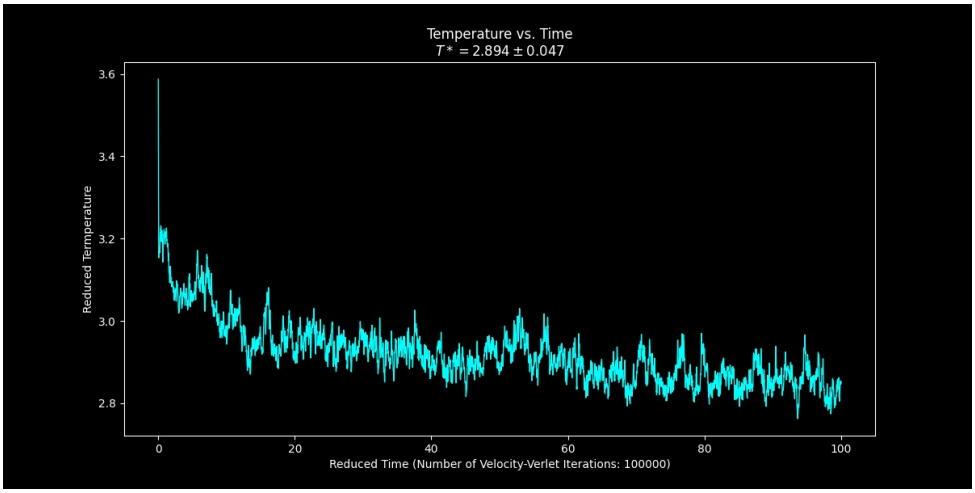
4. **Auto-Correlation of Velocities**

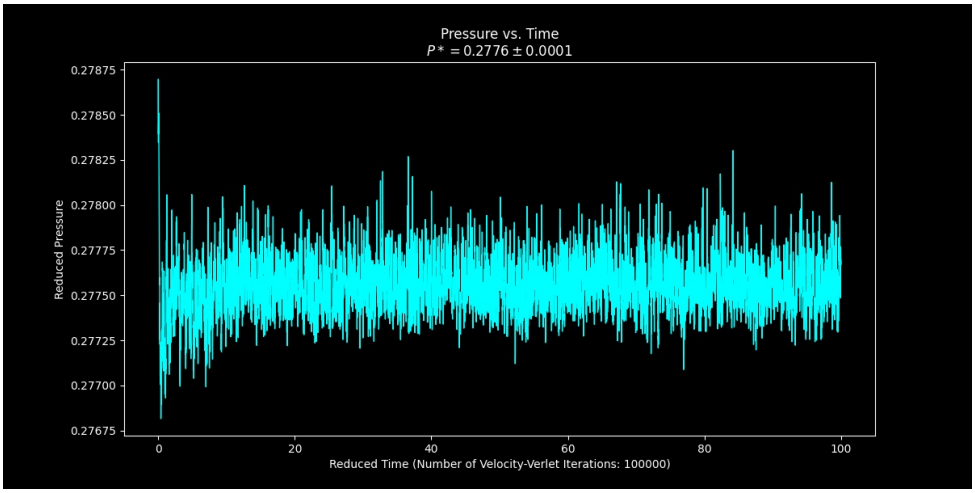
   The system was initialized and 10,000 Velocity-Verlet steps were taken. Using the function *velocity_auto_correlation()*, as the graph shows, the correlation time was calculated to be 1.095 units in reduced time, corresponding to 1,095 Velocity-Verlet steps.



5. **Temperature and Pressure**

   The temperature and pressure were recorded for 100,000 Velocity-Verlet iterations using the function *take_snapshots()*. The values were computed after 2000 steps, when the system had reached equilibrium.





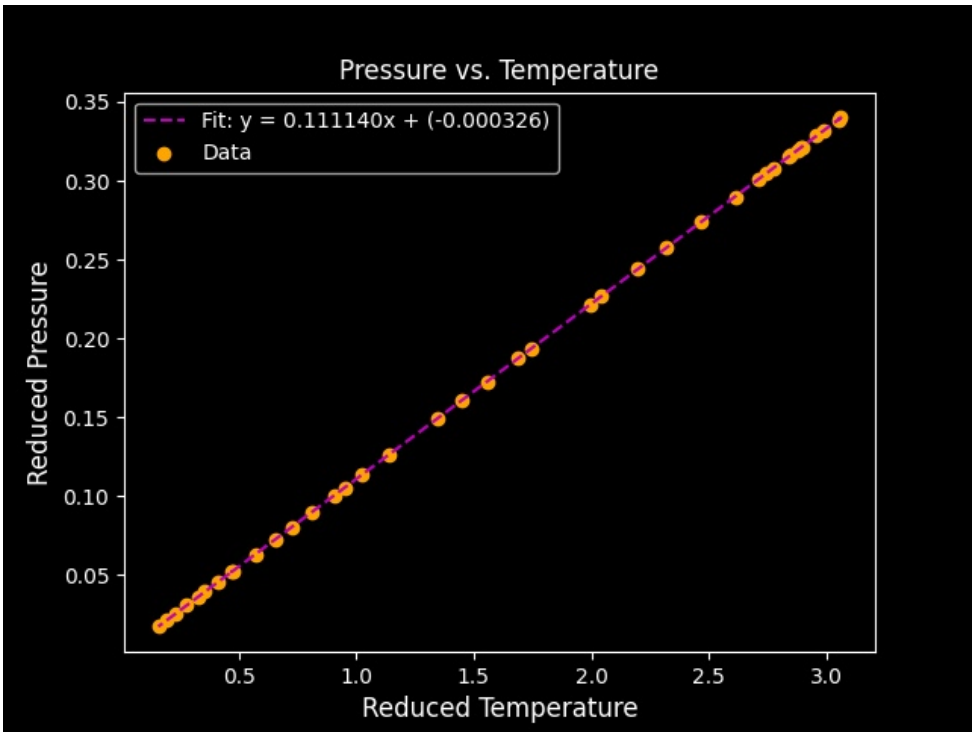
6. **Van der Waals Constants**

   To calculate the Van der Waals constants, the system was initialized with $v_{max} = 5$ and run for 2000 steps. Then it was run for 3000 steps for each value of $v_{max} \in \{5, 4.9, 4.8, ..., 1.2, 1.1\}$ and the temperature and pressure was recorded. Given the Van der Waals equation as follows:

$$P = \left(\frac{R}{V_m - b}\right) T - \left(\frac{a}{V_m^2}\right)$$

   where $V_m = 22.4\ (L/Mol\ in\ STP)$ is the molar volume, the constants in SI Units are as follows:

$$a_{calculated} = 6.886\ (L^2 bar/mol^2) \quad a_{Wikipedia} = 1.355\ (L^2 bar/mol^2) \quad ERROR = 408.186\%$$
$$b_{calculated} = 0.022\ (L/mol) \quad b_{Wikipedia} = 0.03201\ (L/mol) \quad ERROR = 30.687\%$$



7. **Phase Transition**

   Animations for the three phases of solid, liquid and gas were made with *matplotlib.animation.FuncAnimation()*.