PHY407 Final Project

2-D Molecule System Simulation in a Box, Governed by Langevin Dynamics under Simulated Annealing

Rundong Zhou

December 18, 2020

1 Overview

In this project, I developed a 2-D multi-molecule system in a closed box. The system is mostly governed by Langevin dynamics and Lennard-Jones potential, and it undergoes a modified Verlet method. Since the Langevin dynamics involves random process and depends on the temperature of the system, to make the project more interesting, I introduced the Markov Chain method to govern each time step produced by the randomness of Langevin dynamics based on the total energy and temperature of the system. The system also experienced a simulated annealing process, through which I can investigate the ground state of the molecule system. So the project is somewhat based on the materials from Lab6 and Lab11.

2 Physics Background

2.1 Langevin Dynamics

In real life situations, molecule systems are not likely to happen in a complete vacuum. They tend to exist in a sort of medium, like solvent or air. So the molecules will not only experience forces due to each other, but also friction and random high-speed collision exerted by the medium. Thus, a French physicist Paul Langevin developed the Langevin dynamics to simulate such molecule systems in medium.
The Langevin dynamics can be summarized by an elegant formula:

$$M\ddot{X} = -\nabla U(X) - \gamma\dot{X} + \sqrt{2\gamma k_B T}R(t)$$

In this formula, M is the mass of the particle. $X = X(t)$ represents the position of all N particles. $U(X)$ is the potential function due to the interaction of the particles themselves, and $-\nabla U(X)$ calculates the forces exerted on the particles due to each other. In this project, I used Lennard-Jones potential.
γ represents the viscosity of the medium, and $-\gamma\dot{X}$ calculates how the particles get damped due to the friction exerted by the medium. k_B is Boltzmann's constant and T is the temperature of the system.
$R(t)$ is a time-independent Gaussian process with zero-mean. The term $\sqrt{2\gamma k_B T}R(t)$ represents the random interaction between the particles and the medium, which is the most interesting part of the formula. It plays the most important role in my project. This part is mostly modified from Wikipedia.

2.2 Lennard-Jones Potential

This part follows the same background introduced in Lab6. It represent interactions between molecules at large distances due to Van der Waals forces. The potential is calculated by the following formula:

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

r is the distance between two particles. ϵ is the depth of the potential well. The r^{-6} term represents the attractive force and the r^{-12} term represents the repulsive force.

3 Computational Background

3.1 Modified Verlet Method with Markov Chain

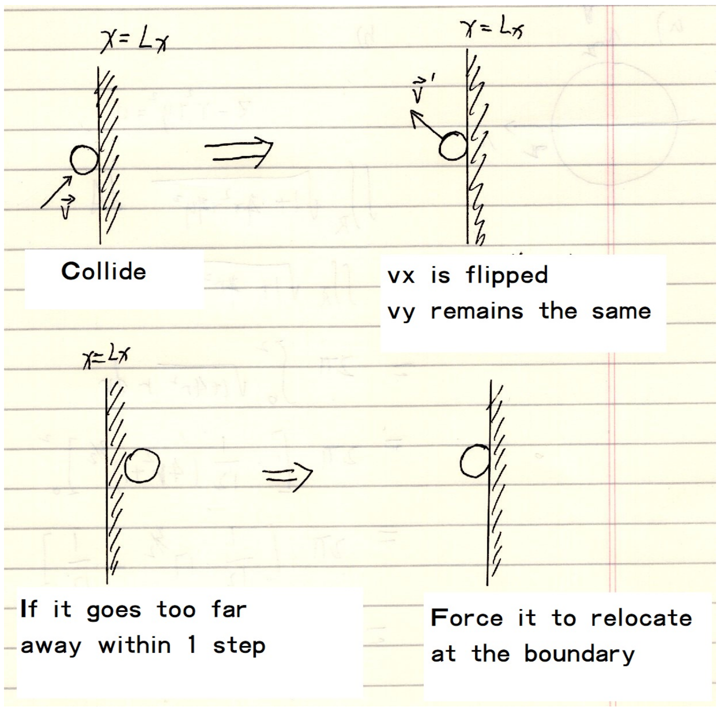
As it was introduced in Lab6, the original Verlet method conserves the total energy of the molecule system under pure Lennard-Jones potential. However, since I introduced the Langevin dynamics to the system, the energy will no longer be conserved due to the random term $\sqrt{2\gamma k_B T}R(t)$ of the new dynamics, also due to the damping effect $-\gamma\dot{X}$.
To govern the energy of the system, I added the Markov Chain method to the original Verlet method. At each time step, the full Langevin formula will first be used to calculate the forces and the resulting velocity & position changes on particles. Then, it monitors the total energy (kinetic + potential) and decides whether the random collision process between the particles and the medium will be revoked.

$$P = \begin{cases} 1 & \text{if } E_{t+h} \leq E_t \\ e^{-(E_{t+h}-E_t)/k_B T} & \text{if } E_{t+h} > E_t \end{cases}$$

If a Langevin dynamics random process is revoked, then the forces are recalculated without the random term $\sqrt{2\gamma k_B T}R(t)$, but the damping term is kept. And the temporal loop runs following the Verlet algorithm. The modified method will be further exposed in the pseudo code part.

3.2 Closed Box Boundary Condition

To make this project interesting, I decide to place the molecule system in a closed box. Such that the particles will not simply run away in free space (like Lab6 Q3a) or wiggle around with less interaction under periodic boundary condition (like Lab6 Q3c). Particles in a box will interact with each other most fiercely, so it is even more exciting to observe how they converge to a ground state beautifully under simulated annealing.
Collision with the wall is fully elastic. So once a particle hits the wall, the sign of the its corresponding velocity component is flipped. And if the particle exceed the boundary, it will be relocated at the boundary, in case the velocity is high the particle goes a little bit too far.
The way to implement the box boundary condition will also be further explained in the pseudo code part.



(Cannot find an ideal figure online, so I draw it by myself)

3.3 Simulated Annealing and Efficiency Concerns

The topic is fully explained in Lab11, so I will not waste space here. However, there are something worth mentioning. I set the time constant relatively small $\tau \sim 10$ for two reasons:

- Unlike Lab11, the time step is small in this project ($h = 0.01$) for the accuracy of the simulation. And each time step is very costly since I need to compute forces, velocities, energies, etc. If I set the time constant as big as in the lab, the run time will be forever for the system to cool down to a desired temperature. So it is wise to choose a small τ to let the system cool quickly under a reasonable time consumption.
- Another reason to choose a small time constant is that, there isn't really a local minimum in the system due to the nature of the Lennard-Jones potential function. So no matter how fast I cool the system, it will always converge to its global minimum, which is the desired ground state of the multi-molecule system. But don't cool it too fast, or the molecule motion will die immediately and the whole simulation becomes boring.

4 Method Breakdown and Pseudo Code

4.1 Important Functions

First, the function to calculate the forces acting on all N particles with Langevin dynamics including the random process:

```
def Force_random(x, y, vx, vy, gamma, kB, T, N):
#x, y are arrays with N elements, corresponding to the position of each particle
#vx, vy are the velocity of each particle, gamma is the viscosity
#T temperature, kB Boltzmann's constant

    Force_x = arrays with N zeros   #initiate forces to be returned
    Force_y = arrays with N zeros   #The arrays record total forces
                                    #on each particle

    for loop, i from 0 up to N:     #loop over all N particles

        Force_x[i] -= gamma * vx[i] #The damping force by Langevin dynamics
        Force_y[i] -= gamma * vy[i]

    ###### Remove this part to recalculate the force if a time step is revoked #####
        Rx, Ry = gauss(1)           #Generate 2 Guassian random number
                                    #centered at 0 with std = 1
        Force_x[i] += sqrt(2*gamma*kB*T) * Rx   #Apply the random collision forces
        Force_y[i] += sqrt(2*gamma*kB*T) * Ry
    ################################################################################
        for loop, j from 0 up to N:
            if i != j:              #avoid interaction with the particle
                                    #itself
                dx = x[i] - x[j]    #calculate the x distance
                dy = y[i] - y[j]    #calculate the y distance

                #Apply the force formula here, which is derived from the
                #'Lennard-Jones potential'

                f = 48/((dx**2 + dy**2)**7) - 24/((dx**2 + dy**2)**4)

                Force_x[i] += f * dx     #sum the total force on each particle
                Force_y[i] += f * dy

    return Force_x, Force_y         #return 2 N-force-arrays
```

Then, I present you the function to calculate the forces without the random process. The procedure is basically the same, to save space, I will not fully show them.

```
def Force_no_random(x, y, vx, vy, gamma, kB, T, N):
    ### Do stuff ###
    return Force_x, Force_y         #return 2 N-force-arrays
```

And there are functions to calculate potential and kinetic energies, they are relatively simple and straightforward, so I will not show them fully.

```
def Potential_energy(x, y):
    ### Do stuff ###
    return u_energy
```

```
def Kinetic_energy(vx, vy):
    ### Do stuff ###
    return k_energy
```

Finally, the function to generate 2-D Gaussian random numbers centered at zero. This is fully taken from the textbook.

```
def gauss(sigma):
    r = sqrt(-2*sigma*sigma*log(1-random()))
    theta = 2*pi*random()
    x = r*cos(theta)
    y = r*sin(theta)

    return x, y
```

4.2 Temporal loop, Modified Verlet Method & Box Boundary Condition

First, I initial the N-molecule system and set all the constants to 1, except $\tau = 10$. So far, my code can only deal with N to be a square number (1, 4, 9...). They are evenly separated in the square box. There is a stability reason behind why I keep this initial position, I will discuss this issue further in the latter part of the report. However, I do generalize the initial velocities of the particles. Some of this part follows Lab6, so I will keep it simple.

```
#set N and all constants
N = # A square number
gamma = 1   #viscosity
kB = 1      #Boltzmann's constant
Tmax = 100  #Initial temperature
T = Tmax
tau = 10    #time constant

### Do stuff here, initial the grid ###

#set time step size
h = 0.01
tpoints = arange(0, 80, h)

#set the initial velocities to be random numbers between (-0.5, 0.5)
vx = rand(N)-0.5
vy = rand(N)-0.5

initial vx_half and vy_half to be zero N-array

#calculate the initial force
dvx, dvy = Force_random(x_initial, y_initial, vx, vy, gamma, kB, T, N)
#calculate the initial v_half
vx_half += 0.5 * h * dvx
vy_half += 0.5 * h * dvy
```

```
#Compute the initial energy for Markov Chain method
totalE = Potential_energy(x, y) + Kinetic_energy(vx, vy)
```

We are done with initializing, the temporal loop starts here.

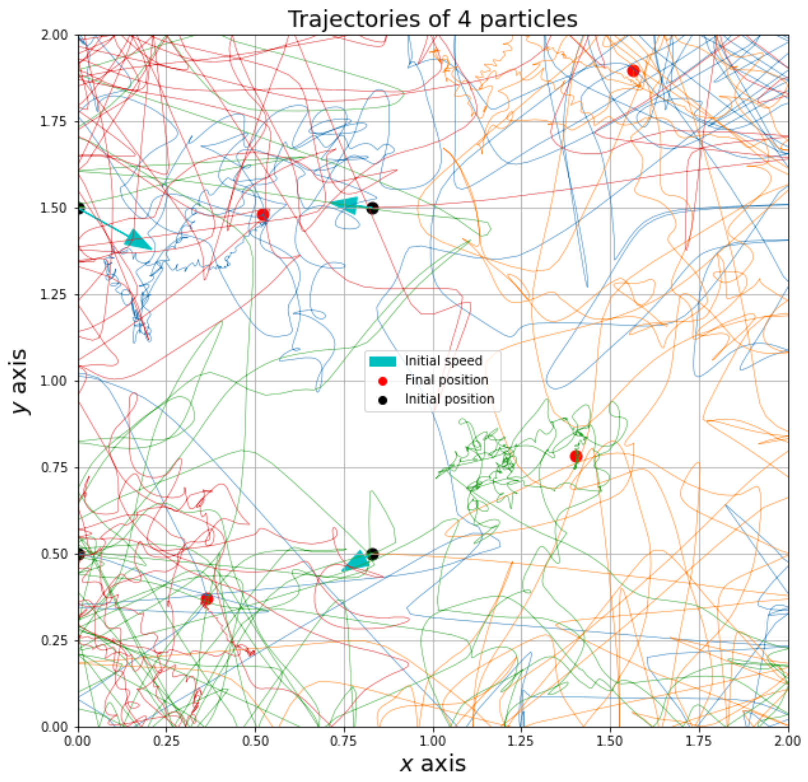
```
for t in tpoints:

    T = Tmax * exp(-t/tau)      #Simulated annealing

    for i in range(N):          #loop through all particles
    ### Box boundary condition ###
        if x[i] >= Lx:
            x[i] = Lx           #Relocate the particle at the boundary
            vx[i] *= -1         #Flip the corresponding velocity component
            vx_half[i] *= -1    #Don't forget v_half!
        elif x[i] <= 0:
            x[i] = 0
            vx[i] *= -1
            vx_half[i] *= -1

        if y[i] >= Ly:
            y[i] = Ly
            vy[i] *= -1
            vy_half[i] *= -1
        elif y[i] <= 0:
            y[i] = 0
            vy[i] *= -1
            vy_half[i] *= -1
    ##############################

    #update the position by v_half from previous step
        x[i] += h * vx_half[i]
        y[i] += h * vy_half[i]

    #First, calculate the force with random process
    kx, ky = h * Force_random(x, y, vx_half, vy_half, gamma, kB, T, N)
    #Update the velocities according to the force from random process
    vx = vx_half + 0.5 * kx
    vy = vy_half + 0.5 * ky

    #Markov Chain method
    old_totalE = totalE         #save previous energy value
    U = Potential_energy(x, y)  #Potential
    K = Kinetic_energy(vx, vy)  #Kinetic
    totalE = U + K              #total
    delta = totalE - old_totalE #Difference

    if random() > exp(-delta/(kB*T)):       #Determine revoke or not
        #reverse the velocities
        vx = vx_half - 0.5 * kx
        vy = vy_half - 0.5 * ky
        #recalculate the force without the random process
        kx, ky = h * Force_no_random(x, y, vx_half, vy_half, gamma, N)

        #update the corrected velocities
        vx = vx_half + 0.5 * kx
        vy = vy_half + 0.5 * ky

        #recalculate the corrected energy
        U = Potential_energy(x, y)  #Potential
        K = Kinetic_energy(vx, vy)  #Kinetic
        totalE = U + K

    #Update v_half
    vx_half += kx
    vy_half += ky
#end of temporal loop

plot energy
plot grids, trajectories, etc.
```

This is the end of my very own immature Markov Chain modified Verlet method. Hope you find it interesting!

5 Numerical Simulation Results & Discussions

5.1 Initial Conditions and Stability Concerns

As I have mentioned in the pseudo code part, I wanted to randomize the initial positions of particles to generalize the simulation. However, not having the particles evenly separated causes some serious stability issues.
This is caused by the nature of Lennard-Jones potential. It has r^{12} and r^6 sitting on the denominator, so the potential function is extremely sensitive to the distances between particles. If two particles get too close to each other ($r < 1$), the potential function will blow up easily.
To find this stability limit, I manually set the distances between 4 particles. I find the distance limit is around $r = 0.82 \sim 0.83$. The following are plots showing a stable case and an unstable one.
Just some captions on the plots, the black dots represent the initial positions of the particles, the red dots represent the final positions when the system is fully cooled. The blue arrows represent the random initial velocities of the particles, and the colored lines are their trajectories. On the energy plots, I plot potential, kinetic, and total energy of the system together in one graph against the temperature.



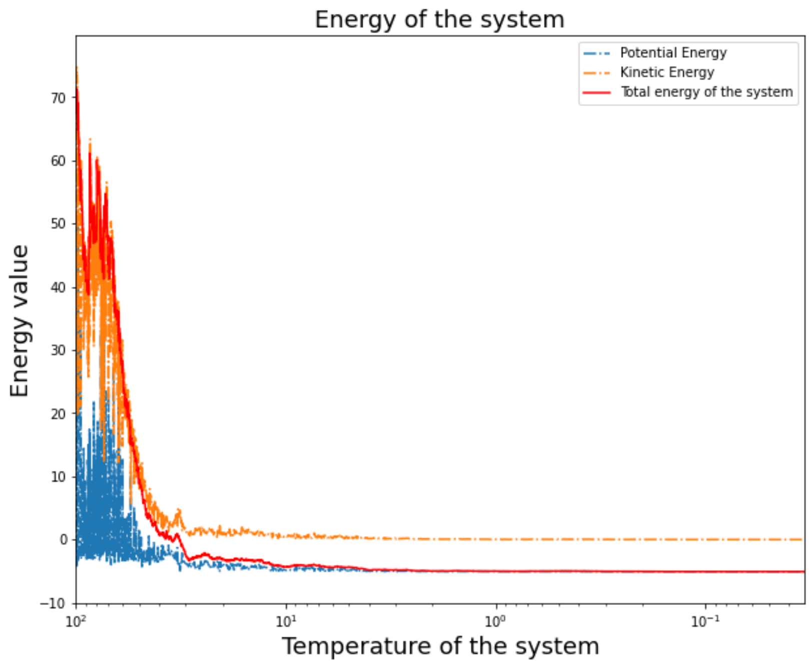
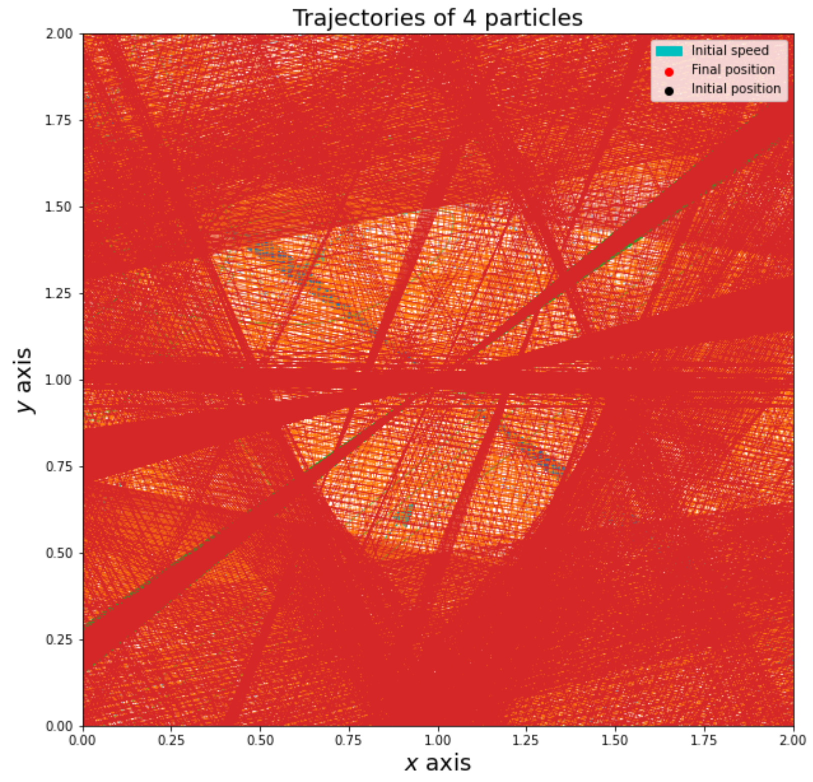


Figure 1: Stable case, minimum initial distance $r_{min} = 0.83$



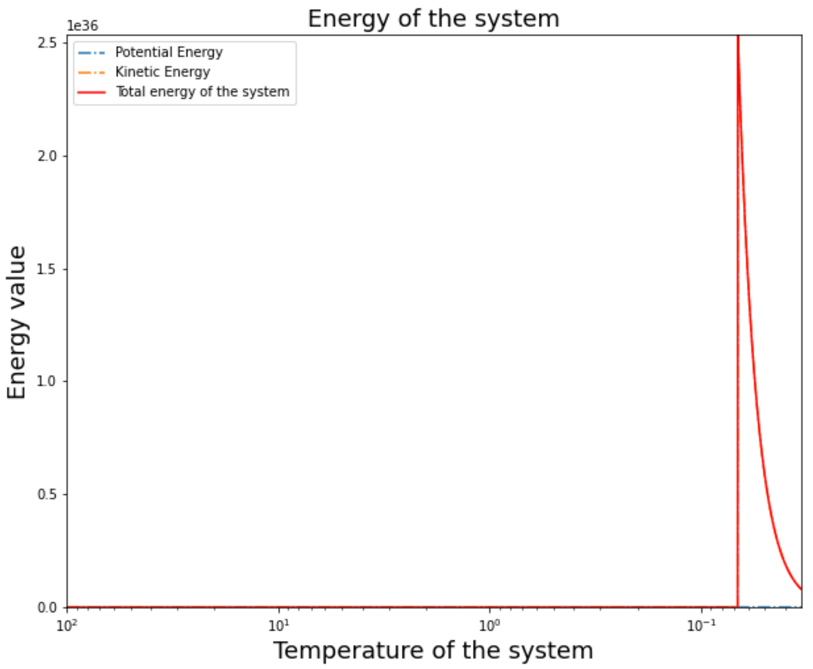
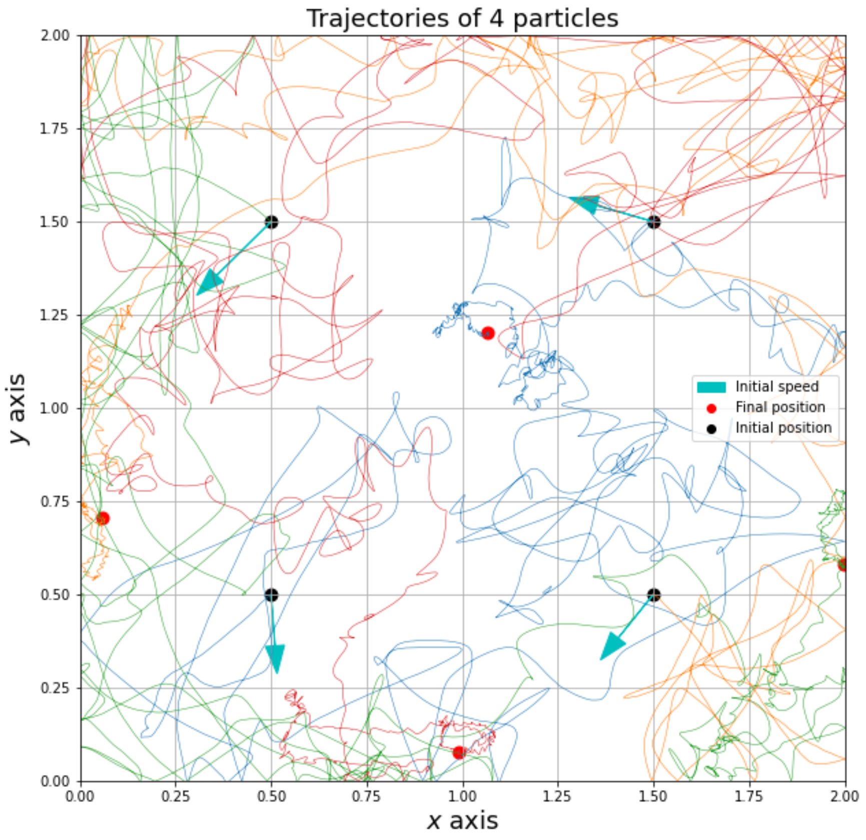


Figure 2: Unstable case, minimum initial distance $r_{min} = 0.82$

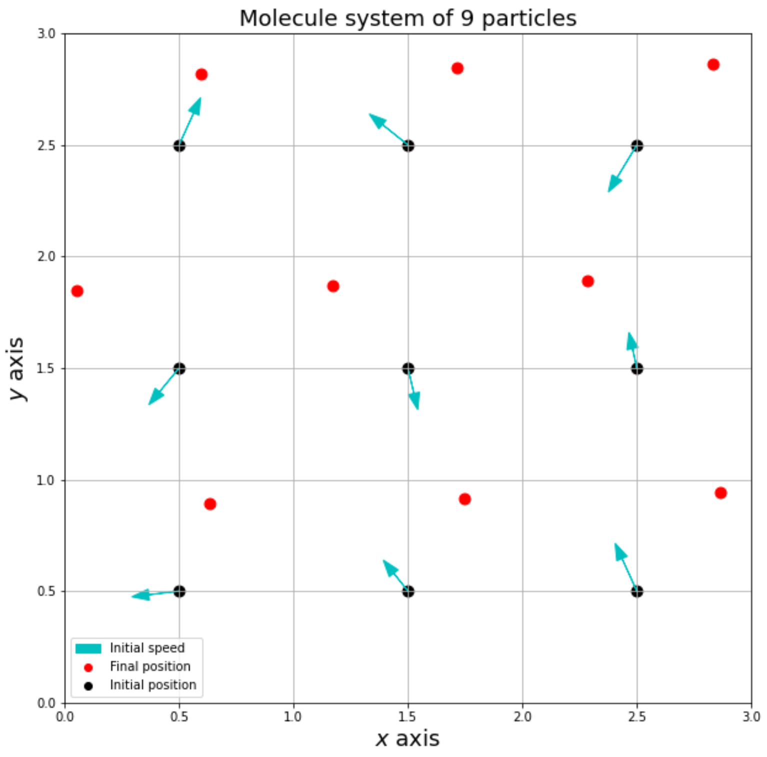
By comparing the stable and unstable case, we can see that the energy of the stable system decreases smoothly as the temperature cools down. While in the unstable case, the energy blows up randomly at some point due to the denominator r^{12}. The trajectory plot becomes a complete chaos. To eliminate this instability, from now on, the particles will be evenly separated in the box ($r_{min} = 1$) as the initial positions. The initial velocities will still be randomly generated to give the system some sort of generalization.

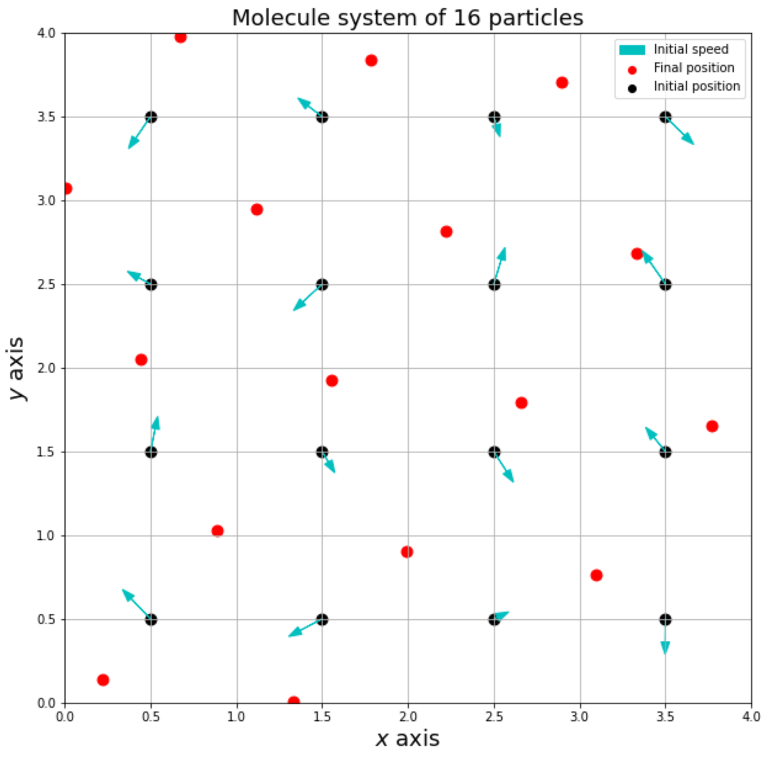
5.2 Discovery of the Ground State, More Plots with More Particles

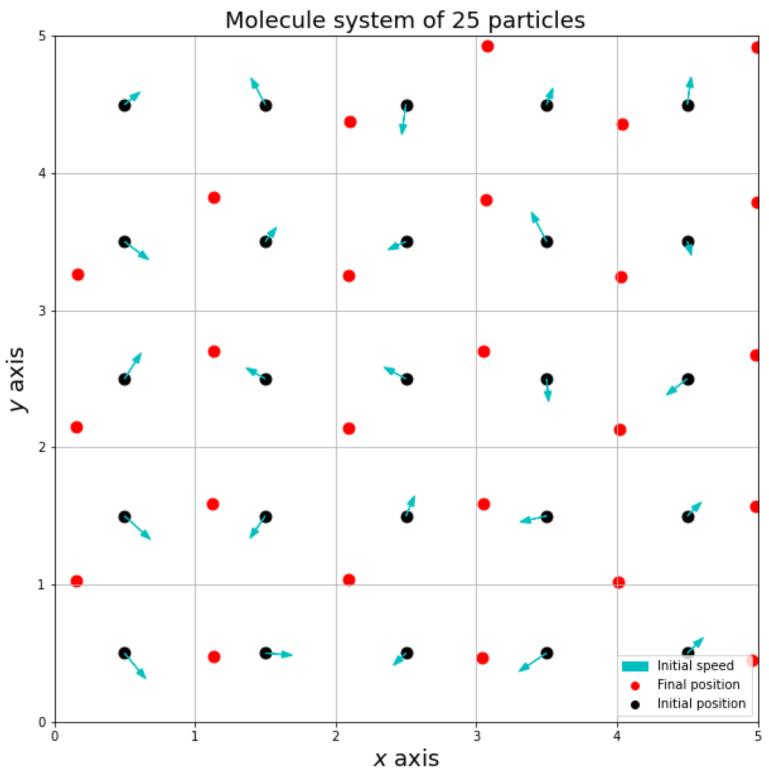
If you look closely enough of the stable trajectory plot from the previous section. Although the trajectories of particles look very chaotic, exactly what we expect from microscopic particles' behavior, random and chaos. However, their final position undergone the simulated annealing tells a different story, they seem to rest in a diamond shape pattern. And this is not a coincidence!
Here is another trajectory plot of 4 molecules system, with evenly separated initial distances and random initial velocities:



The diamond appears again! How interesting is it. So I present you the plots of 9, 16, and 25 particles system undergoing simulated annealing. For clarity of the plots, I omit the chaotic trajectories for higher number molecule systems, just to show you the initial and final positions of the particles.


Molecule system of 9 particles
y axis
x axis
Initial speed
Final position
Initial position

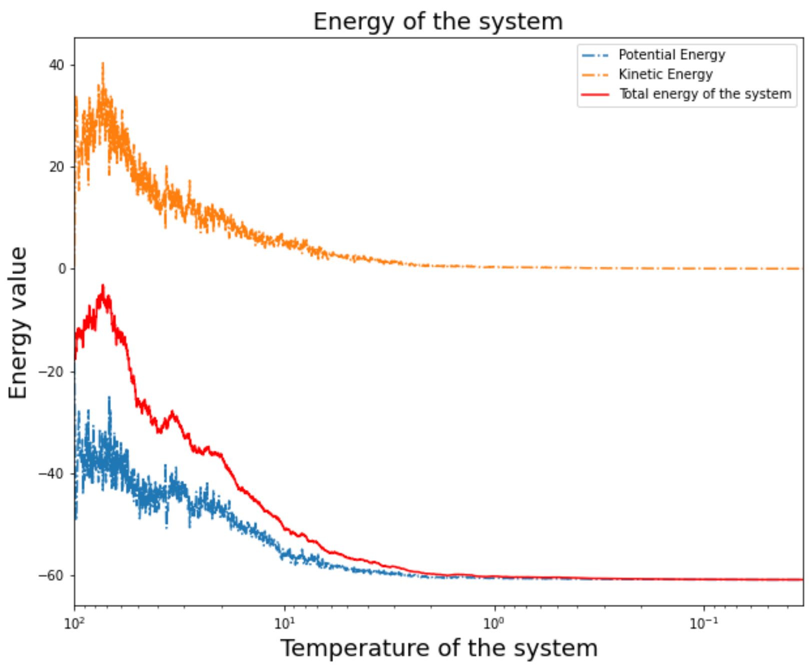

Molecule system of 16 particles
y axis
x axis
Initial speed
Final position
Initial position



The molecules undergone simulated annealing form spontaneously into these beautiful lattice-like pattern. I will call this pattern as the ground state of the system.

5.3 Some Comments on the System's Energy

The following is a plot of energies of a 25-particle system:



As we can tell from the energy plot:

- The kinetic energy has a high initial value due to the random initial velocities. When the system cools down, it gradually drops due to the damping effect of Langevin dynamics and the Markov Chain method. The particles slowly lose their velocity, and get 'frozen'.
- The potential also gradually drops. This represents the process which particles find the way to the ground state.
- The total energy starts at an intermediate value, as the kinetic energy dies out, it converges to the potential energy.

5.4 Distances between Neighbouring Particles at the Ground State

I wrote a function to calculate the average distance between particles and their 2 neighbours at the ground state, which is also the side length of a lattice. Although a particle can have at most 6 neighbours, but considering the base case with only 4 particles, the function will count only the closest 2 neighbours.
Here is a table for all the cases I have tested:

Number of particles	Average distance
4	1.121483688386607
9	1.1156818133799182
16	1.1133598212142801
25	1.1133967274792287

We can tell that despite the numbers of particles are different, the average distances agree each other and center at around 1.115. What does this number mean? Let's take a look at the formula of Lennard-Jones potential:

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

If we take $\epsilon = 1$ and $\sigma = 1$ as I have done in the code, the r value to minimize the potential function yields:

$$r = \sqrt[6]{2} \simeq 1.122$$

This analytic minimum agrees with our simulation result.

6 Conclusion

Through out this project, I developed a modified Verlet method with simulated annealing governed by Langevin dynamics. I applied the method to a 2-D multi-particle system. I investigated efficiency and stability concerns associated with the method. By cooling the system to close-zero temperature, I discovered the ground state of this 2-D system. The length of lattices formed by fully cooled particles agrees with the theoretical minimum of Lennard-Jones potential. To demonstrate the cooling process straightforwardly, I made 2 animations for 4-particle and 9-particle systems, please enjoy!