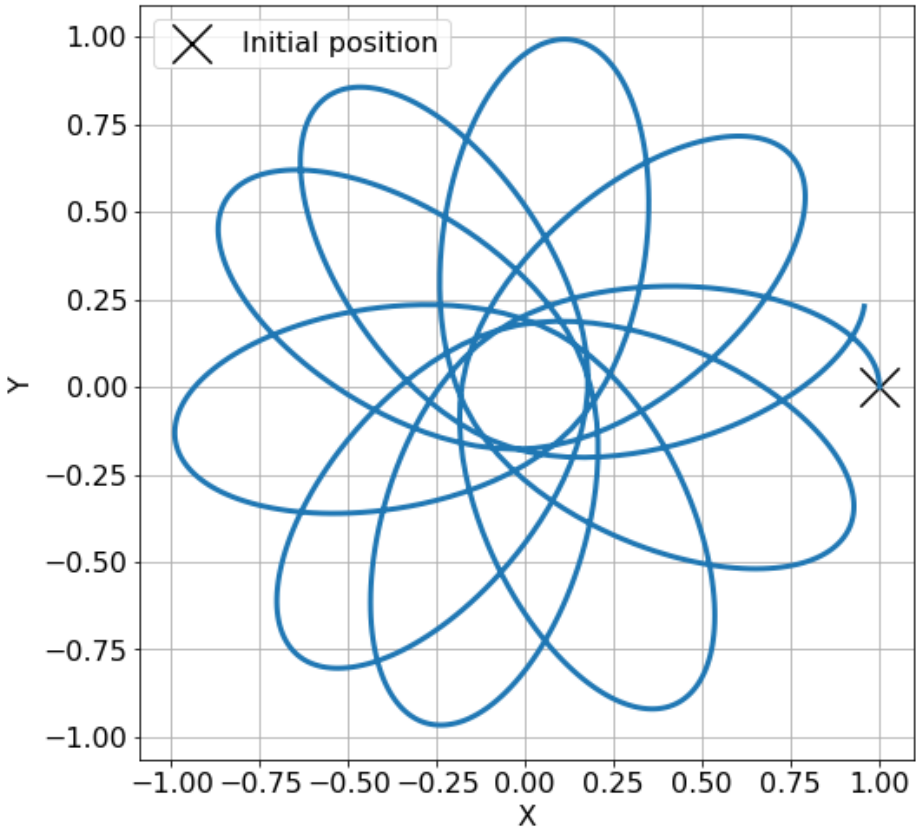
Lab6 Report

Zirui Wan (Question 1, 2), Rundong Zhou (Question 3)

October 24, 2020

1 Question 1

1.1 Q1 (b)



From part (a), it's derived that the second-order ODEs of the x and y position of a ball-bearing under the effect of the gravitational pull of a rod could be

expressed as:

$$\frac{d^2x}{dt^2} = -GM\frac{x}{r^2\sqrt{r^2+L^2/4}} \frac{d^2y}{dt^2} = -GM\frac{y}{r^2\sqrt{r^2+L^2/4}} \tag{1}$$

where $r = \sqrt{x^2+y^2}$.

Solving these two equations using the fourth-order Runge-Kutta method, the orbit of the ball-bearing is calculated and plotted above. You can see that, as expected, the ball-bearing has a precessing orbit, where the major axis of the orbit is rotating in the x-y plane.

2 Question 2

2.1 Q2 (a): some derivation

In this problem, we are trying to solve the ODEs associated with particles under the effect of the Lennard-Jones potential:

$$V(r) = 4\epsilon[(\frac{\sigma}{r})^{12} - (\frac{\sigma}{r})^6] \tag{2}$$

The corresponding force could be derived as:

$$F(r) = -\frac{dU(r)}{dr} = 48\epsilon[(\frac{\sigma^{12}}{r^{13}}) - (\frac{\sigma^6}{2r^7})] \tag{3}$$

Here the first term in the bracket is the attractive term, while the second term is the repulsive term. The x and y components are:

$$F_x(r) = 48\epsilon[(\frac{\sigma^{12}}{r^{13}}) - (\frac{\sigma^6}{2r^7})]\frac{x}{r} \tag{4}$$

$$F_y(r) = 48\epsilon[(\frac{\sigma^{12}}{r^{13}}) - (\frac{\sigma^6}{2r^7})]\frac{y}{r} \tag{5}$$

where $r = \sqrt{x^2+y^2}$.

2.2 Q2 (b): pseudo code and trajectories

The pseudo code I wrote for this problem is below:

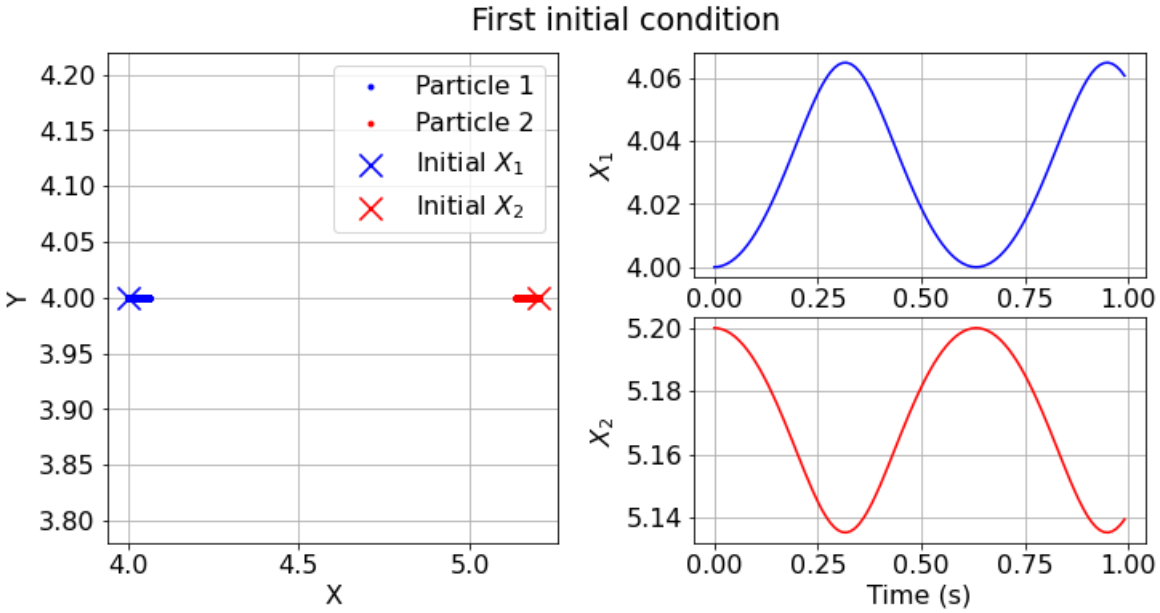
```
#define a function that calculates accelerations using positions of particles
def f(r, t):
    """ Function calculate first-order derivatives for the Verlet algorithm.
    INPUT:
        r[floats]: array of the state of the system: 4 positions (x1, y1, x2, y2)
        t[float]: time
    OUTPUT:
        fr[floats]: array of accelerations for 2 particles along x and y directions
```

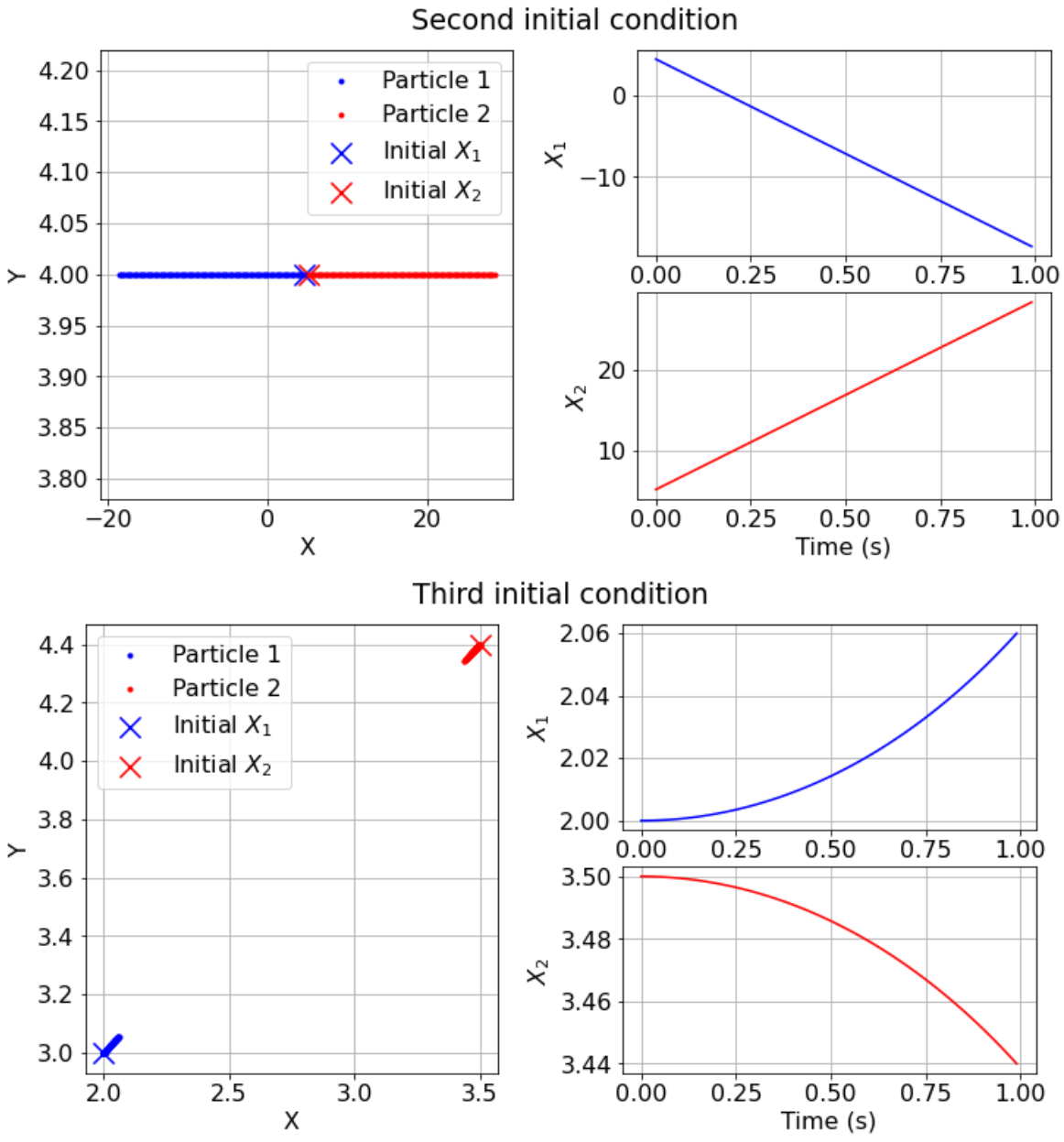
```
    """
    # extract x, y, vx, vy information from the state array
    x1, y1, x2, y2 = r[0], r[1], r[2], r[3]

    # calculate distances
    dx1 = x1 - x2
    dy1 = y1 - y2
    dx2 = -dx1
    dy2 = -dy1
    rr1 = sqrt(dx1**2 + dy1**2)
    rr2 = sqrt(dx2**2 + dy2**2)

    # calculate accelaration on particle 1
    fvx1 = acceleration_1_along_x*dx1/rr1
    fvy1 = acceleration_1_along_y*dy1/rr1
    # calculate accelaration on particle 2
    fvx2 = acceleration_2_along_x*dx2/rr2
    fvy2 = acceleration_2_along_y*dy2/rr2

    fr = array([fvx1, fvy1, fvx2, fvy2])
    return fr

#define array for time points
tpoints = array from 0 to 1 with step size h=0.01
# initialize system state with initial positions
r = [initial x1, initial y1, initial x2, initial y2]
# initialize velocity vector with initial velocities
v = [initial velocities]

vhalf = v + h/2*f(r, t0)   # first step of Verlet algorithm
# start a for loop
for time in tpoints:
    r = r + h*vhalf           # update position
    k = h*f(r, t+h)            # calculate k
    vth = vhalf + k/2       # calculate velocity for next step
    vt32h = vhalf + k       # calculate half-step velocity for next iteration
    vhalf = vt32h             # overwrite vhalf for next iteration

    # save outputs
    append r[0] to x1points
    append r[1] to y1points
    append r[2] to x2points
    append r[3] to y2points

# visualize y vs x for both particles
plot(x1, y1)
```

```
plot(x2, y2)
```

Using the code attached to this report, the trajectories of the 2 particles for these 3 initial conditions are simulated and plotted:

1. $r_1 = [4, 4]$, $r_2 = [5.2, 4]$
2. $r_1 = [4.5, 4]$, $r_2 = [5.2, 4]$
3. $r_1 = [2, 3]$, $r_2 = [3.5, 4.4]$





In order to better visualize the motion of the two particles, besides the trajectories, I also plotted the x component of the position vector $\vec{r}$ as time series for both particles.

2.3 Q2 (c): discussion

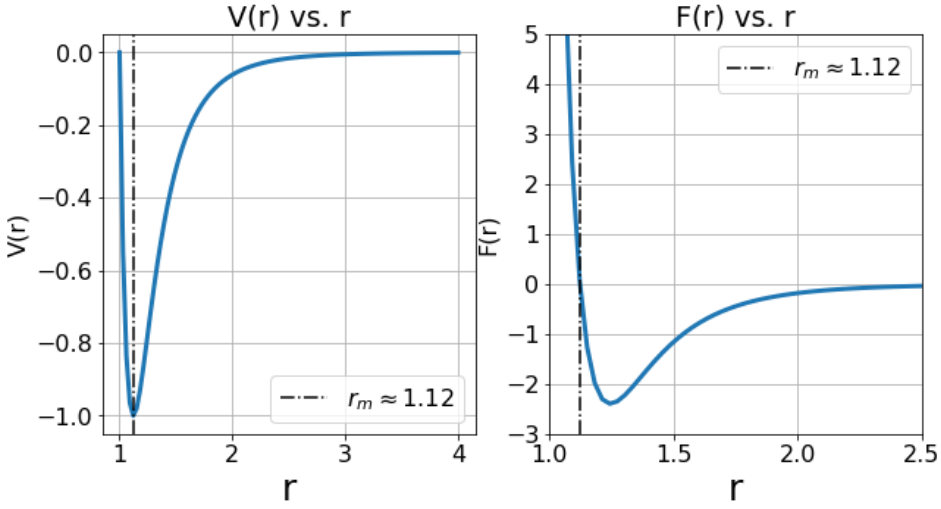
You can see that:

1. The oscillatory solution comes from the first initial condition (IC), which could be clearly seen from the subplots of time series for both particles. From energy perspective, this occurs because the initial displacement is just around the equilibrium position of the Lennard-Jones potential (see below). And potential energy and kinetic energy will just transfer to each other, as they do in a simple harmonic oscillator.

2. For the second IC, the distance between two particles is now smaller than the first IC. As a result, the two particles now both experience the repulsive term in the Lennard-Jones potential and they are quickly pushed away from each other.

3. By contrast, if the two particles are too far away from each other, they will experience very weak attractive force from each other. As a result, they will be slowly pulled together. However, since the attractive force increases as the distance decreases, it will be an accelerated motion.

To quantitatively investigate this, I plotted the diagram of the Lennard-Jones potential and the force caused by it. You can see that the equilibrium position of the L-J potential is roughly 1.12 for this problem. And indeed, distance between two particles for the first IC is 1.2, which is just around it. For the second IC, the initial distance between two particles is only 0.7. From the figure you can see that the repulsive force will be extremely large, due to the dependence on $\frac{1}{r^{12}}$. For the third IC, initial distance between two particles is 2.05, where the force due to L-J potential would be extremely small (about 0.1) compared to the repulsive force in the second IC.



3 Question 3

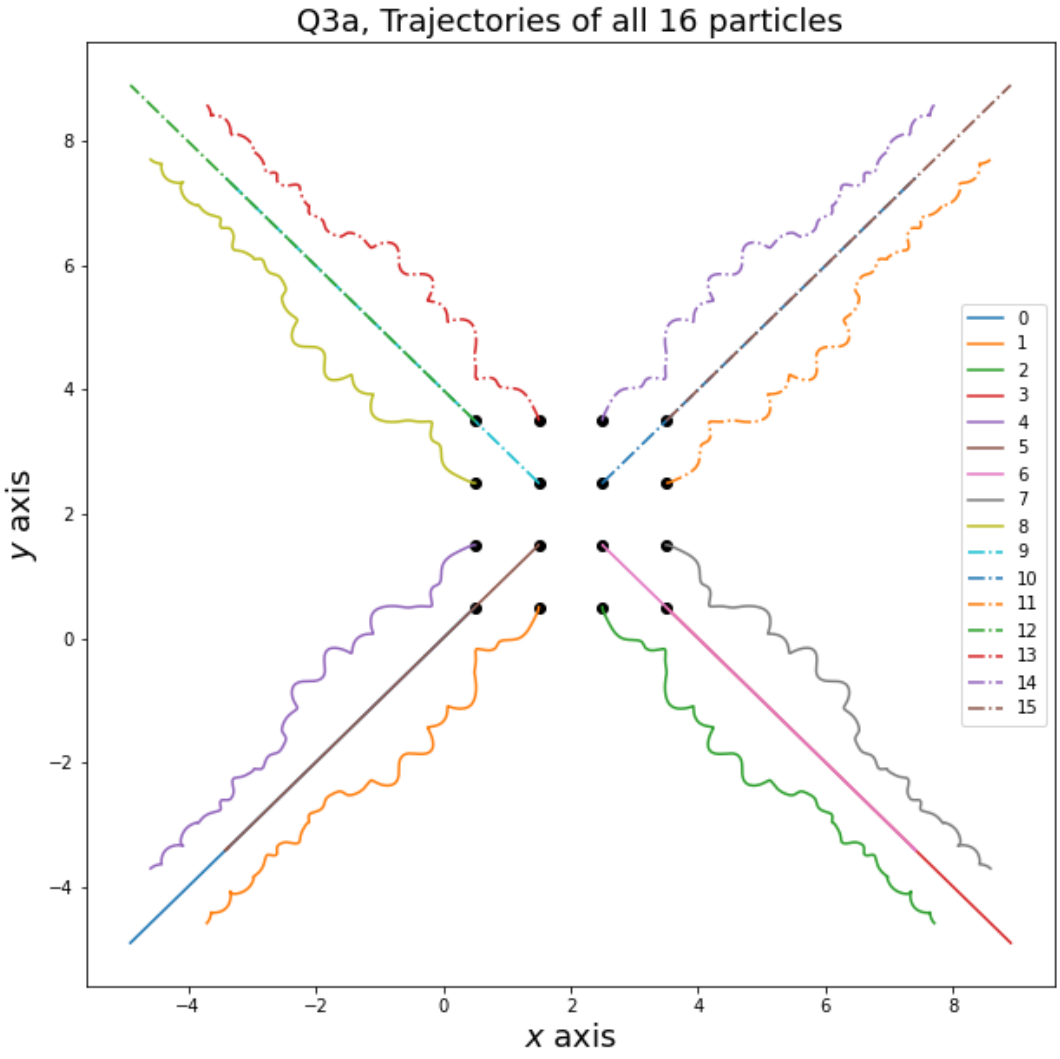
3.1 Q3c, Pseudo code and Plot of the trajectories of 16 particles

```
#Define a function to take x and y position arrays
#and calculate the force on each particle

def Force(x, y):    #x, y are arrays with N elements, corresponding
                    #to the position of each particle

    Force_x = arrays with N zeros   #initiate forces to be returns
    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
        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

Create the initial condition grid according to the lab handout
#The initial condition is stored in 2 N-arrays:
x_initial
y_initial

x_points = empty array              #Create 2 arrays to record the trajectory
y_points = empty array              #they are arrays of arrays
for loop, i from 0 up to N:
    x_points append an empty array  #Create an empty array for each particle
    y_points append an empty array

    x_points[i] append x_initial[i] #update the initial condition to the trajectory
    y_points[i] append y_initial[i] #array
```

```
#set up the iteration points
h = 0.01
t_points = array, from 0 to 10, with step-size of h

#####Verlet method starts here#####
#Create x, y position arrays for iteration, assign it with initial conditions
x = x_initial
y = y_initial

#Initial velocity arrays for Verlet method
vx = array with N zeros             #v(t+h)
vy = array with N zeros

vx_half = array with N zeros        #v(t+0.5h)
vy_half = array with N zeros

#since m=1, the force is just the acceleration
kx, ky = Force(x_initial, y_initial)

#Calculate the first v(t+0.5h)
for loop, i up to N:
    vx_half[i] += 0.5 * h * k[i]
    vy_half[i] += 0.5 * h * k[i]

#iteration starts
for loop, t in t_points:
    for loop, i up to N:
        #calculate the position r(t+h) for each particle
        x[i] += h * vx_half[i]
        y[i] += h * vy_half[i]

        #update the result to trajectory array
        x_points[i] append x[i]
        y_points[i] append y[i]

    kx, ky = h * Force(x, y)  #calculate the acceleration with r(t+h)

    for loop, up to N:
        #update v(t+h) and v(t+3/2 h)
        vx = vx_half + 0.5*h*kx
        vy = vy_half + 0.5*h*ky

        vx_half += h*kx
        vy_half += h*ky

#Plot the trajectory
```

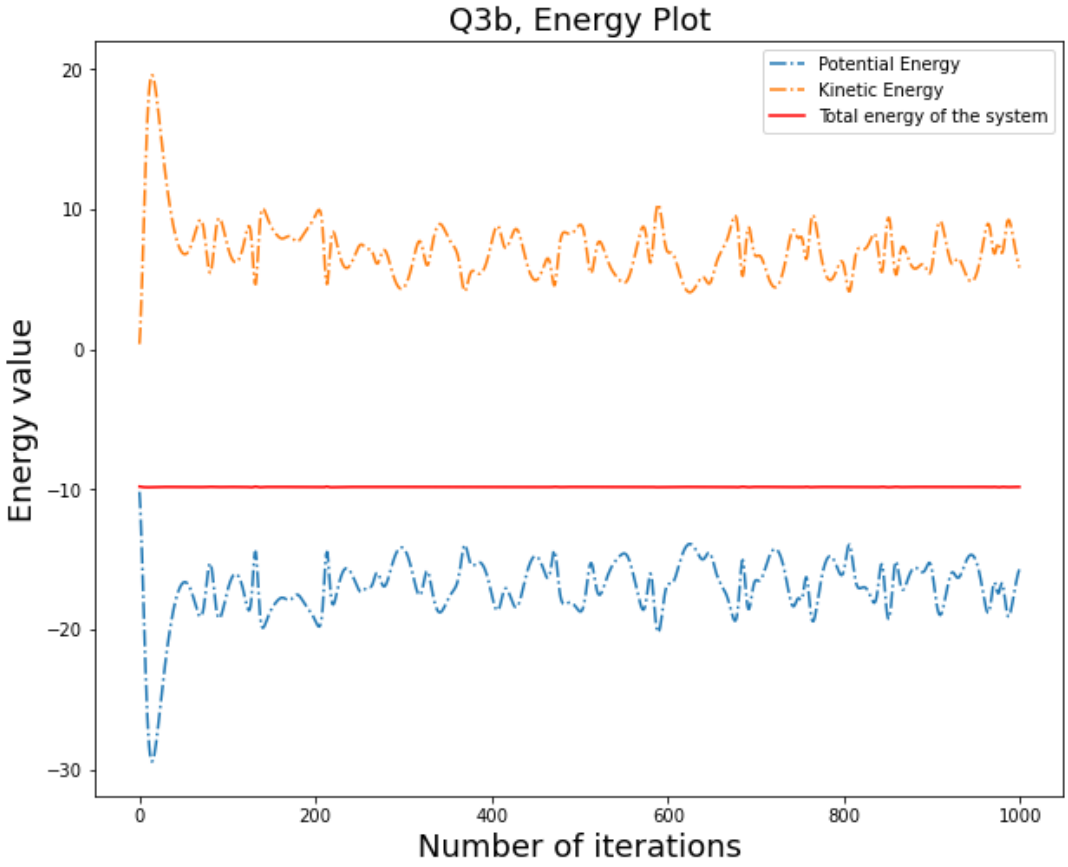
```
plot x_points, y_points
```

The plot of the trajectories of 16 particles is shown below:
Please note I omit all the units in the axes, since all the constants are normalized to 1.

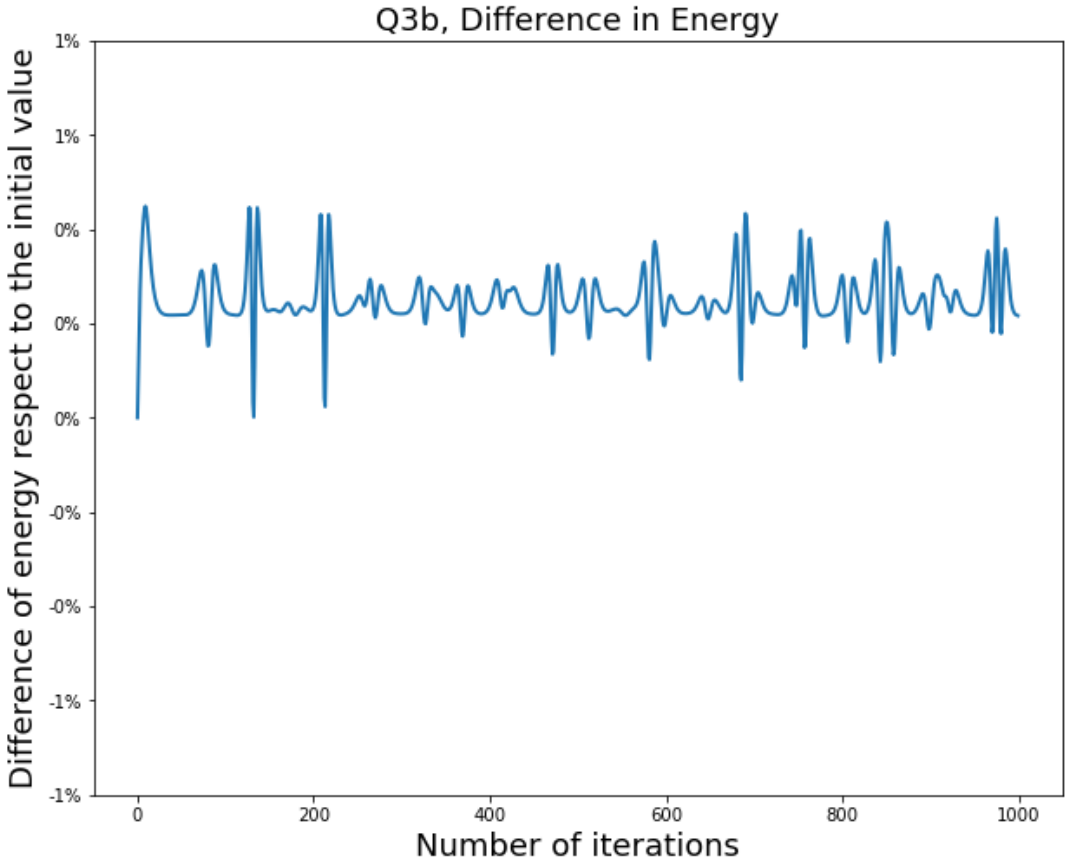


3.2 Q3b, Energy conservation of the system

I plot the potential energy, kinetic energy, and the total energy of the 16 particle system. As we can see from the graph, the total energy shows a straight line, which indicates the total energy of the system is conserved.
Please note I omit the units in the y-axis, since all the constants are normalized to 1.



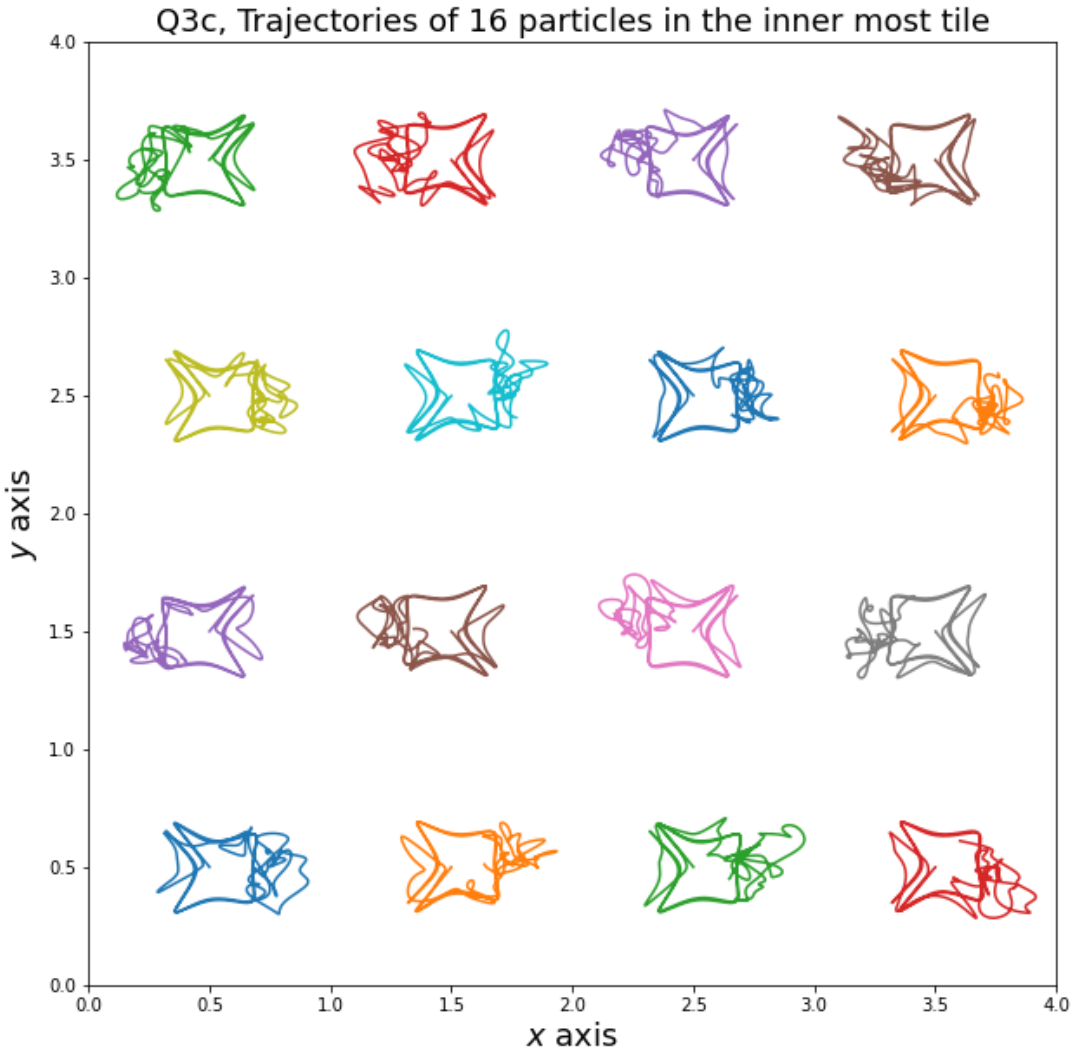
I also plot the total energy's difference from initial value throughout the simulation progresses.



As we can tell from the graph, the total energy of the system is conserved within 1% as the simulation progresses.

3.3 Q3c, Trajectory of 16 particles with periodic boundary conditions

The following is the plot of 16 particles with periodic boundary conditions: Please note I omit the units in the axes, since all the constants are normalized to 1.



Due to the 8 imaginary tiles, the particles no longer travel away like in free space. Instead, they wiggle around their initial position. This is because that they are trapped in the potential well due to all the infinite particles around them.