PHY407 Lab6

Genevieve Beauregard (1004556045)
Anna Shirkalina (1003334448)

Oct 2020

Question 1

We are asked consider the following ODEs (from part (a)):

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

$$\frac{\mathrm{d}^2y}{\mathrm{d}t^2} = -GM\frac{y}{r^2\sqrt{r^2+L^2/4}} \tag{2}$$

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

We work in units such that G = 1, and M = 10, L = 2, and we utilise the initial conditions: $(x, y) = (1, 0)$ and $(v_x, v_y) = (0, 1)$. We use the RK4 method to solve the above sets of ODEs.

We use the techniques shown in Section 8.3 of the textbook. We let $v_x = \frac{\mathrm{d}x}{\mathrm{d}t}$ and $v_y = \frac{\mathrm{d}y}{\mathrm{d}t}$.

We solve the ODEs simultaneously and plotted the numerically resolved x and y coordinates. Our plot of x against y can be seen in Fig 1.

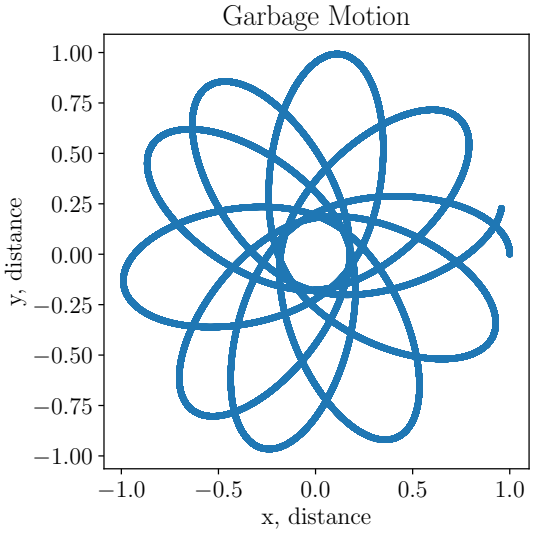

Figure 1: Numerically solved motion of simplified space garbage orbiting.

Question 2

a)

We are asked to consider a 2-dimensional molecular dynamics simulation with 2 particles. We set $\sigma = 1$, $\epsilon = 1$. Motion is solely influenced by

$$V(r) = 4\epsilon\left[(\frac{\sigma}{r})^{12} - (\frac{\sigma}{r})^{6}\right] = 4\epsilon\left[(\frac{\sigma}{\sqrt{x^2+y^2}})^{12} - (\frac{\sigma}{\sqrt{x^2+y^2}})^{6}\right] \tag{3}$$

Where $x = (x_2 - x_1)$ and $y = (y_2 - y_1)$ distances.

In terms of coordinates

$$V(x_1, x_2, y_1, y_2) = 4\epsilon\left[\frac{1}{((x_2-x_1)^2+(y_2-y_1)^2)^6} - \frac{1}{((x_2-x_1)^2+(y_2-y_1)^2)^3}\right] \tag{4}$$

$$\mathbf{F} = m\mathbf{a} = \mathbf{a} = -\boldsymbol{\nabla}\mathbf{V} \tag{5}$$

as m = 1

We consider a set of particles as follows: x_i and y_i are their coordinate positions in terms of the x and y axis, v_{x_i} and v_{y_i} are their velocities likewise, and a_{x_i} and a_{y_i}.

For the acceleration of x-coordinates:

$$a_{x_1} = -\frac{\mathrm{d}V}{\mathrm{d}x_1} \tag{6}$$

$$= 12\left[-\frac{2(x_2 - x_1)}{((x_2 - x_1)^2 + (y_2 - y_1)^2)^7} + \frac{(x_2 - x_1)}{((x_2 - x_1)^2 + (y_2 - y_1)^2)^4}\right] \tag{7}$$

The other particle would just be the opposite sign by Newton's Third Law,

$$a_{x_2} = \frac{\mathrm{d}V}{\mathrm{d}x_1} \tag{8}$$

$$= 12\left[\frac{2(x_2 - x_1)}{((x_2 - x_1)^2 + (y_2 - y_1)^2)^7} - \frac{(x_2 - x_1)}{((x_2 - x_1)^2 + (y_2 - y_1)^2)^4}\right] \tag{9}$$

For the acceleration of the y-coordinates:

$$a_{y_1} = -\frac{\mathrm{d}V}{\mathrm{d}y_1} \tag{10}$$

$$= 12\left[-\frac{2(y_2 - y_1)}{((x_2 - x_1)^2 + (y_2 - y_1)^2)^7} + \frac{(y_2 - y_1)}{((x_2 - x_1)^2 + (y_2 - y_1)^2)^4}\right] \tag{11}$$

$$a_{y_1} = \frac{\mathrm{d}V}{\mathrm{d}y_1} \tag{12}$$

$$= 12\left[\frac{2(y_2 - y_1)}{((x_2 - x_1)^2 + (y_2 - y_1)^2)^7} - \frac{(y_2 - y_1)}{((x_2 - x_1)^2 + (y_2 - y_1)^2)^4}\right] \tag{13}$$

b)

In this section, we are asked to resolve a system with two particles, under the influence of the Lennard-Jones potential using the Verlet method. As we already have ODEs for acceleration of each component for each particle, we solve the odes simultaneously. Our pseudocode is as follows.

```
# initialize constants

# set initial r array = [r1, r2], where r1 and r2 are similarly
# 2 dim vectors with x, y position

# initialize time array and number of steps N

# Def f(r):
    # return np.array([[ax1, ay1], [ax2, ay2]], dtype=float), as per our found equations


# def v(v1,v2):
```

```
    # return np.array([v1,v2])


# # define the following helper functions for the verlet loop
# # Eqn corresponds to the equation number in the lab handout

# def Eqn8(r, v, h):
    # return eqn 8

# def Eqn9(r, h)
    # return eqn 9

# def Eqn10(v, k)
    # return eqn 10

# def Eqn11(v, k)
    # return eqn 10


# Obtain first step of v using equation 7 of handout, v =[v1,v2]

# # Initialize accumalator variables and populating with first step
# r1_array = [r1]
# r2_array = [r2]
# v1_array = [v1]
# v2_array =[v2]

# # Initialize accumalator variables for v_energy
# v1_energy = [[0,0]]
# v2_energy = [[0,0]]

# Set term to be updated
# v_energy = np.array([v1, v2])

# For loop over range(1, N):
    # update r array, r(t + h) array with equation 8
    # append new r values to r1_array and r_2 array


    # obtain k vector with equation 9

    # obtain v(t + h) with equation 10
    # append


    # obtain next step for v array, v(t + 3/2 h) with equation 11
```

```
    # append new v values to v_1 array and v_2 array


# Obtain r_1, r_2 and plot trajectories
```

We programmed this and our plots can be seen in Fig 2 to 4.

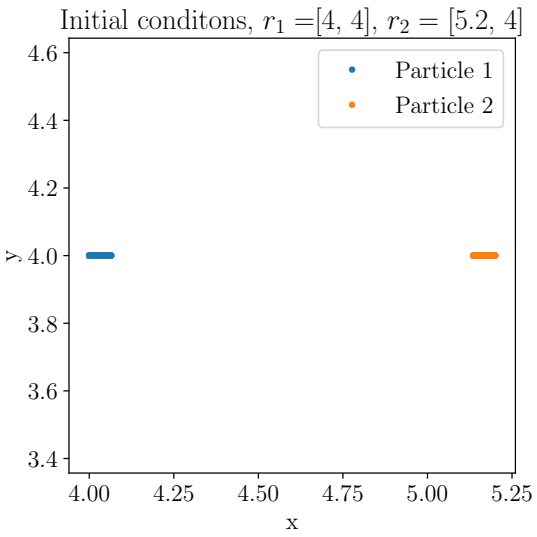

Figure 2: Question 2b(i), with the given initial conditions.

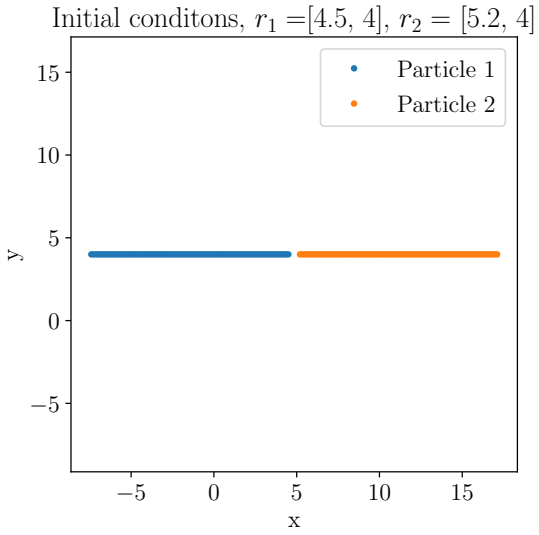

Figure 3: Question 2b(ii), with the given initial conditions.

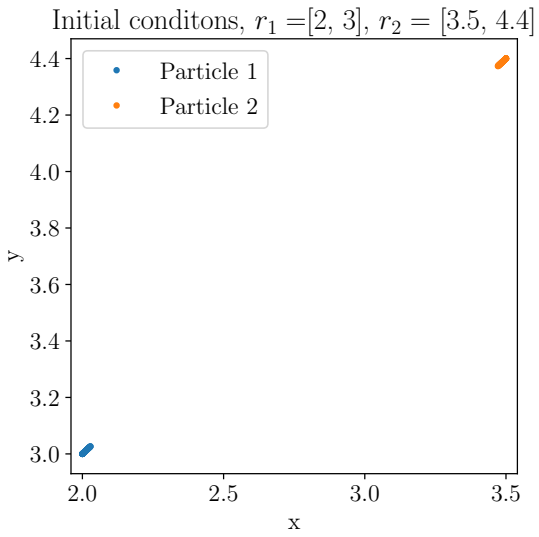

Figure 4: Question 2b(iii), with the given initial conditions.

c)

We are asked to find the initial condition that results in oscillatory motion. We do this in a fairly rudimentary manner – we plot each particles x and y components against time for each initial condition and look for oscillatory motion. We also plotted Total Energy, Kinetic energy and Potential energy of the system against time. All nine of the plots can be seen in the appendix, from Fig 12 to Fig 20 (we ran into image placement issues). We will only put relevant plots in this section.

The only oscillatory motion we observed was that of x-component with the initial conditions for 2i. We can see this plot in Fig 5. This makes sense when we look at its energy against time graph in Fig 6. We see that the systems entire energy starts at its full potential and oscillates within a potential well, while maintaining a constant energy.

We can contrast this with the energy of the systems created by the other initial conditions. For initial condition (iii), we see a system of ever decreasing potential in Fig 8 thus not lending itself well to oscillatory motion. For (ii) in Fig 7, we see that the systems entire energy lies in its kinetic energy – once again a signal its not oscillating.

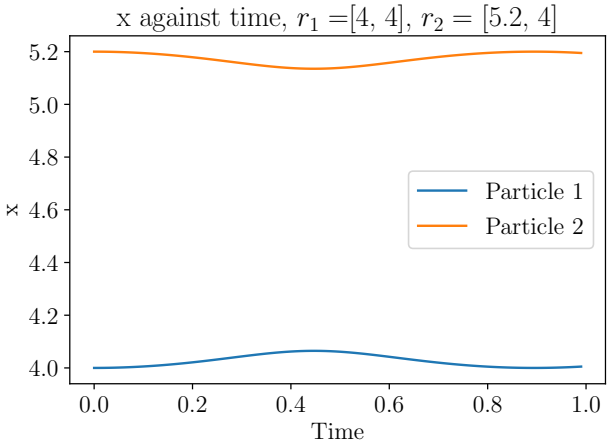

Figure 5: Question 2c initial condition bi, oscillatory motion observed in the x component.

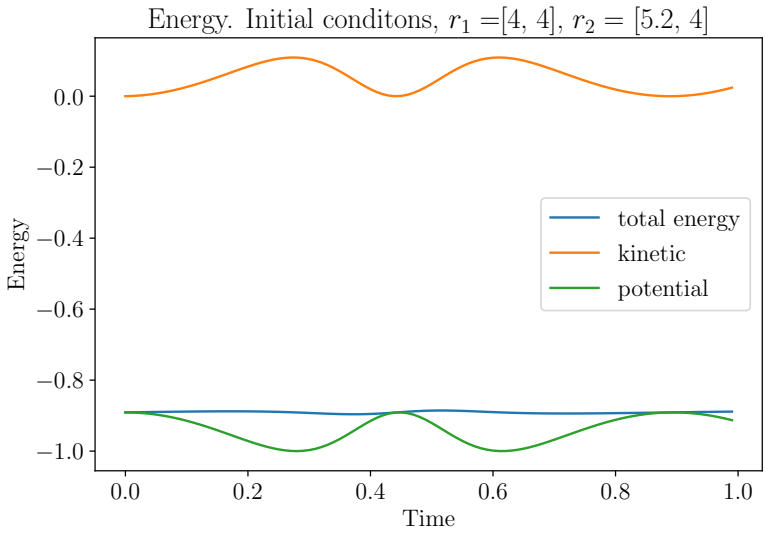

Figure 6: Question 2c initial condition bi, total Energy of the system observed against time.

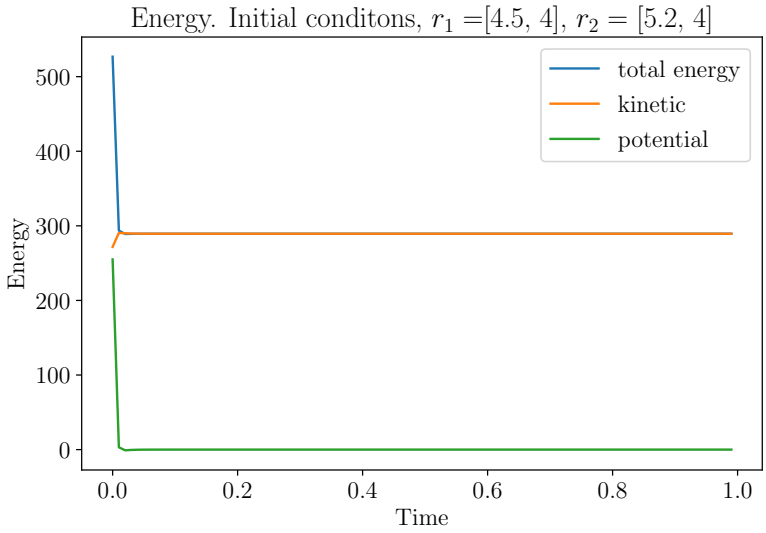

Figure 7: Question 2c initial condition bii, total Energy of the system observed against time.

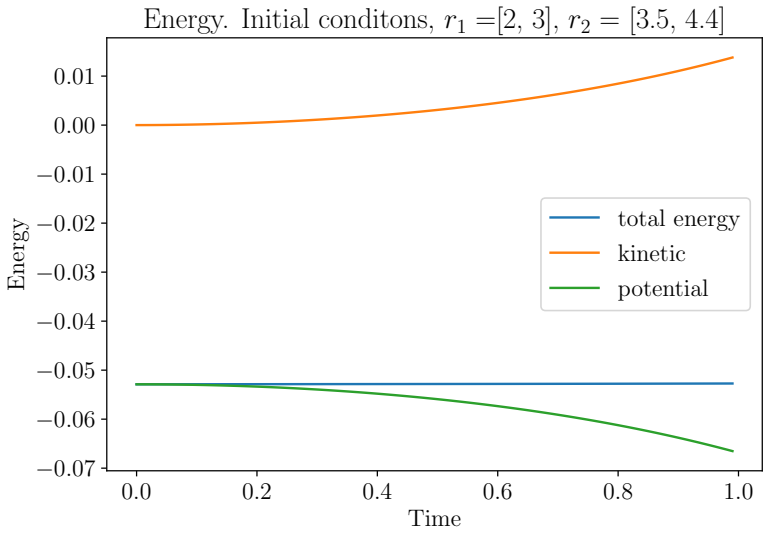

Figure 8: Question 2c initial condition bii, total Energy of the system observed against time.

Question 3

a)

```
N = # of particles
Lx = x - dimension
Ly = y - dimension
dx = Lx / sqrt(N)  dy = Ly / sqrt(N)
x_grid = arange(dx / 2, Lx, dx)
y_grid = arange(dy / 2, Ly, dy)
xx_grid, yy_grid = meshgrid(x_grid, y_grid)
x_init = flatten(xx_grid)
y_init = flatten(yy_grid)

# initialize velocity
velocity_init = np.zeros(N, 2)


time_array = np.arange(0, t_stop, dt)
num_steps = int(t_stop / dt)
velocity = np.zeros((num_steps, N, 2))
velocity_odd = np.zeros((num_steps, N, 2))
positions = np.zeros((num_steps, N, 2))


def acceleration_helper(x, y):
"""Helper function for the main acceleration function, returns the
acceleration for a given r = (x, y) between particles"""
denominator = (x ** 2 + y ** 2)
if denominator == 0:
    return 0, 0
a_x = 12 * x * ((-2 / (denominator ** 7)) + (1 / (denominator ** 4)))
a_y = 12 * y * (((-2) / (denominator ** 7)) + (1 / (denominator ** 4)))
return a_x, a_y


def acceleration(position_matrix):
    """Return the acceleration matrix at time t, given the position matrix at
    time t. The position matrix is of the form (i, 2) where
    i = num_particles.
    Return a matrix of size (i, 2)
    """

    n = number of particles
    initialize acceleration_array =
```

```
    for i in range(n):
        a_particle_i = np.zeros((n, 2))
        for j in range(n):
            r_x = position_matrix[j] - position_matrix[i]
            r_y = position_matrix[j] - position_matrix[i]
            a_x, a_y = acceleration_helper(r_x, r_y)
            a_particle_i[j]  append a_x and a_y
        sum over the a_particle_i
    return acceleration_array


# verlet loop
for t in range(time range):
    do equation 8, positions[t + 1] = positions[t] + dt * velocity_odd[t]
    do equation 9, k = dt * acceleration(positions[t + 1])
    do equation 10, velocity[t + 1] = velocity_odd[t] + 0.5 * k
    do equation 11, velocity_odd[t + 1] = velocity_odd[t] + k
```

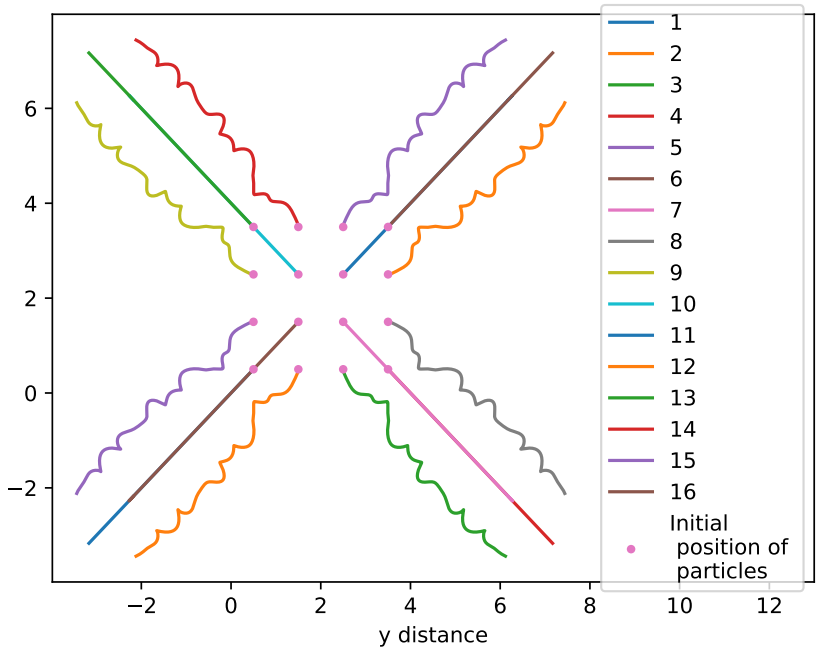

Figure 9: The trajectory of the 16 particles through time

b)

Code for this section can be found in `Lab06_Q3.py`. We can tell that energy is conserved because the total mechanical energy is constant through time. Not ewe have to scale the potential energy by a factor of $\frac{1}{4}$, because the potential energy per particle is half of the potential energy generated by a system of two particles and because the double for loop in the code that calculates the potential energy double counts the particles.

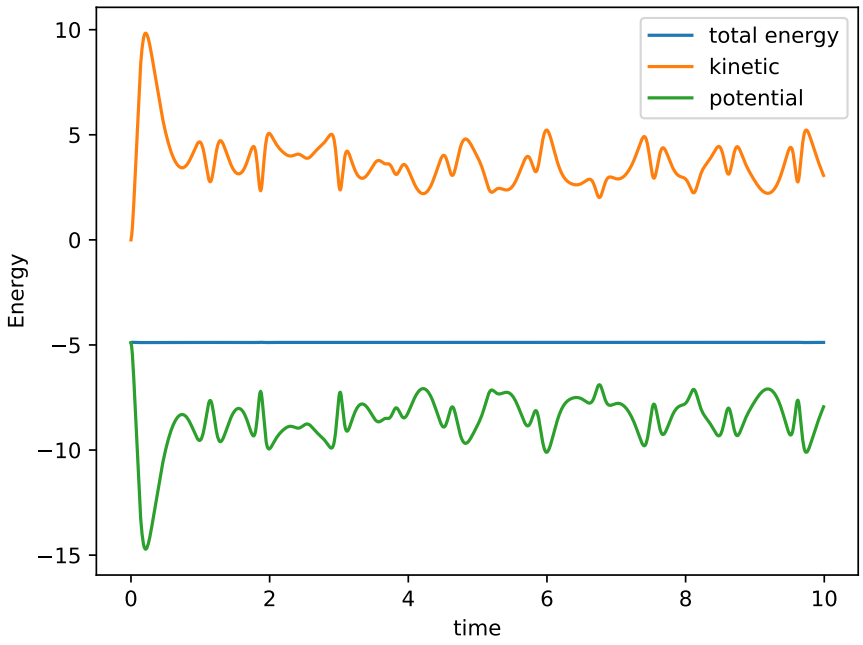

Figure 10: The energy of the system over time

c)

We implement the periodic boundary conditions and consider the interactions between particles outside of the computational domain with the particles inside the computational domain. In part a we can see since the middle four particles are really close together we get 4 groups of particles flying away from each other since we have not implemented any boundary conditions. In contrast in figure 11 we see that all 16 of the particles move around in proximity to their starting positions. We do not see any particles flying off outside of the edge of our domain because of repulsive force that comes from the 8N particles

located on the 8 surrounding tiles. Thus the attractive and repulsive forces in the potential energy are much more balanced and the particles stay near their initial position during the whole simulation. We do not observe any particles crossing the boundary of our domain.

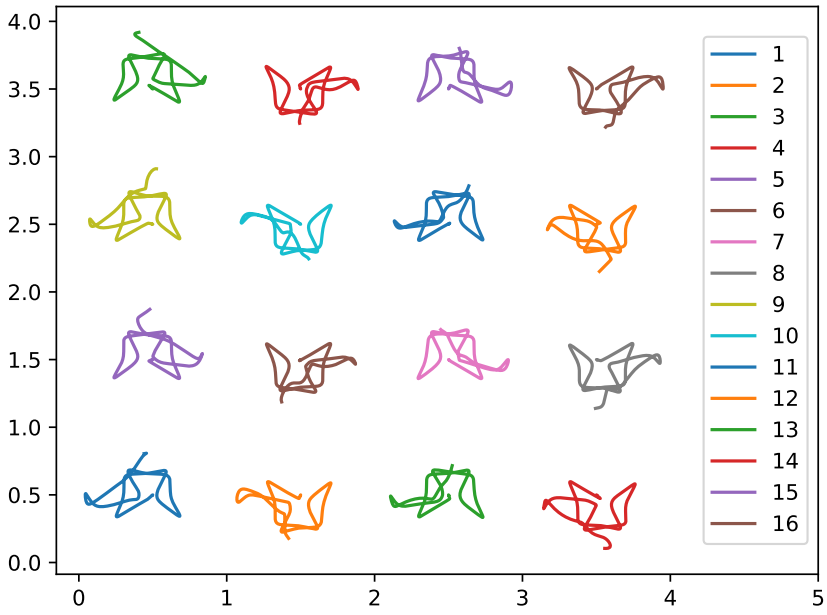

Figure 11: Trajectories of the particles with periodic boundary conditions

Appendix

Question 2c

X against Time

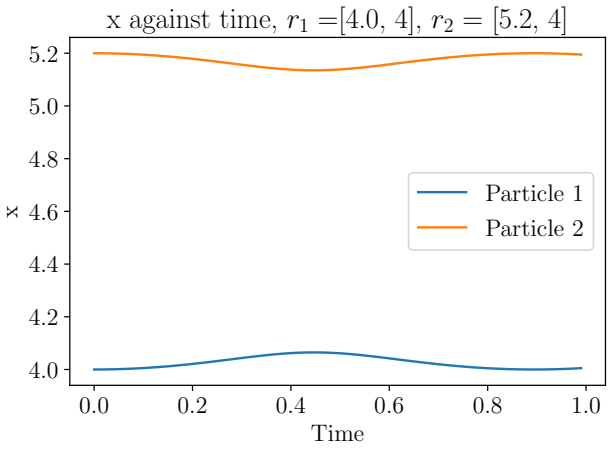

Figure 12: Question 2ci x position versus time.

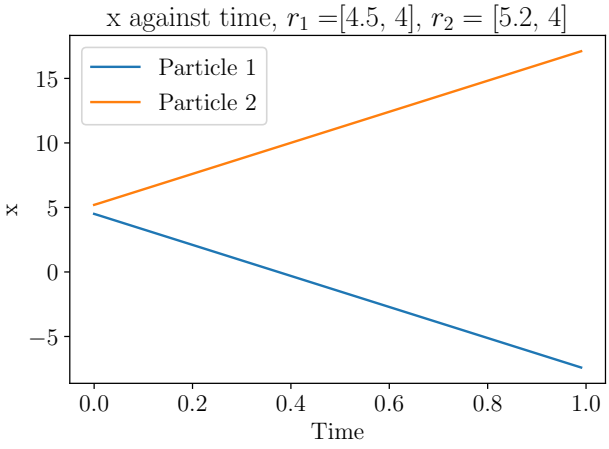

Figure 13: Question 2cii x position versus time.

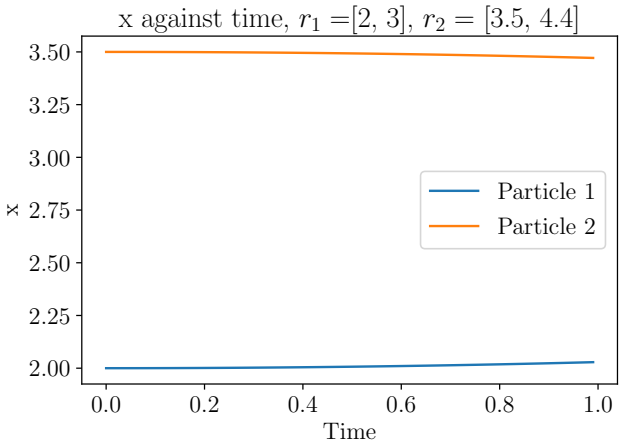

Figure 14: Question 2ciii x position versus time.

Y against time

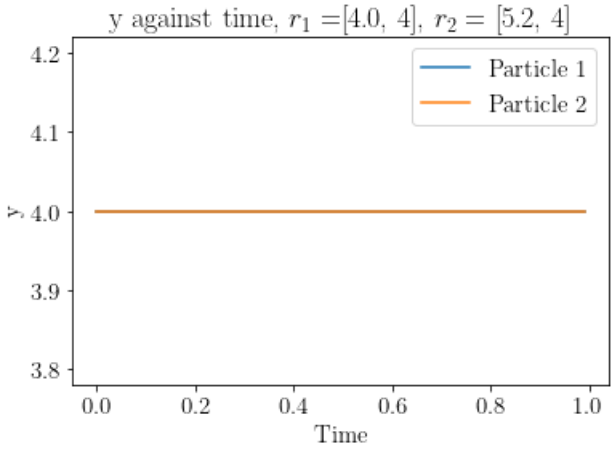

Figure 15: Question 2ci y position versus time.

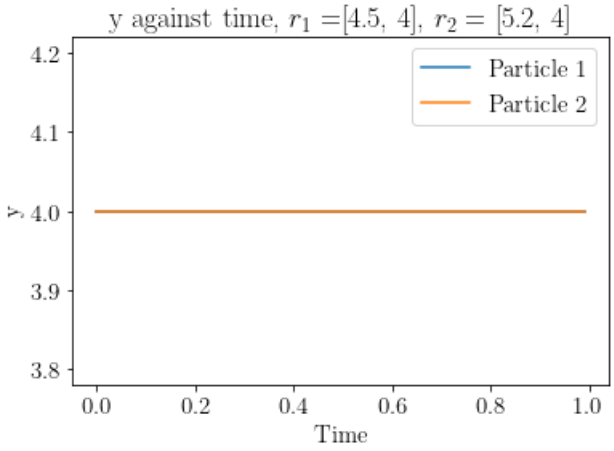

Figure 16: Question 2cii y position versus time.

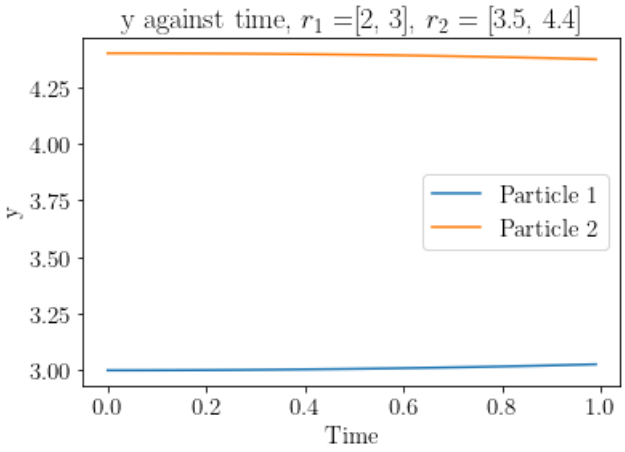

Figure 17: Question 2ciii y position versus time.

Energy against time

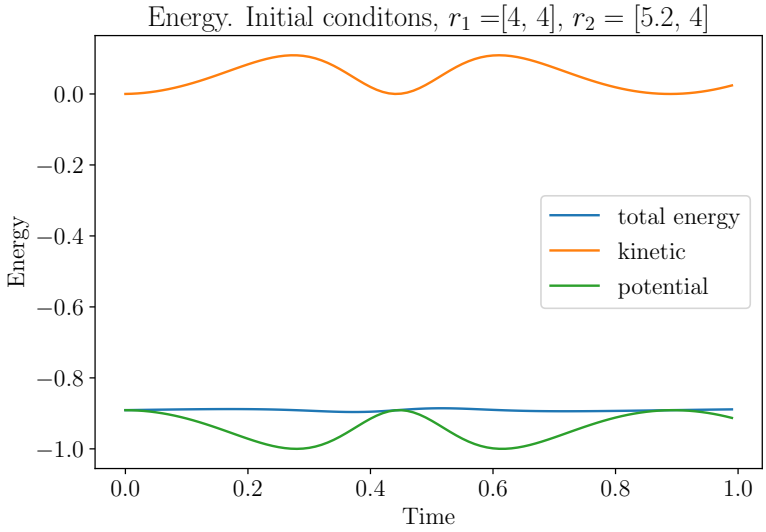

Figure 18: Question 2ci energy versus time.

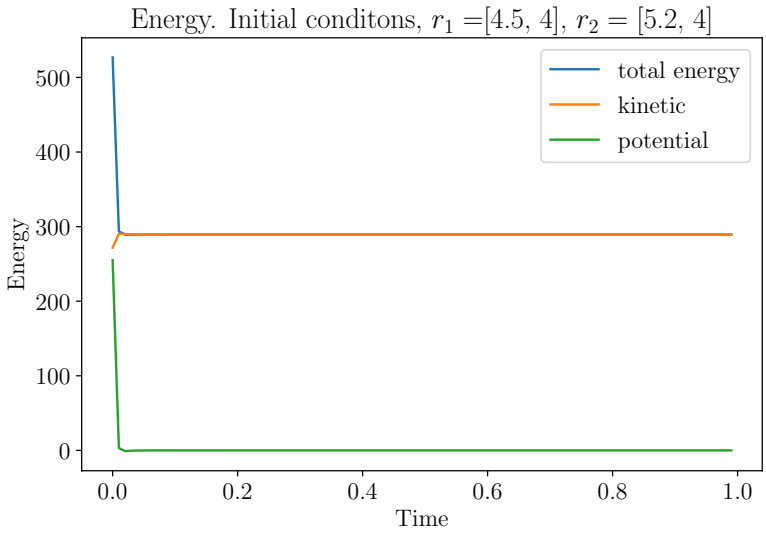

Figure 19: Question 2cii energy versus time.

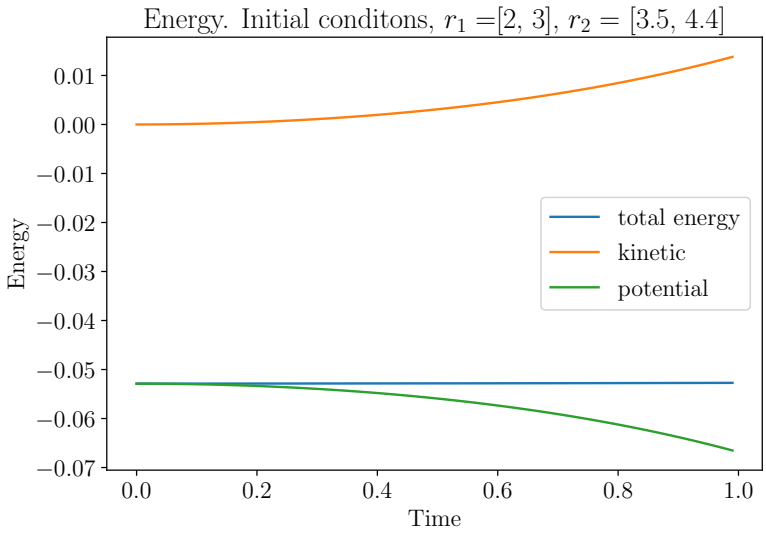

Figure 20: Question 2ciii energy versus time.