Lab Report 1

By Seunghyun Park 1003105855 & Juann Jeon 1005210166

Q1 done by Seunghyun Park & Juann Jeon
Q2 done by Seunghyun Park & Juann Jeon
Q3 done by Seunghyun Park
Lab Report by Juann Jeon

Q1 b)

```
import numpy as np
import matplotlib.pyplot as plt

#Constants
dt = Time step, in yr
t = Time interval of 1 year
M_s = Mass of the sun, in kg
G = Gravitational constant, in AU^3 M_s^-1 yr^-2

#np array to store values of x and y position and x and y velocity for Q1 c)
velocity_of_mercury_in_x_direction = Create np array with same length as t, with all indexes 0 for x velocity
velocity_of_mercury_in_x_direction[0] = Assign initial value of x velocity, in AU/yr
position_of_mercury_in_x_direction = Create np array with same length as t, with all indexes 0 for x position
posotion_of_mercury_in_x_direction[0] = Assign initial value of x position, in AU
velocity_of_mercury_in_y_direction  = Create np array with same length as t, with all indexes 0 for y velocity
velocity_of_mercury_in_y_direction [0] = Assign initial value of y velocity in AU/yr
position_of_mercury_in_y_direction  = Create np array with same length as t, with all indexes 0 for y position
position_of_mercury_in_y_direction [0] = Assign initial value y position, in AU
orbital_radius_of_mercury = Create np array with same length as t, with all indexes 0 for orbital radius of the Mercury
orbital_radius_of_mercury[0] = Assign initial value of orbital radius of the Mercury

#Calculate the velocity and position of the Mercury due to the Sun using for loop
for i in range(length_of_t):
```

$v_{x,i+1} = v_{x,i} - \frac{GM_s x_i}{r_i^3} \times dt$ #Calculate velocity of Mercury in x-direction

$x_{i+1} = x_i + v_{x,i+1} \times dt$ #Calculate position of Mercury in x-direction

$v_{y,i+1} = v_{y,i} - \frac{GM_s y_i}{r_i^3} \times dt$ #Calculate velocity of Mercury in y-direction

$y_{i+1} = y_i + v_{y,i+1} \times dt$ #Calculate position of Mercury in y-direction

$r_{i+1} = \sqrt{x_{i+1}^2 + y_{i+1}^2}$ #Calculate orbital radius of Mercury orbiting the Sun

#Plot the result
plt.plot(x velocity as a function of time)
plt.plot(y velocity as a function of time)
plt.plot(plot of the orbit)

Q1 c)

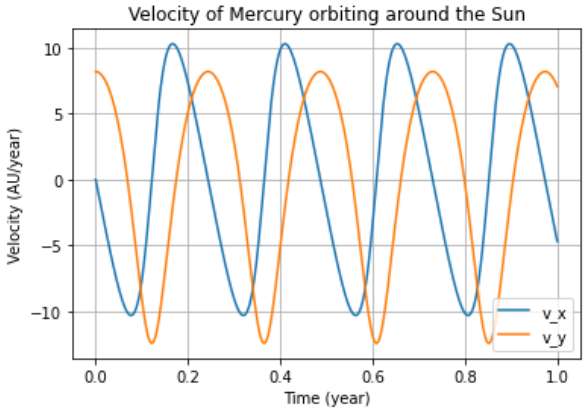

Figure 1

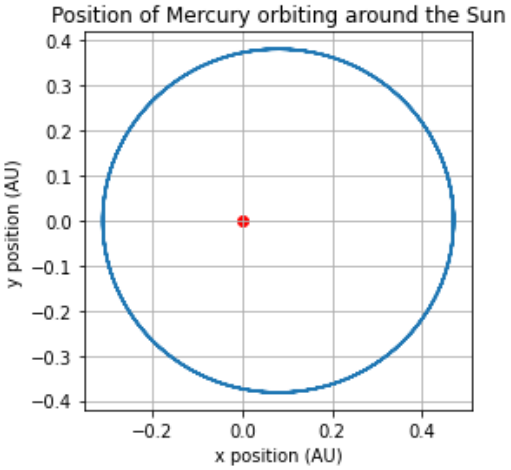

Figure 2

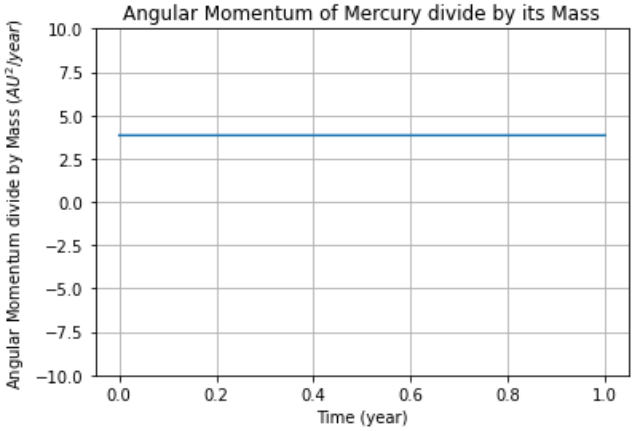

Figure 3

Figure 1 represents the velocity of Mercury orbiting around the Sun for 1 year, where the blue line represents velocity in x-direction and the orange line represents velocity in y-direction. Notice its velocities are shown in sinusoidal waves because planetary orbiting is in angular motion as we can see from Figure 2, which shows a perfect elliptical displacement. Figure 1 also shows angular momentum is conserved as it shows no damping in the graph. Figure 3, which shows the angular momentum of the Mercury, indeed shows that angular momentum is conserved.

Q1 d)

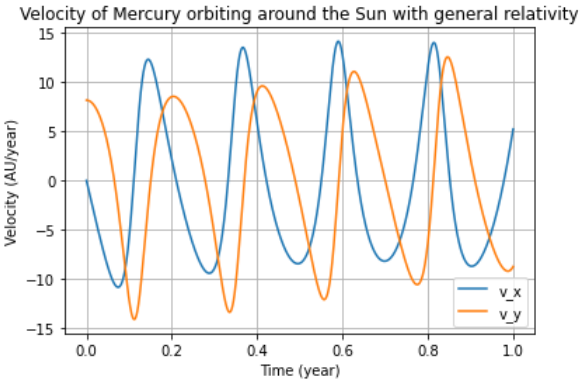

Figure 4

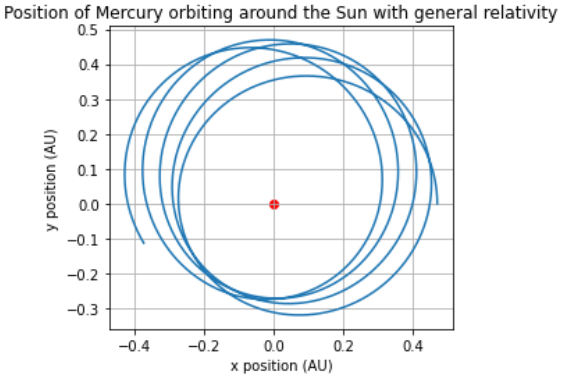

Figure 5

Figure 4 represents the velocity of Mercury orbiting around the Sun accounting general relativity, where the blue line represents the velocity in x-direction and the orange line represents velocity in y-direction. In Figure 4, constant $\alpha = 0.01AU^2$ is used to exaggerate the effect of general relativity between the Sun and the Mercury, as it can be observed from Figure 5. Notice that Figure 5 shows deviating elliptical displacement of Mercury in contrast to perfect elliptical displacement on Figure 2, due to the curvature of spacetime that general relativity accounts for but classical Newtonian gravitational force does not.

Q2 a)

```
import numpy as np
import matplotlib.pyplot as plt

#Constants
dt = Time step, in yr
t = Time interval of 10 years
M_s = Mass of the Sun, in kg
M_j = mass of the Jupiter, in kg
G = Gravitational constant, in AU^3 M_s^-1 yr^-2

#Position and velocity of the Earth
velocity_of_earth_in_x_direction = Create np array with same length as t, with all indexes 0 for x velocity
velocity_of_earth_in_x_direction[0] = Assign initial value of x velocity
position_of_earth_in_x_direction = Create np array with same length as t, with all indexes 0 for x position
posotion_of_earth_in_x_direction[0] = Assign initial value of x position
velocity_of_earth_in_y_direction  = Create np array with same length as t, with all indexes 0 for y velocity
velocity_of_earth_in_y_direction [0] = Assign initial value of y velocity
position_of_earth_in_y_direction  = Create np array with same length as t, with all indexes 0 for y position
position_of_earth_in_y_direction [0] = Assign initial value y position
orbital_radius_of_earth_orbiting_sun = Create np array with same length as t, with all indexes 0 for orbital radius of the Earth orbiting Sun
orbital_radius_of_earth_orbiting_sun[0] = Assign initial value of orbital radius of the Earth orbiting Sun
distance_between_earth_and_jupiter = Create np array with same length as t, with all indexes 0 for distance between the Earth and Jupiter
distance_between_earth_and_jupiter[0] = Assign initial value of distance between the Earth and Jupiter

#Position and velocity of the Jupiter
#Repeat exactly the same process that we did for position and velocity of the Earth, except we do not need to make one for orbital_radius_of_jupiter_and_earth as mass of the Earth is negligible to have meaningful gravitational force to pull Jupiter.

#Calculate the velocity and position of the Jupiter due to Sun using for loop
for i in range(length_of_t):
```

$$v_{xj,\,i+1} = v_{xj,\,i} - \frac{GM_s x_{j,\,i}}{r_i^3} \times dt \ \text{\#Calculate x velocity}$$

$$x_{j,\,i+1} = x_{j,\,i} + v_{xj,i+1} \times dt \ \text{\#Calculate x position}$$

$$v_{yj,\,i+1} = v_{yj,\,i} - \frac{GM_s y_{j,\,i}}{r_i^3} \times dt \ \text{\#Calculate y velocity}$$

$$y_{j,\,i+1} = y_{j,\,i} + v_{yj,\,i+1} \times dt \ \text{\#Calculate y position}$$

$$r_{j,\,i+1} = \sqrt{x_{j,\,i+1}^{\,2} + y_{j,\,i+1}^{\,2}} \ \text{\#Calculate orbital radius}$$

#Calculate the velocity and position of the Earth due to Jupiter and Sun using for loop
for i in range(length_of_t):

#Calculate x velocity of Earth

$$v_{xe,\,i+1} = v_{xe,\,i} - \frac{GM_s x_{e,\,i}}{r_{es,\,i}^{3}} \times dt - \frac{GM_J(x_{e,\,i} - x_{j,\,i})}{r_{ej,\,i}^{3}} \times dt$$

#Calculate x position of Earth

$$x_{e,\,i+1} = x_{e,\,i} + v_{xe,i+1} \times dt$$

#Calculate y velocity of Earth

$$v_{ye,\,i+1} = v_{ye,\,i} - \frac{GM_s y_{e,\,i}}{r_{es,\,i}^{3}} \times dt - \frac{GM_J(y_{e,\,i} - y_{j,\,i})}{r_{ej,\,i}^{3}} \times dt$$

#Calculate y position of the Earth

$$y_{e,\,i+1} = y_{e,\,i} + v_{ye,\,i+1} \times dt$$

#Calculate the orbital radius of Earth orbiting the Sun

$$r_{es,\,i+1} = \sqrt{x_{e,\,i+1}^{2} + y_{e,\,i+1}^{2}}$$

#Calculate the distance between Earth and Jupiter

$$r_{ej,\,i+1} = \sqrt{(x_{e,\,i+1} - x_{j,\,i+1})^{2} + (y_{e,\,i+1} - y_{j,\,i+1})^{2}}$$

#Plot the result
plt.plot(plot orbit of the Jupiter)
plt.plot(plot orbit of the Earth)

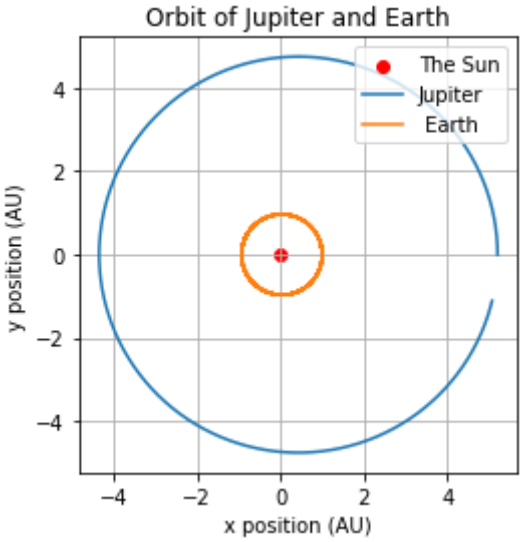

Figure 6

Figure 6 shows the Orbit of Jupiter and the Earth around the Sun for 10 years. It shows that Earth is slightly affected by the gravitational force by Jupiter, but the gravitational force by the Sun is strong enough to keep Earth in a stable orbit around the Sun.

Q2 b)

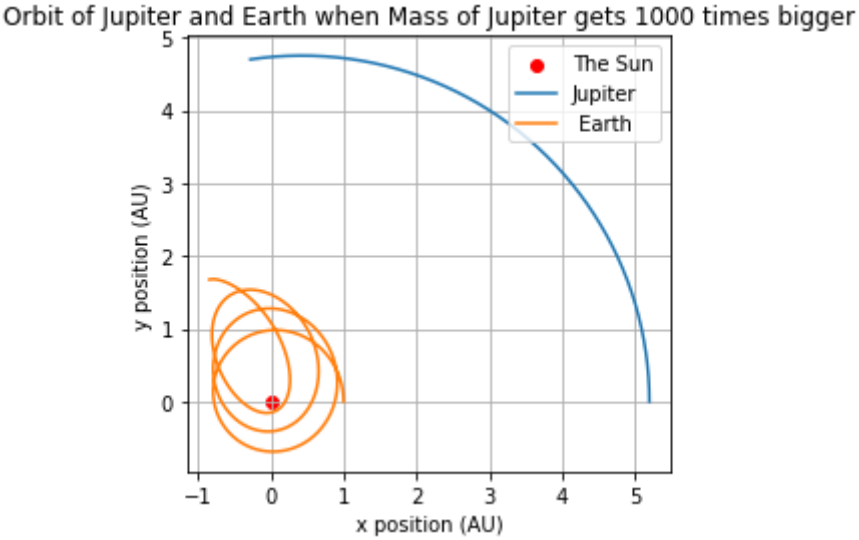

Figure 7

Figure 7 shows the orbit of the Earth in orange color for 3 years with the mass of Jupiter is 1000 times greater than its initial value in Q2 a), which is equal mass with the Sun. Initially it shows the usual elliptical displacement but quickly it starts to wobble around; it gets very close to the Sun and immediately gets further away from the Sun. It is predicted that Earth will eventually "shoot away" from the Sun, as the gravitational force by the Sun is not strong enough to keep Earth orbiting around the Sun due to the gravitational force by the unusually heavy Jupiter.

Q2 c)

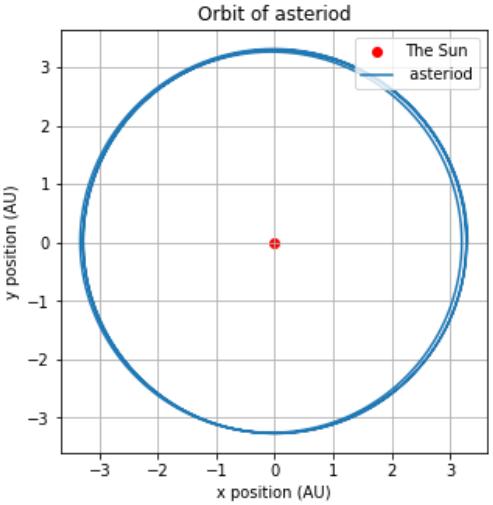

Figure 8

Figure 8 shows the slight perturbations in the asteroid's orbit after it orbited for 20 years around the Sun due to gravitational force by Jupiter.

Q3

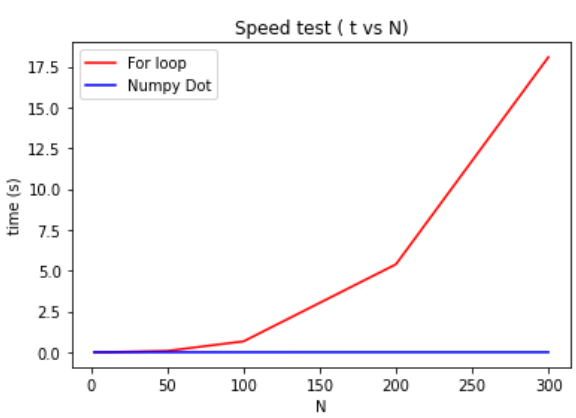

Figure 9

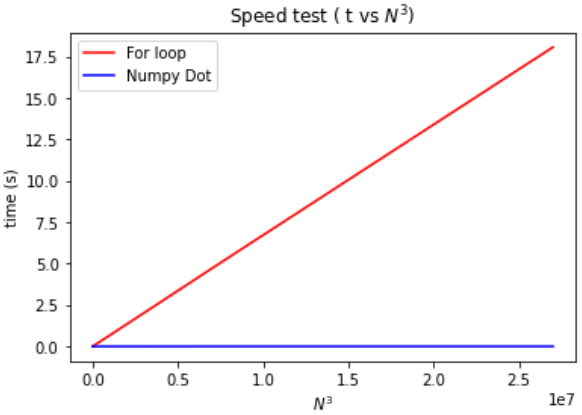

Figure 10

Figure 9 shows the time it took for Python to calculate $N \times N$ matrix multiplication using two different methods; red used for loop and blue used numpy.dot(). Figure 10 is same plot as the Figure 9, with only difference of x-axis is scaled by power of 3 to match $2N^3$ operation (as noted from page 137 of the textbook) of for loop method.

From both Figure 9 and Figure 10, it is observed that as size of $N \times N$ matrix gets larger, for loop method takes significantly longer time to calculate while numpy.dot() takes calculation immediately, staying relative to 0sec even with large $N \times N$ matrix.

In Python, Python list calls each element in indexes, meaning when performing $N \times N$ matrix multiplication using a for loop, it calls all of the elements in the list one by one[1]. So for example, if the list has a length of 100, during the for loop it calls all 100 elements (and this is precisely the reason why it takes $2N^3$ operation for for loop in Q3). On the other hand, numpy is mainly implemented in C and C++, meaning it does not undergo python implementation which is unoptimized in exchange for user-friendly experience[2]. Furthermore, numpy operation is highly optimized for mathematical and scientific operation unlike Python list does; such as numpy restricts arrays to be homogenous (i.e. only one type of element is allowed) while Python list can be heterogeneous (i.e. multiple types of elements are allowed)[3] . Because of these factors, numpy has great benefits over Python lists such as consuming much less memory and faster calculation time, as Figure 9 and 10 shows.

[1]https://stackoverflow.com/questions/10442365/why-is-matrix-multiplication-faster-with-numpy-than-with-ctypes-in-python
[2]https://www.w3schools.com/python/numpy/numpy_intro.asp#:~:text=NumPy%20is%20a%20Python%20library%20and%20is%20written%20partially%20in,in%20C%20or%20C%2B%2B.
[3]https://towardsdatascience.com/a-hitchhiker-guide-to-python-numpy-arrays-9358de570121#:~:text=NumPy%20arrays%20are%20made%20to,possible%20with%20heterogeneous%20data%20sets.