PHY407 Lab 1: Basic Python - Report

Contributor: Patrick Sandoval (Q1, Q3) and Chin Chong Leong (Q2, Q3)

patrick.sandoval@mail.utoronto.ca

kelvin.leong@mail.utoronto.ca

Due Date: September 16, 2022

1 Q1 - Modelling Planetary Orbits Through Euler-Cromer Method

1.1 Q1.b Pseudo-Code for Newtonian Orbit

For the pseudo-code for this question please refer to the .py file titled `Lab01_Q1.py`. We want to remark that the method outlined in the pseudo-code is mathematically equivalent to eq.4a or eq.4b in the lab but with a shift in the index for implementation purposes.

1.2 Q1.c Modelling Newtonian Orbit

Following the pseudo-code from subsection 1.1 we run a for loop for `len(t)-1` iterations while using the initial conditions posed by the question. Additionally, we converted the gravitational constant to units of $\mathrm{AU}^3/\mathrm{yr}^2/\mathrm{kg}$ such that the units would cancel out with the units of the initial conditions the sun's mass. We continue to initialize arrays of zeros which we'll later populate through the for loop. Within the for loop we need to compute r from the previous position instances `x[i-1]` and `y[i-1]`, (note) `i` starts at 1. In a more precise manner the code is set up in the following way

$$\begin{aligned} a_{x,i} &= \frac{GM_s}{r^3} x_{i-1} & a_{y,i} &= \frac{GM_s}{r^3} y_{i-1} \\ v_{x,i} &= v_{x,i-1} + dt\, a_{x,i} & v_{y,i} &= v_{y,i-1} + dt\, a_{y,i} \\ x_i &= x_{i-1} + dt\, v_{x,i} & y_i &= y_{i-1} + dt\, v_{y,i} \end{aligned} \tag{1}$$

where $a_{k,i}$ is the k^{th} component of the acceleration vector at the i^{th} index, and similarly for the velocity and position vectors.

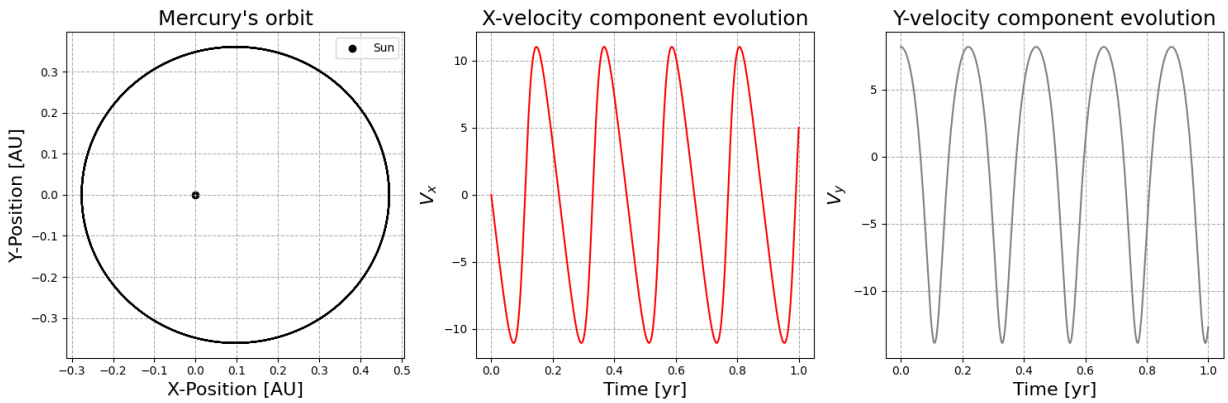

Figure 1: (Left-most panel) Mercury's orbit around the sun for 1 year, adopting the system of equations in 1. (Middle & right-most panels) Cartesian velocity components of Mercury's orbit

From Figure 1 we can see a closed elliptical orbit and the expected sinusoidal behaviour of the Cartesian velocity components. Because mercury is under the influence of a central potential then $\vec{r} \perp \vec{v}$ which leads to the following result

$$|\vec{L}|^2 = M_m^2 |\vec{r}|^2 |\vec{v}|^2 \tag{2}$$

where M_m^2 is the mass of Mercury and is a constant. Therefore, we can compute the magnitude of the angular momentum throughout the entirety of the simulation in order to verify if it is conserved.

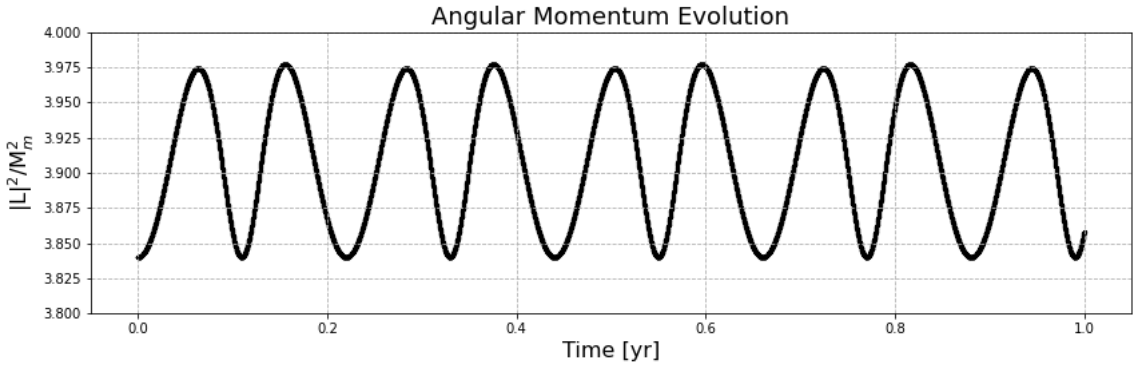

Figure 2: Magnitude of angular momentum vector (squared divided by M_m^2) throughout the simulation. Oscillatory behaviour is attributed to the limitation of the Euler-Cromer integration method. Altogether results point towards the conservation of total angular momentum.

As we can concluded from Figure 2, the angular momentum is conserved throughout the orbit; though we noticed there are small amplitude of oscillations in the angular momentum, possibly due to the limitation of the Euler-Cromer method (i.e. numerical simulation are not perfect).

1.3 Q1.d Modelling Mercury's Orbit with General Relativity

Using the same initial conditions posed in Q1.c, subsection1.2, we can model Mercury's orbit through the gravitational force predicted by general relativity.

$$\vec{F_g} = -\frac{GM_s M_p}{r^3}\left(1 + \frac{\alpha}{r^2}\right)(x\hat{x} + y\hat{y}) \tag{3}$$

where M_s is the mass of the sun, M_p is the mass of the planet, r is the sun-planet euclidean distance and α is a constant depending on the scenario. From eq.3 we can create the the same system of equations as shown in 1 in order to compute the position and velocity components of Mercury. The main difference between the implementation here and the one in 1 is the term $(1 + \alpha/r^2)$ in the acceleration computation. Aside from that the codes are identical.

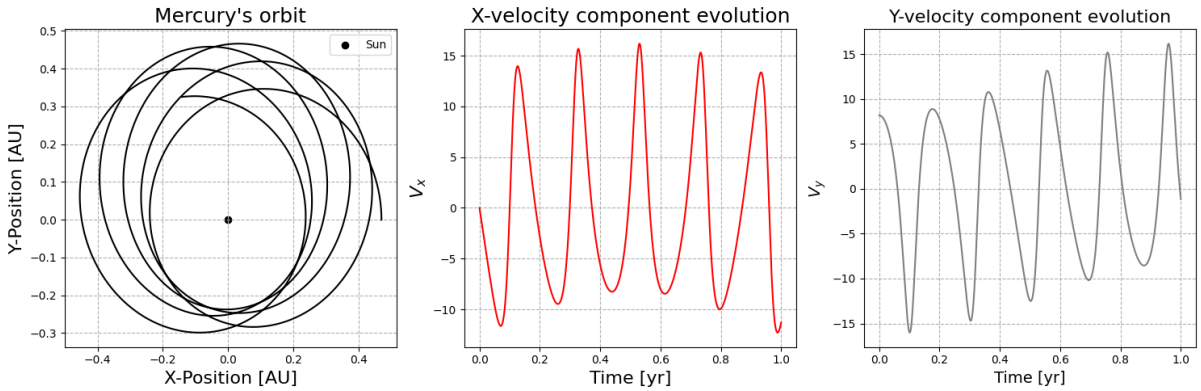

Figure 3: (Left-most panel) Mercury's orbit for 1 year, adopting the gravitational force predicted by general relativity as shown in eq.3. (Middle & right-most panel) Cartesian velocity components of Mercury correspondingly to eq.3

2 Q2 - Modelling Three Body Problem

2.1 Q2.a - Modelling Jupiter's and Earth's Orbit

For the pseudo-code for this questions please refer to the `Lab01_Q2.py`. Following the pseudo-code, we initialize arrays of zeros of positions and velocities for Jupiter and Earth respectively and set the initial conditions posed by the question. By running a for-loop for `len(t)-1` iterations, the arrays for Jupiter are populated using the Euler-Cromer method with the following set of equations:

$$\begin{aligned} a_{J,x,i} &= -\frac{GM_s}{r^3}x_{J,i-1} & a_{J,y,i} &= -\frac{GM_s}{r^3}y_{J,i-1} \\ v_{J,x,i} &= v_{J,x,i-1} + dt\, a_{J,x,i} & v_{J,y,i} &= v_{J,y,i-1} + dt\, a_{J,y,i} \\ x_{J,i} &= x_{J,i-1} + dt\, v_{J,x,i} & y_{J,i} &= y_{J,i-1} + dt\, v_{J,y,i} \end{aligned} \tag{4}$$

where r here is the distance between the Sun and Jupiter, $a_{J,k,i}$ is defined as the k^{th} component of Jupiter's acceleration vector at the i^{th} index, and similarly defined for the components of the velocity and position vectors. Subsequently, another for-loop was run for `len(t)-1` iterations, and the arrays for Earth are populated using the Euler-Cromer method the following set of equations:

$$\begin{aligned} a_{E,x,i} &= -\frac{GM_s}{r_{ES}^3}x_{ES,i-1} - \frac{GM_J}{r_{EJ}^3}(x_{E,i-1} - x_{J,i-1}) \\ a_{E,y,i} &= -\frac{GM_s}{r_{ES}^3}y_{ES,i-1} - \frac{GM_J}{r_{EJ}^3}(y_{E,i-1} - y_{J,i-1}) \\ v_{E,x,i} &= v_{E,x,i-1} + dt\, a_{E,x,i} \\ v_{E,y,i} &= v_{E,y,i-1} + dt\, a_{E,y,i} \\ x_{E,i} &= x_{E,i-1} + dt\, v_{E,x,i} \\ y_{E,i} &= y_{E,i-1} + dt\, v_{E,y,i} \end{aligned} \tag{5}$$

where M_J is the mass of Jupiter, r_{ES} is the distance between the Sun and Earth, and r_{EJ} is the distance between the Earth and Jupiter; $a_{E,k,i}$ is defined as the k^{th} component of Earth's acceleration vector at the i^{th} index, and similarly defined for the components of

the velocity and position vectors. The Earth's and Jupiter's positions over 10 years were plotted in Figure 4.

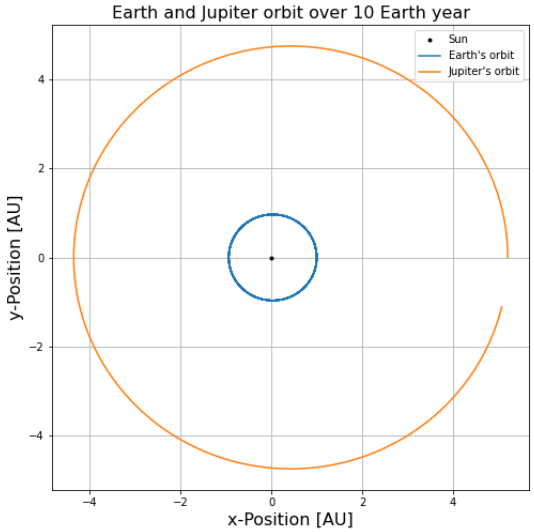

Figure 4: The orbits of Earth and Jupiter over 10 years. The blue line represents the positions of Earth and the orange line represents the positions of Jupiter. Note that since Jupiter's orbital period is 11.86 years, Jupiter has yet to finish its orbit in the figure

As we can see from the figure, Earth's orbit stays very stable, which implies that Jupiter does not have a big effect on Earth as expected.

2.2 Q2.b - Modelling Jupiter's and Earth's orbit with 1000 times more mass on Jupiter

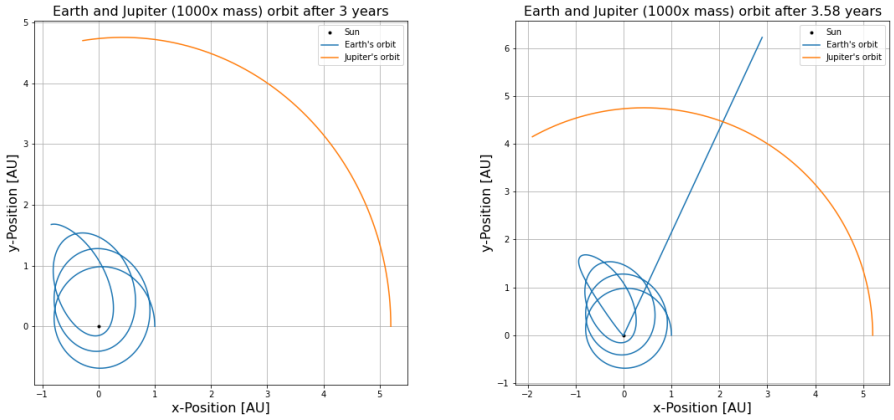

Figure 5: The orbits of Earth and Jupiter if Jupiter is 1000 times more massive. (Left Panel) Planets' orbit after 3 years. (Right Panel) Planets' orbit after 3.58 years, where Earth is slingshot out of the system.

For this part of the exercise, we set Jupiter's mass to be 1000 times more massive (i.e. $M_J = 1M_\odot$). Then we followed the same procedure as in section 2.1 to generate Jupiter's and Earth's orbit.

The left panel in Figure 5 shows the orbit of Earth and Jupiter after 3 years, and evidently it has significant effects on Earth's orbit when Jupiter was set to 1000 times more massive. For long-term behaviour of Earth, we could also see from the same figure that Earth's aphelion point becomes closer to the Sun over time due to its unstable orbit; eventually after approximately 3.58 years, the Earth is sling-shot out of the system and its orbit becomes unbounded, as seen in the right panel in Figure 5.

2.3 Q2.c - Modelling an Asteroid's and Jupiter's orbit

For this part of the exercise, we set Jupiter mass back to be normal and used the initial conditions for the asteroid posed by the question. Their orbits over 20 years are plotted in Figure 6. There are indeed some noticeable perturbation in the asteroid's orbit.

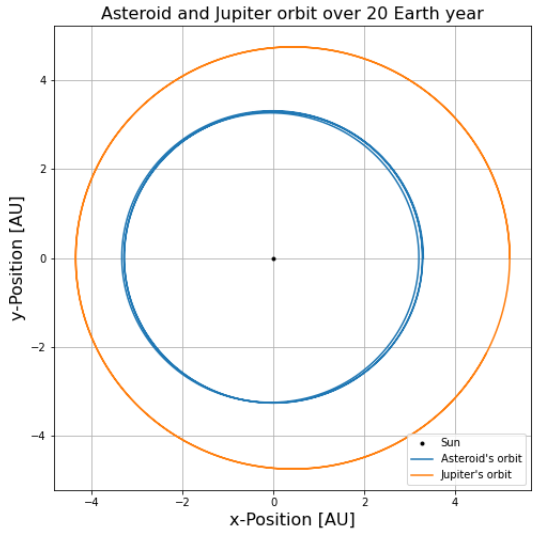

Figure 6: The orbits of Jupiter and an asteroid starting from 3.3 AU from the Sun over 20 years. The blue line represents the positions of the asteroid and the orange line represents the positions of Jupiter.

3 Q3 - Timing Matrix Multiplication

Please refer to `Lab01_Q3.py` for the python code regarding this question. Multiplying two matrices of size $O(\mathrm{N})$ on a side take $O(\mathrm{N}^3)$ operations. In this section we will motivate this statement by performing matrix multiplication of $N \times N$ matrices while at the same time we increase N to a couple hundreds. We will time each iteration of the multiplication and then plot the time it took for the operation to complete vs. the size of the matrices. In order to verify that $t \propto N^3$ we will then plot the time vs N^3 and we will expect to a see a linear relationship which will confirm our first statement in this section.

Additionally, we will compare the timing of the matrix multiplication algorithm to

the time of the `numpy.dot` function for the same task. We will plot the `numpy` results in $t \propto N$ and in $t \propto N^3$. However, in this case we don't expect to see any major trends because of how fast the `numpy` can perform these matrix operations.

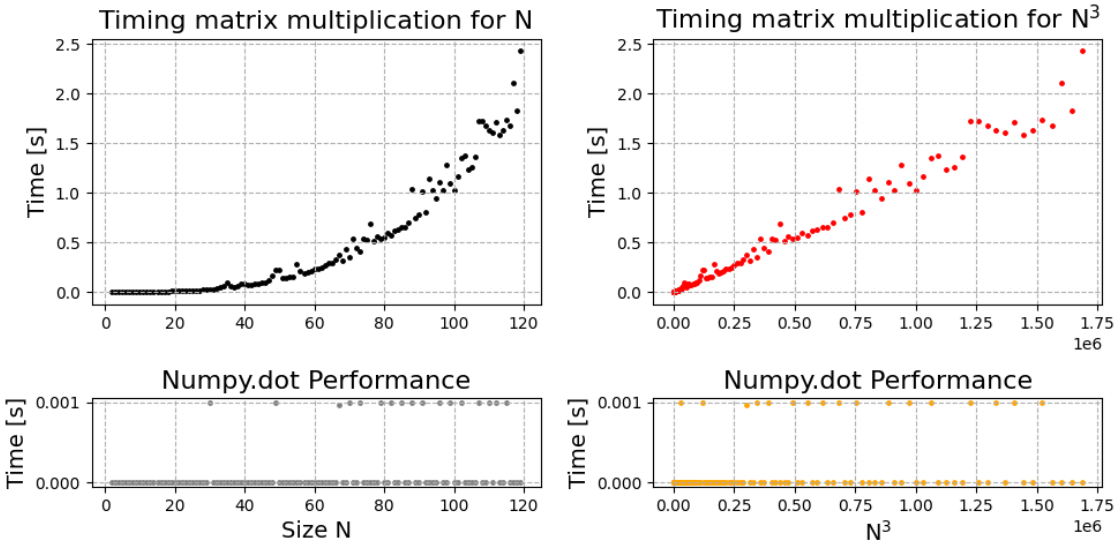

Figure 7: Matrix multiplication computational time for two N square matrices. (Left panel) verifies the power relationship between time and matrix size ($t \propto N^3$)

We can see in the top left panel of Figure 7 the power relationship between the time and the size of the matrix and on the top right panel we see the linear relationship between t and N^3. Lastly with the bottom two plots we see that the `numpy.dot` function keeps the computation time constants with very small fluctuations in time.