PHY407 Lab 1: Python Basics

Work Distribution:
Emma Jarvis: Q1ab, Q3
Lisa Nasu-Yu: Q1cd, Q2

September 17, 2021

1 Modelling a Planetary Orbit

1.a

We started by rearranging the equations governing the motion of the planet in Cartesian coordinates into equations suitable for numerical integration with the Euler-Cromer method. These equations are then used in step 7 of the pseudocode in Q1b.

1.b

We then wrote pseudocode to compute the position and velocity of a planet as a function of time (refer to the pseudocode below). We then turned this pseudocode into actual Python code to write this program (Q1c).

```
"""
Pseudocode that computes the position (x and y) and x and y components of the
velocity (vx and vy) of a planet as a function of time under
Newtonian gravity force given the initial conditions for x, y, vx and vy and
using the Euler-Cromer method.
Author: Emma Jarvis, Sept. 2021
"""

# Pseudocode:
# Import necessary packages (numpy and matplotlib.pyplot)
# 1. Define the gravitational constant G, and the mass of the sun, M_s
# 2. Define initial x0, y0, vx0, vy0
# 3. Define end time and dt the time step
# 4. Initialize the time array from initial to final time with a time step of dt.
# 5. Set first value of x, y, vx and vy with their respective initial values.
# 6. Initialize x, y, vx, and vy arrays with the same number of elements.
# 7. For a number of iterations corresponding to one less than the length of the time array:
    # Define the distance between the planet and the sun,
    # r = sqrt(x[i]**2 + y[i]**2)
    # Increment the vx array with vx[i+1] = vx[i] - (G*M_s/r**3) * x[i] * dt
    # Increment the vy array with vy[i+1] = vx[i] - (G*M_s/r**3) * y[i] * dt
    # Increment the x array with Euler-Cromer: x[i+1] = x[i] + vx[i+1] * dt
```

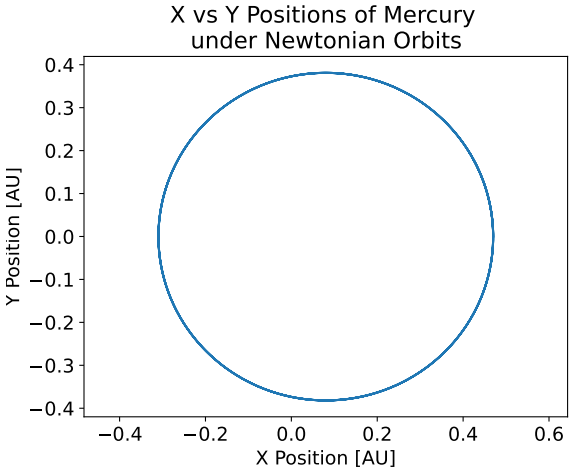
```
    # Increment the y array with Euler-Cromer: y[i+1] = y[i] + vy[i+1] * dt
# 8. Plot vx vs. t
# 9. Plot vy vs. t
# 10. Plot x vs. y
```

1.c

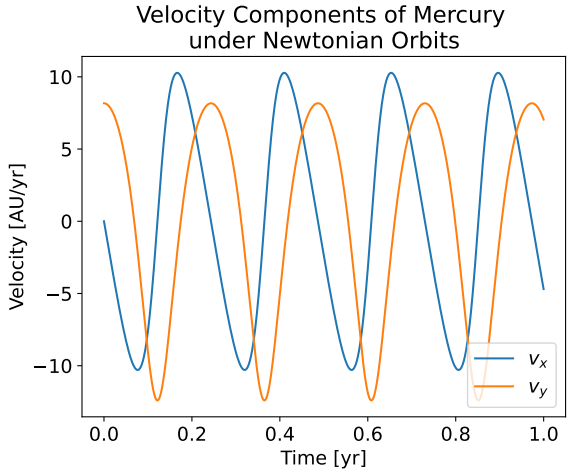
Using our pseudocode from Q1b as a guideline, we computed the position and velocity of Mercury in the x and y directions as a function of time. The initial conditions of Mercury were set as given in the lab assignment. Angular momentum was also calculated as a function of time to determine whether it is conserved throughout the orbit.

	x	y
Position [AU]	0.47	0
Velocity [AU/yr]	0	8.17

Table 1: Initial conditions of Mercury



(a) x vs y positions, plotted for year



(b) x and y components of velocity

Figure 1: 1a shows the x and y positions of Mercury in a Newtonian orbit, relative to the Sun. 1b shows the x and y components of Mercury's orbital velocity.

Although 1b appears to be a single orbit, it actually contains several closely overlapped orbits over the course of 1 Earth year. Mercury's shorter period is more apparent in 1a, where both x and y velocities oscillate at an amplitude of 10.3 AU/yr. The phases of the 2 components are shifted such that one approaches zero as the absolute value of the other approaches its maximum. These are all characteristic qualities that we would expect from an orbit.

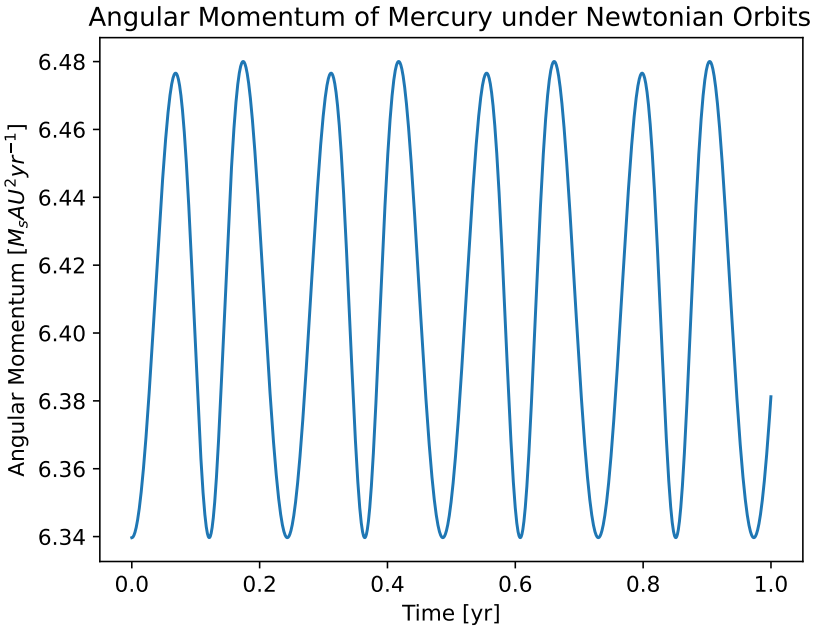
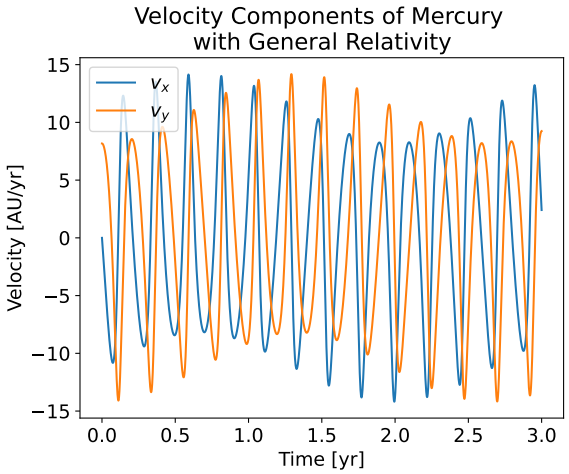


Figure 2: The angular momentum, $L = mvr$, of Mercury as a function of time. This was calculated using our positions and velocities computed from the simulation.

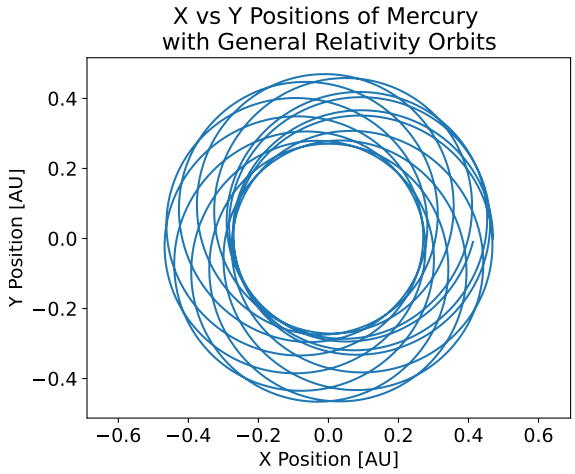
It is clear to see from 2 that angular momentum is not conserved throughout Mercury's orbit, as it oscillates between 6.34 and 6.48 $AUM_{sun}s^{-1}$. Mercury orbits the Sun at a slight inclination. As such, there can be a non-negligible angle between the gravitational force and position vectors, resulting in torque. With torque acting on the planet, angular momentum is not conserved.

1.d

We altered our code to use the general relativity form of gravitational force in our integration and presented Mercury's orbital precession. For easier observation, we took α=0.01AU^2, instead of the actual value of 1.1×10^{-8}AU^2, to exaggerate the results.



(a) x and y components of velocity



(b) x vs y position, plotted for a duration of 3 years

Figure 3: 1a shows the x and y positions of Mercury in a orbit determined by general relativity. 1b shows the x and y components of Mercury's orbital velocity.

With the general relativity form of gravity several orbits within the 3 year span are clearly visible, with a moving perihelion in 1b. The relativistic gravity also had an effect on orbital velocity, as shown in 1a. In addition to the oscillation found in the Newtonian evaluation of Mercury, there is a general oscillation in the absolute maximum velocities of each orbit, which agrees with the oscillating perihelion.

2 The Three Body Problem

2.a

We simulate the orbits of Earth and Jupiter in the three body problem involving the 2 planets and the Sun. Due to its high mass, we assume the Sun is not affected by Earth or Jupiter's gravitational force, and is at the centre of the Solar System. For the same reasoning, we also assume that Jupiter is not affected by the gravitational force of the Earth. Initial conditions were set as given in the lab assignment.

```
"""
    Pseudocode for computing the x and y position (x, y) and velocity (vx, vy) of
    Earth and Jupiter in a three body system under Newtonian gravitational force,
    given their initial conditions for position and velocity in the x-plane and
    using the Euler-Cromer method.
    Author: Lisa Nasu-Yu, Sept. 2021
"""

# Pseudocode:
# Import necessary packages (numpy and matplotlib.pyplot)
# 1. Define the gravitational constant, G and the mass of the sun, M_s

# 2. Apply Q1 b pseudocode to calculate x & y position and velocity for Jupiter,
```

```
#   and corresponding initial conditions xj0, yj0, vxj0, vyj0

# 3. Define time step, dt=0.0001 year, and duration of integration, t_f=10 years
# 4. Define Earth's initial x & y position and velocity: xe0, ye0, vxe0, vye0
# 5. Initialize time array from initial time to final time with a time step of dt.
# 6. Initialize x & y position and velocity (x, y, vx, and vy) arrays with the same
#   number of elements as the time array.
# 7. Set first element of x, y, vx, and vy with their respective initial values.
# 8. For number of iterations corresponding to 1 less than the length of the time array:
#         Calculate distance between Earth and Jupiter,
#         rj = sqrt((xj[i] - x[i])**2 + (yj[i] - y[i])**2)
#         Define gravitational force fj on Earth due to Jupiter (G*M_s/rj**3)
#         Calculate distance between Earth and Sun, rs = sqrt(x[i]**2 + y[i]**2)
#         Define gravitational force fj on Earth due to Jupiter (G*M_s/rs**3)
#         Total gravitational force felt by Earth is the sum of that from
#         Jupiter and the Sun:
#           Increment the vx array with
#           vx[i+1] = vx[i] - (fs * x[i] - fj * (xj[i] - x[i])) * dt
#           Increment the vy array
#           with vy[i+1] = vy[i] - (fs * y[i] - fj * (yj[i] - y[i])) * dt
#           Increment the x array with Euler-Cromer: x[i+1] = x[i] + vx[i+1] * dt
#           Increment the y array with Euler-Cromer: y[i+1] = y[i] + vy[y+1] * dt
# 9. Plot x vs. y
```

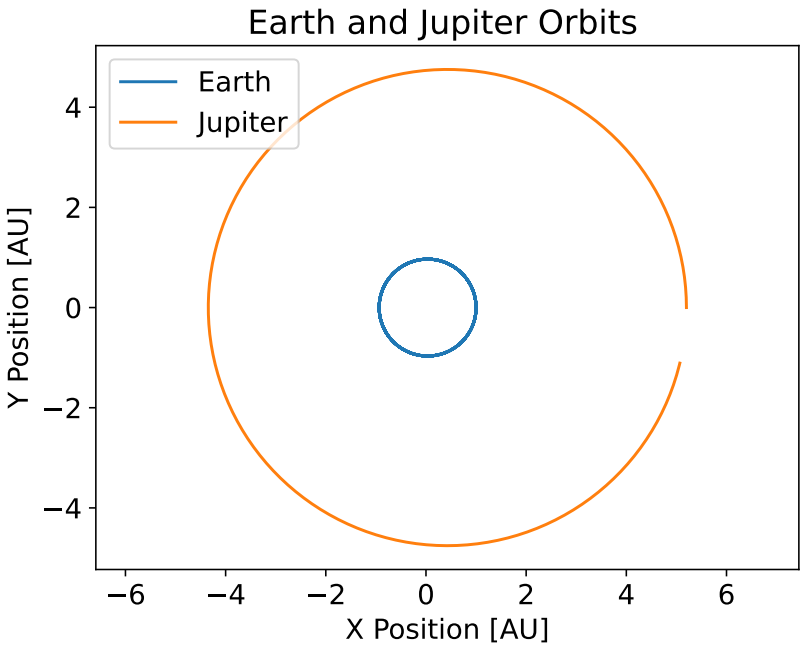
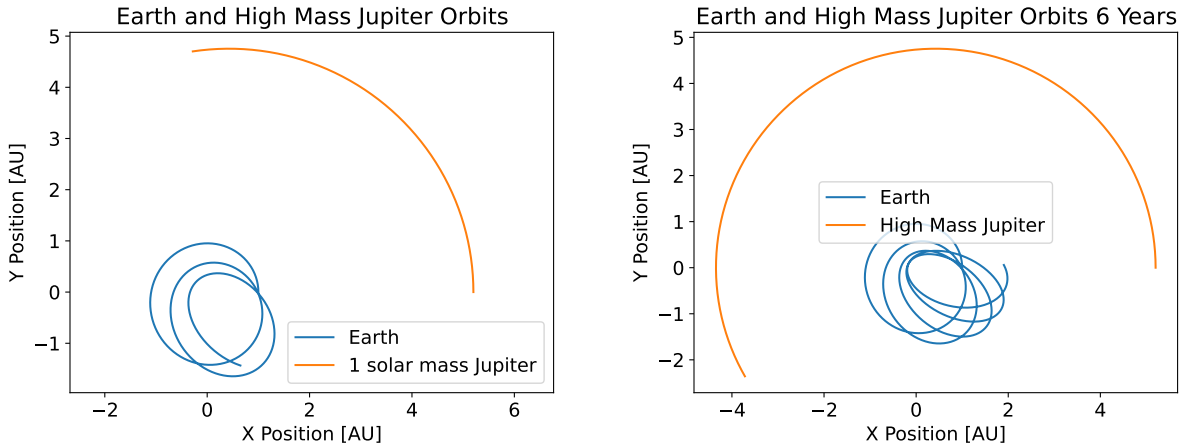


Figure 4: Spatial coordinates in the x-y plane from a simulation of orbital motion for Earth and Jupiter over 10 years. The Earth's orbit has no visible perturbations, indicating that Jupiter's gravitational forces have little effect on the Earth's orbit.

2.b

Here, the simulation was run twice more, but with Jupiter's mass increased by 1000 times, to have the same mass as the Sun, and decreasing the duration of the simulation to 3 and 6 years, respectively. All other initial conditions remained the same.



(a) Orbits plotted over 3 years

(b) Orbits plotted over 6 years

Figure 5: Spatial coordinates in the x-y plane from simulated orbits of Earth and an object with the same initial conditions as Jupiter, but a mass of 1 solar mass. 5a and 5b were run for 3 and 6 years, respectively.

The higher mass Jupiter caused large perturbations in the Earth's orbit as shown in Figure (a). In 5b, the 6 year simulation shows a rotation in the center of the Earth's orbit. This suggests that our assumption that the Sun is at rest is incorrect, and it should in fact be affected by the gravitational force of the 1 M_s Jupiter.

2.c

We ran the same simulation again, this time replacing Earth with an asteroid at an orbital distance of 3.3 AU. Jupiter's mass was returned to its actual value of $10^{-3}M_s$, and the simulation was run for 20 years.

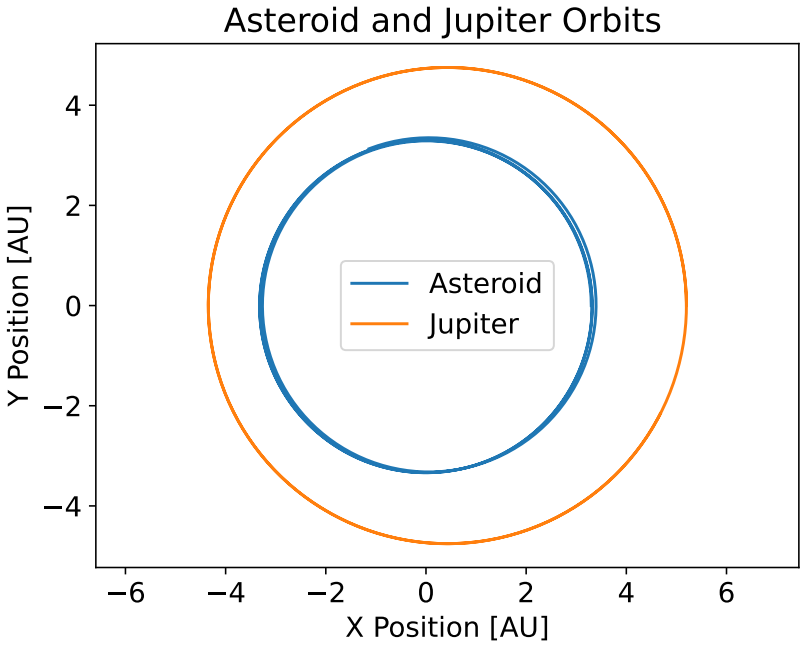


Figure 6: Spatial coordinates in the x-y plane from simulated orbits of an asteroid with an orbital radius of 3.3 AU and Jupiter, in a 3 body system. The asteroid's orbit is greater than that of Earth and slight perturbations are visible, as we expected.

3 Timing Histogram

3.a

We started by writing pseudocode to histogram N random samples into M linearly spaced bins. We then used this pseudocode to write real Python code that specifies the number of bins ($M = 1000$), number of samples (column 1 of Table 2) and range of the bins (-5 to 5).

```
"""
Pseudocode that histograms N random samples generated with numpy.random.randn into
M linearly spaced bins within a specified range.
Author: Emma Jarvis, Sept. 2021
"""
# Pseudocode
# 1. Import numpy and matplotlib.pyplot
# 2. Set the minimum bin value
# 3. Set the maximum bin value
# 4. Generate an array of M linearly spaced bins between a range using numpy.linspace
# 5. Generate an array N samples using numpy.random.randn
# 6. Sort the array of N samples into the M bins.
```

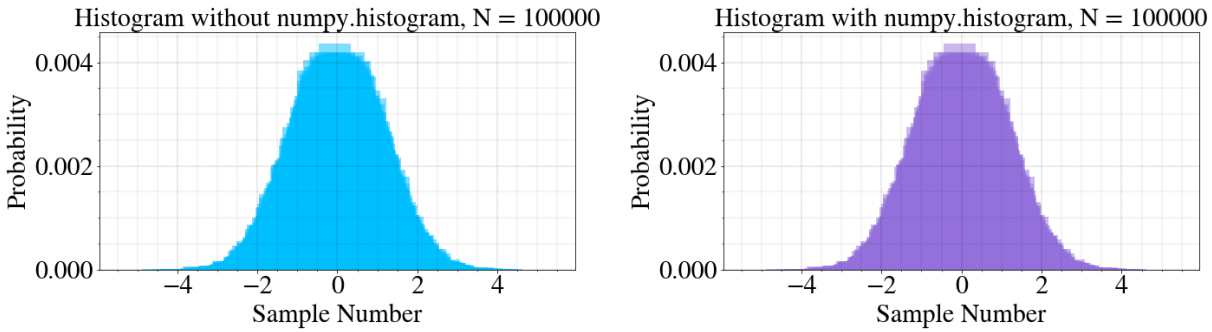
3.b

We then used `time` to compute the time that it takes to generate this histogram. Column 2 of Table 2 contains the times to generate a histogram without using `numpy.histogram` for a range

from 10 to 10^6 samples. The histogram was made by dividing the samples into 1000 linearly spaced bins ranging from -5 to 5. The samples were Gaussian-distributed. Figure 7a shows one of the histograms that was generated with 10^5 samples.

After generating the histogram from scratch, the same histogram was generated using `np.histogram`. Figures 7a and 7b demonstrate that both methods yield the same result by plotting histograms with the same 1D array of random samples. Table 2 contains the timing values for the histogram function to run with and without using `numpy`.

Note to TA: these timing values vary slightly every time new random samples are generated using `numpy.random.randn` so the timing values cannot be exactly replicated.



(a) Without `numpy.histogram`

(b) With `numpy.histogram`

Figure 7: Figure (a) depicts the histogram generated from scratch without the use of `numpy.histogram` for data with 10^5 samples. Figure (b) depicts the histogram generated from scratch using `numpy.histogram` for data with 10^5 samples. Both methods yield identical results.

Number of Samples	Time without `np.histogram` (s)	Time with `np.histogram` (s)
10^1	0.0057871341705322266	0.0002460479736328125
10^2	0.006127119064331055	0.00032901763916015625
10^3	0.009304046630859375	0.0003581047058105469
10^4	0.019370079040527344	0.0009250640869140625
10^5	0.11069488525390625	0.011124849319458008
10^6	1.8761353492736816	0.08561825752258301

Table 2: Time taken to generate a histogram for a variety of numbers of samples without and with using `numpy.histogram`.

Figure 8 depicts the timing values to create histograms from scratch and using `numpy.histogram` as a function of the number of samples. These values correspond to the values in Table 2. The y-axis of the plot is plotted using a log-scale. This figure shows that for each number of samples, the histogram is consistently generated more quickly using `numpy.histogram` than the histogram generating function from scratch. This discrepancy is because generating a histogram from scratch does not optimize the speed like `numpy`. The for loop takes longer than the method that `numpy` uses.

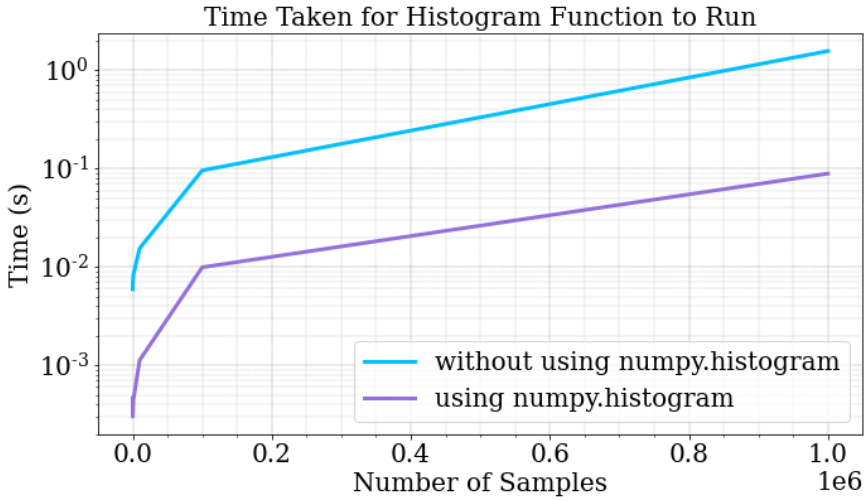


Figure 8: The time taken to generate a histogram using 2 methods; making the histogram from scratch (blue line) and using `numpy.histogram` (purple line). The time (y-axis) is plotted with a log scale. With both methods, as the number of samples increases, the time increases. For all numbers of samples, using `numpy.histogram` is faster than writing the histogram function from scratch.