n-Body Newtonian Gravitational Simulation using Numerical Integration Methods

Katie Archer
(Dated: 11/12/20)

We design a flexible python program for performing gravitational simulations on point masses using Newtonian physics, using two different numerical integration methods which we compare, demonstrating by comparing simulation results to ephemerides that the Euler-Cromer method is usually more accurate than the Euler method except for slow moving, far out objects. We simulate projectile motion, the motion of the planets in the solar system, and the motion of the planets and large moons.

I. INTRODUCTION

There are many circumstances under which the simulation of massive bodies under the effects of gravity may be desired, such as to predict the future positions of objects in our solar system. Currently the most accurate physical description of gravity comes from Einstein's relativity, although in many situations this can be approximated very well by Newton's laws of motion and gravity, which proved to be sufficiently accurate for the Apollo space missions to the moon. Whilst the 2-body problem with Newtonian mechanics does have an analytical solution, there is as yet no similar means of calculating the future state of a system of an arbitrary number of bodies with only the initial conditions; this is known as the n-body problem. In this problem the acceleration experienced by each body can change with time in a complicated manner, so for the general case the acceleration cannot be expressed as a clear function of time, and so numerical methods are required to approximately integrate it. By twice numerically integrating the acceleration over a time period, we can obtain an approximation for the position of the body an amount of time in the future. We will do this using two different methods, the Euler method and the Euler-Cromer method, and compare the results from each for the same simulation scenario. The Euler method is not very accurate and can be unstable for oscillatory situations [1] such as planetary orbits, so it is not expected to perform as well as the Euler-Cromer method.

From the definition of a definite integral, integrating a variable acceleration over a time interval to give the change in velocity over the interval can be expressed as the infinite sum

$$\Delta v\left(t\right)=\int_{t_0}^{t}a(t)dt\equiv\lim_{N\to\infty}\sum_{i=1}^{N}a\left(t_i\right)\left(\frac{t_f-t_0}{N}\right) \tag{1}$$

where $a\left(t\right)$ and $\Delta v\left(t\right)$ are the acceleration and the change in velocity over a time interval as functions of time, with the start and endpoints of the time interval as t_0 and t. This integral can be thought of as the sum of the acceleration at a time t_i multiplied by a small time step of $\frac{1}{N}$ times the size of the time interval over an infinite number N of time steps, because we assume that over a small enough time step the acceleration can be approximated as constant. If we instead consider a finite but still large value of N, it's possible to obtain an approximate value for the integral and therefore also the velocity at time t, which we can express as the finite sum

$$\vec{v}\left(t\right)=\vec{v_0}+\Delta\vec{v}\approx\sum_{i=1}^{N}\Delta t\cdot\vec{a}\left(t_i\right)=\vec{a_1}\Delta t+\vec{a_2}\Delta t+...+\vec{a_N}\Delta t \tag{2}$$

where Δt is the time step, $\frac{t-t_0}{N}$, $\vec{v_0}$ is the velocity at time t_0, and $\vec{a_i}$ is the acceleration at time t_i. This can be expressed in a vector form, as we apply the same integral to each of the components of the acceleration vector. A similar expression

$$\vec{r}\left(t\right)\approx\sum_{i=1}^{N}\Delta t\cdot\vec{v}\left(t_i\right)=\vec{v_1}\Delta t+\vec{v_2}\Delta t+...+\vec{v_N}\Delta t \tag{3}$$

can be obtained for the position vector $\vec{r}$ from the velocity. So given the acceleration $\vec{a_j}$ at a point in time t_j, the velocity $\vec{v_j}$ at that time can be found, and from that the position $\vec{r_j}$ at that moment can also be found, as shown in Equations 4 and 5.

$$\vec{v_j}\approx\sum_{i=1}^{j}\Delta t\cdot\vec{a}\left(t_i\right)=\vec{a_1}\Delta t+\vec{a_2}\Delta t+...+\vec{a_j}\Delta t \tag{4}$$

$$\vec{r_j}\approx\sum_{i=1}^{j}\Delta t\cdot\vec{v}\left(t_i\right)=\vec{v_1}\Delta t+\vec{v_2}\Delta t+...+\vec{v_j}\Delta t \tag{5}$$

The acceleration at a time t_j is given by the sum of the gravitational acceleration from each other body using their masses and current positions $\vec{r}_{\text{body}}(t_j)$ as in Equation 6, and thus after every time step it is possible to calculate an approximate acceleration during that time step, use that to calculate the change in velocity and position over the step, and then use the new position to calculate the new acceleration to be used over the next step.

$$\vec{a_j}\approx\sum_{\text{body}}\frac{GM_{\text{body}}}{|\vec{r}_{\text{body}}(t_j)-\vec{r_j}|^3}\left(\vec{r}_{\text{body}}(t_j)-\vec{r_j}\right) \tag{6}$$

This is the basis behind the two numerical methods we will use in the simulations. The Euler method has the implicit formulae

$$\vec{v}_{n+1} = \vec{v_n} + \vec{a_n}\Delta t \qquad \vec{r}_{n+1} = \vec{r_n} + \vec{v}_n\Delta t \tag{7}$$

and the Euler-Cromer method is given by

$$\vec{v}_{n+1} = \vec{v_n} + \vec{a_n}\Delta t \qquad \vec{r}_{n+1} = \vec{r_n} + \vec{v}_{n+1}\Delta t \tag{8}$$

with the only difference between the two being that in the Euler-Cromer method the position is updated using the velocity at the end of the step [1].

II. DESIGN

An object oriented approach was taken to the simulations, by creating a Particle class which handles the state of each body as well as updating its position, velocity and acceleration after a given time step. By changing the method parameter passed to the class' update function, it was possible to choose which numerical integration method to use. The file containing this class could be imported as a module by the simulation script containing the simulate method, which runs the simulation, as well as code to initialise the Particle objects. After calling the simulate method and passing it the bodies to be simulated along with the number of time steps and the length of each, as well as the numerical method to use, the simulation would be run, updating all the Particle objects' accelerations and then positions and velocities a step at a time. Every so many steps the time in the simulation and the state of the bodies at that time was recorded, by appending a list containing the time and copies of the Particle objects to the list variable Data. Once the required number of iterations was completed, the simulate method would write the Data array to a file, and also return the array.

Due to this OOP approach, the program can accept an arbitrary number of bodies, each of which can have a unique mass and set of starting conditions. This means it can calculate simple cases involving only two Particle objects, such as the Earth and a projectile, or it can calculate more complex cases like the orbits of the planets and moons in the solar system, at the cost of requiring more time per simulation step and more smaller time steps for increased accuracy. Because each Particle needs to calculate its acceleration from the sum of acceleration contributions from all the other objects, the computation required for n Particle objects is of order $O(n^2)$.

Additionally, a module containing the ParticleData and Attribute classes was created to simplify handling of the data generated by a simulation, by extracting the properties from the copies of the Particle objects stored in Data in the simulation output file. The resultant arrays of each property over the duration of the simulation were stored in Attribute objects which were themselves stored as attributes of the ParticleData object for the corresponding particle. This allowed the Attribute objects to be accessed easily, for example ParticleData.v would give the Attribute object containing the velocity 3-vectors for that particle. The Attribute objects themselves made axis labeling easier, as calling the object would return an appropriate axis label along with an array of magnitudes of the property; calling the objects with an index argument would return an array of the corresponding component of the property, such as all the x components, along with an appropriately prefixed label.

Variations of a plotting script were used to produce matplotlib plots, and a utility script for extracting the required ephemeris data automatically from the text files downloaded from JPL HORIZONS was also used.

III. RESULTS

A. Comparing Projectile Motion to Analytical Solution

The trajectory of a 1kg mass projected from the Earth's surface at an angle of 45 degrees above the horizontal, with both horizontal and vertical velocity components equal to $10ms^{-1}$, was simulated using the programs, and as expected gave a parabolic trajectory. Under the assumption of near constant acceleration, the equations of projectile motion were used to plot the expected trajectory of the projectile. The two trajectories are shown overlaid over each other in Figure 1, and they appear to be extremely similar. Plotting the distance between the simulated and kinematic positions at each point in time gives Figure 2, showing that there is very little deviation in the simulation from what we would expect given constant acceleration, as even at the end of the simulation the projectile is only $12cm$ away from where we'd expect it to be from projectile motion equations, which is small compared to the horizontal distance of almost $20m$ travelled by the projectile.

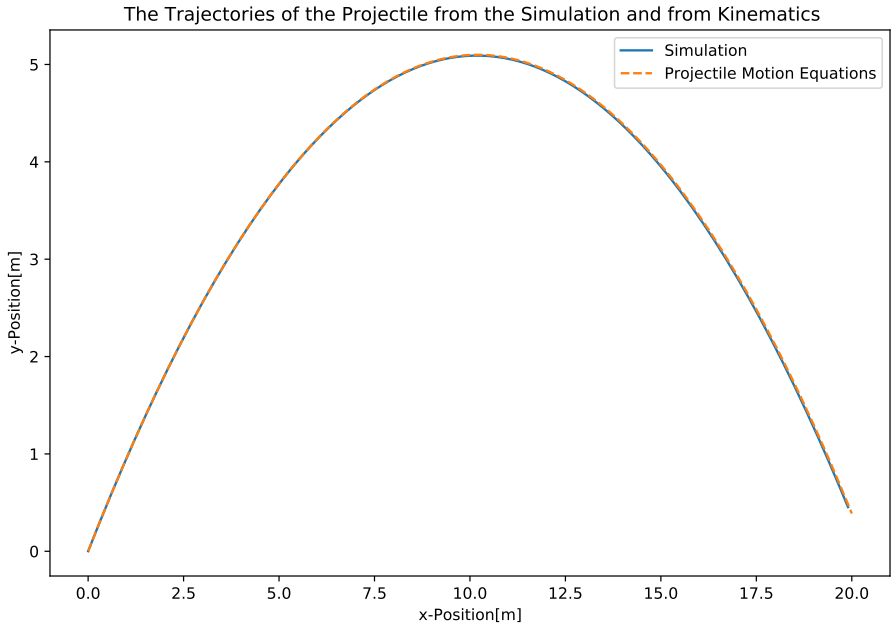

FIG. 1. The trajectory of the projectile produced by the simulation, with the trajectory predicted by equations of motion assuming constant acceleration overlaid on top.

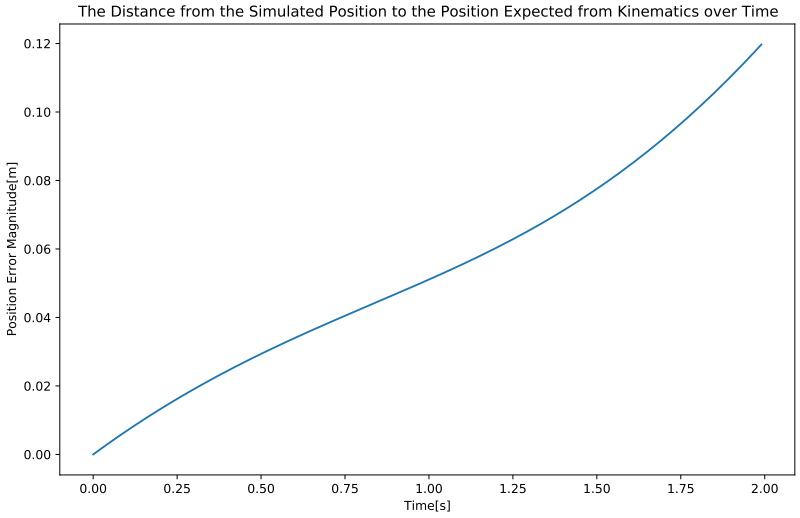

FIG. 2. The distance between where the simulation predicts the projectile to be and where the equations of motion assuming constant acceleration predict it to be over the course of the simulation.

B. Errors in Rectangle Rules

In the Euler method used, the integration of velocity to give position is done using the left-hand rectangle rule, whereas in the Euler-Cromer method this is done using the right-hand rectangle rule. In both methods the error is of the order $O(\Delta t)$ for the step size Δt, but the Euler-Cromer method is expected to be more accurate. This is because both methods integrate acceleration to give velocity using the left-hand rectangle rule, which will give an underestimate for increasing acceleration and an overestimate for decreasing acceleration, but the Euler-Cromer

method will then give an overestimate for increasing velocity and an underestimate for decreasing velocity, serving to cancel it out somewhat.

With initial positions from the ephemerides for the start of the year 2000, the 8 solar system planets and the Sun were simulated over the course of a calendar year, and then their final positions compared to ephemerides for the start of the year 2001. A chart was plotted (see Appendix A) which showed quite large percentage errors that seemed to be inversely proportional to the distance from the Sun, however it became clear after investigation that the length of time being simulated was equivalent to 365 days, yet 2000 was a leap year with 366 days, hence the planets in the simulation were a day behind the ephemeride positions in their orbits. After simulating a further day, the size of the percentage errors was much more reasonable, and these are shown in Table I for the different bodies using the two methods.

	Error in Position Magnitude (%) [3sf]	
Body	Euler-Cromer	Euler
Sun	0.00144	0.00159
Mercury	0.0159	7.19
Venus	0.00420	0.502
Earth	0.00187	0.134
Mars	0.00380	0.0112
Jupiter	0.00154	0.00157
Saturn	0.000587	0.000601
Uranus	0.000183	0.000185
Neptune	0.000107	0.000103

TABLE I. The distance between the final position of each body and the position expected at that time from JPL ephemerides, as a percentage of the magnitude of the expected position vector.

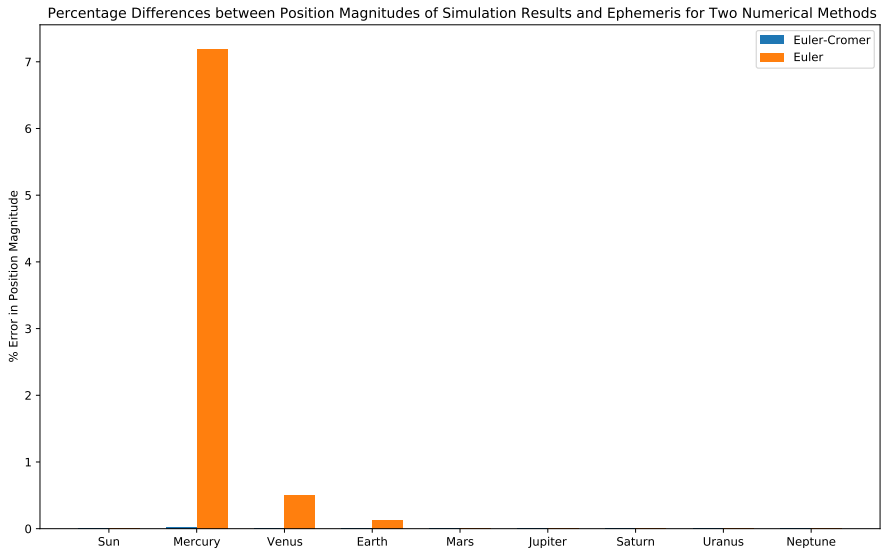

FIG. 3. A chart showing how the percentage position error compares between bodies for the Euler method, and how much greater they are compared to the Euler-Cromer method.

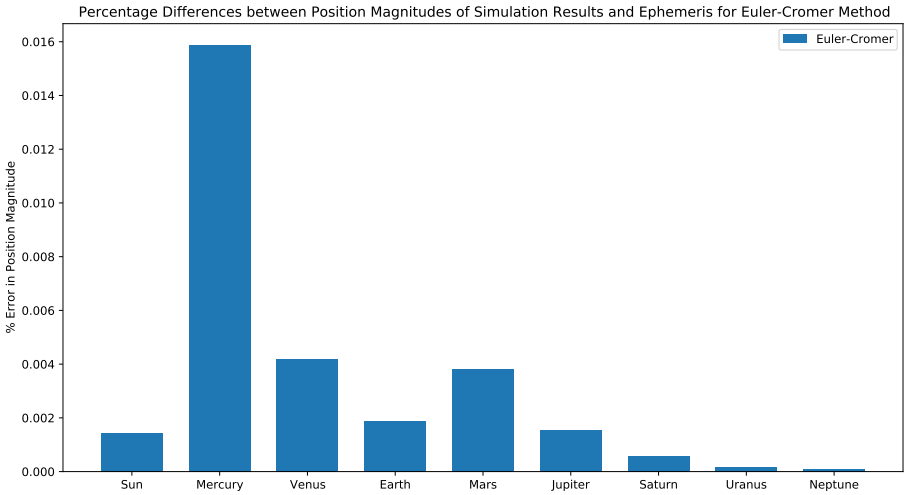

FIG. 4. A chart showing how the percentage position error compares between bodies for the Euler-Cromer method.

C. Moons Orbiting Planets

The Sun, the 8 planets, and the 7 biggest moons (the Moon, the Galilean moons, Titan and Triton) were all simulated over a year to observe whether the moons would orbit their planets as expected. The most complicated part of the system was around Jupiter, with 4 moons all orbiting it. Due to the relatively short period of the moons' orbits, the time step was reduced to $100s$ for this. Plotting the trajectories of the moons relative to Jupiter showed consistent, near-circular elliptical orbits as we would expect (see Figure 5), and plotting their paths relative to the Sun gave shapes similar to a brachistochrone curve (see Figure 6), which is what we would expect from such motion.

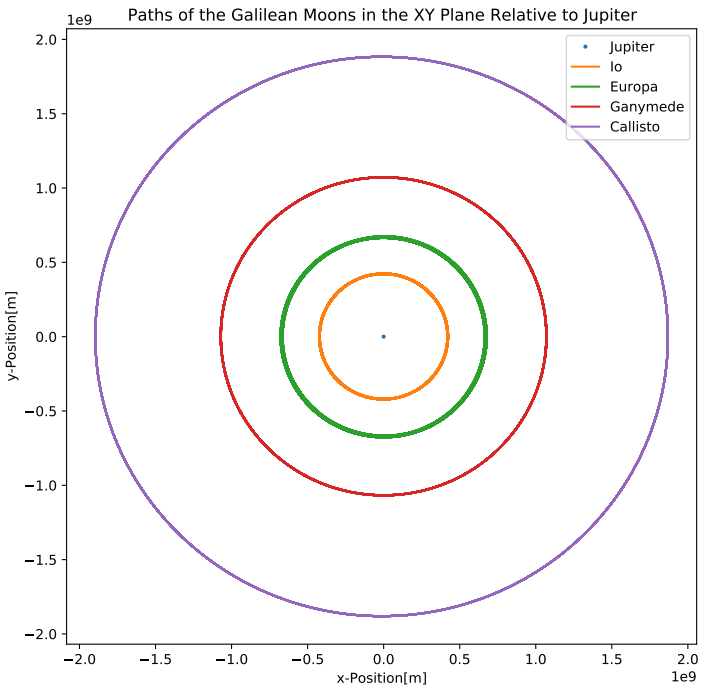

FIG. 5. The orbits of the four Galilean moons around Jupiter, from a simulation of the planets and major moons of the solar system.

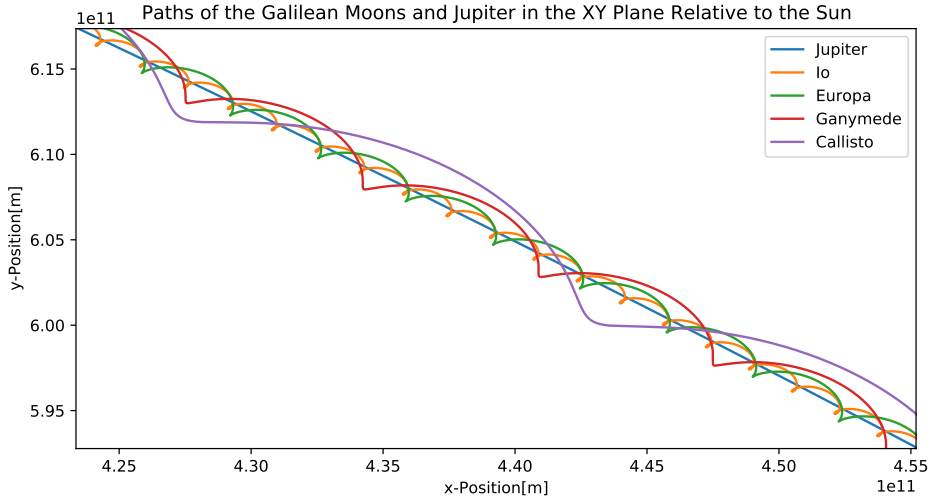

FIG. 6. The paths traced out by the positions of the Galilean moons and Jupiter relative to the Sun.

IV. SUMMARY

It has been shown that even a simple numerical integration method can simulate the orbits of planets and their satellites with quite high accuracy. The Euler-Cromer method resulted in position errors over a year of at most 0.0159% for Mercury, compared to 7.19% for Mercury using the Euler method. Only for Neptune was the percentage error lower with the Euler method, and then only by 2×10^{-6}%. Therefore it is clear that the Euler-Cromer method is generally more accurate, except for the case of slower, further out objects, where the difference becomes smaller until the Euler method becomes more accurate for Neptune, and presumably beyond.

However, there are many more advanced methods that would likely be more accurate, such as the Runge-Kutta methods, and these are worth investigating in future. Additionally, Newtonian physics is not a perfect model, and it would be interesting to implement gravity based on relativity in order to observe effects such as the precession of Mercury's orbit. Further automation of accessing epheremerides, perhaps through an API, could make adding many smaller bodies feasible, which should provide an even more accurate simulation of the solar system. And finally, it should be investigated whether the simulation handles more complicated orbits and trajectories successfully, for example with objects at Lagrange points or for a spacecraft performing a serious of gravity assists through the system.

[1] https://modules.lancaster.ac.uk/course/view.php?id=11842

Appendix A: Percentage Errors from Incorrect Simulation Time

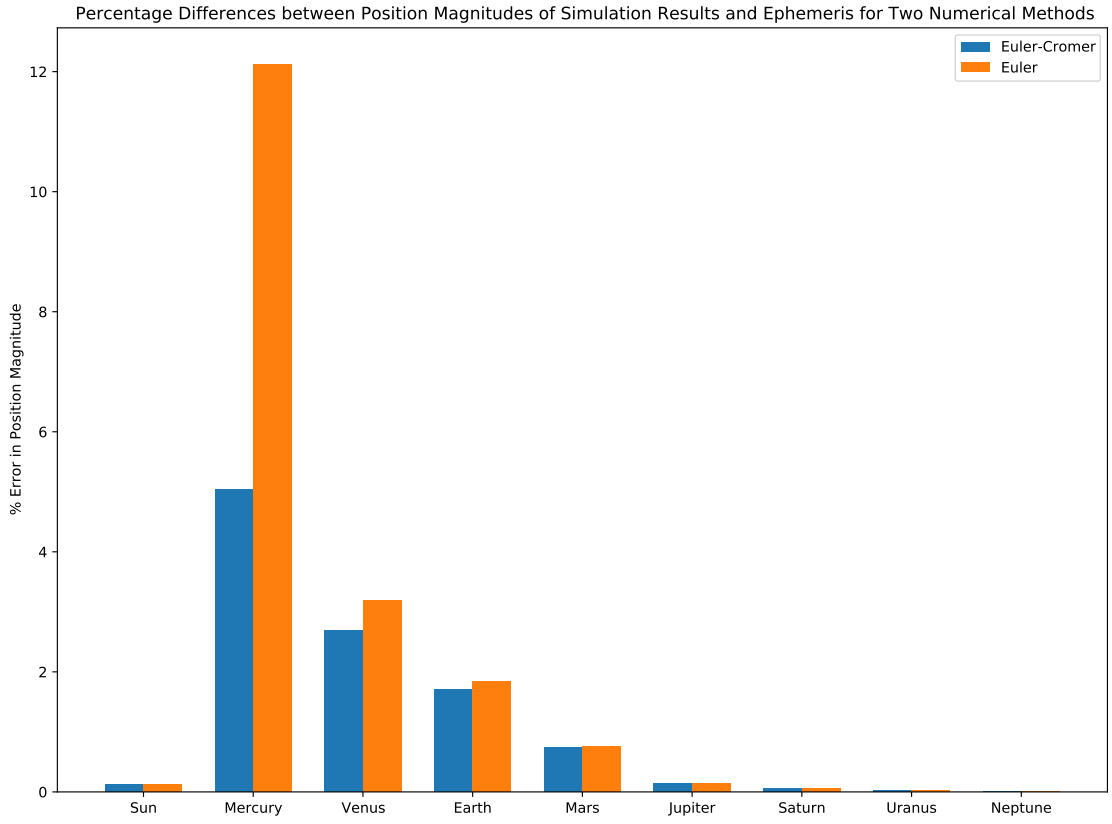

FIG. 7. The distance between the final position of each body after 365 days and the position expected after 366 days from JPL ephemerides, as a percentage of the magnitude of the expected position vector.

Appendix B: Planets' Paths During the Year 2000

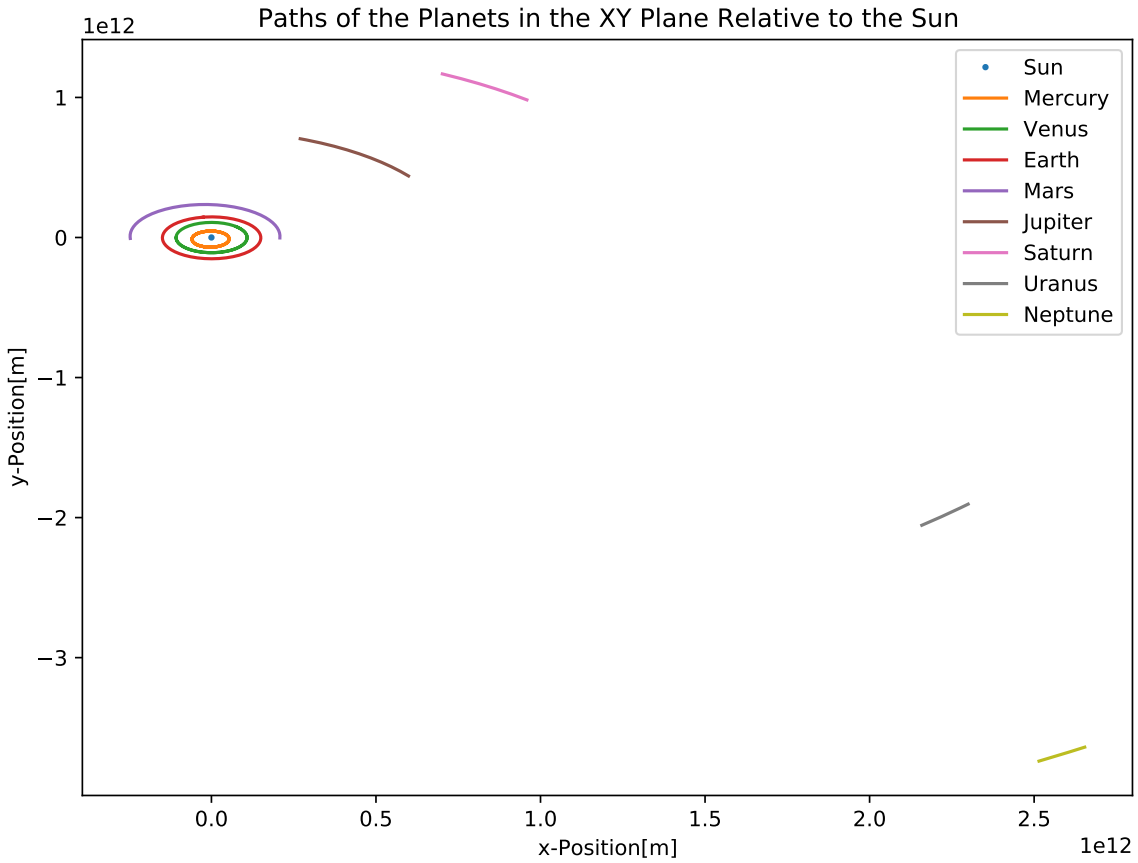

FIG. 8. The trajectories of the 8 planets relative to the Sun from the start of the year 2000 to the start of 2001.