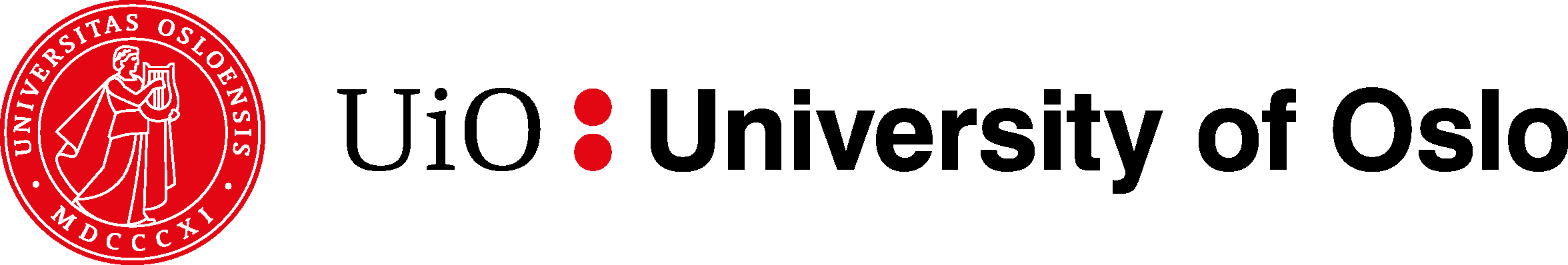
Computational Physics

Project 4



Hunter Wilhelm Phillips (*hunterp*)

November 19, 2018

GitHub Repository at <https://github.com/robolux/Computational_Physics>

**Abstract**

This project explores the implementation of different integration methods to build a working model of the solar system. Object oriented programming is utilized to develop a class structure for the simulation. The computation of the celestial bodies orbits was the core part of the algorithmic development. This requires the integration of Newton’s law of motion and for this project; Forward Euler and Velocity Verlet were chosen. The project started with modeling a two-body system (Earth-Sun) and progressed until a complete solar system was simulated. Throughout the development, multiple facets of the system were verified including conservation of energy, determining stable velocity for circular orbit, and time step overshoot analysis. Benchmarks were ran comparing the two chosen integrators, which resulted in showing the inaccuracies of the Forward Euler method. Various experiments were conducted with the algorithm such as changing the mass of Jupiter then analyzing the result on the associated celestial bodies and calculating the perihelion precessions of Mercury verifying that there are indeed 43. The results confirmed that an integrator and the associated law of motion coupled with object orientation produces a stable simulation of the solar system.

1. Discussion of Problem Statement

In this project, the goal is to successfully simulate the solar system through the laws of motion. To achieve this two integrators were chosen, Forward Euler and Velocity Verlet to solve the coupled differential equations (DE). First, the Earth-Sun system is set to be analyzed with the resulting discretized DE’s being used as the basis for computation. Second, the algorithms are to be tested and tuned for optimal performance. This includes transferring variables into class structures to transition towards object orientation. To produce a circular orbit, the initial value for velocity must be changed to allow for stability, even with varying time steps. To verify that the conservation of energy and angular momentum is held; kinetic, potential, and angular momentum are to be calculated for analysis. A comparison between Forward Euler and Velocity Verlet is to be completed resulting in the decision to only use Velocity Verlet in the rest of the project. Escape velocity is next to be calculated and a custom β modifier is experimented with to change the gravitational force. A three-body problem is simulated next that includes the Sun, Earth, and Jupiter with varying masses of Jupiter to send the system into a disarray. A final model of the solar system that includes all planets is then modeled to verify the algorithm successfully handles the celestial bodies in our solar system. Finally, the perihelion precession of Mercury is to be calculated through the use of a relativistic correction of Newton’s gravitational force and comparing it to the uncorrected original equation. Through this, the challenges at hand have been laid out and the theory and methods explaining the solution are described in the following section.

2. Discussion of Theory & Methods

2.1 The General Ising Model

Assuming that the formation of a -dimensional lattice is coupled in a set of adjacent lattice positions. For each lattice elemental location, there is a representation of the spin of the element and in the form of a discrete variable . Each individual spin configuration is connected to a specific state of the lattice.

If there are two elemental locations, *i* and *j*, in the set, one of them will have an interaction . This same characteristic site will also have an external magnetic field coupled with it. The energy of the system specified can be given by the following Hamiltonian as seen in (1):

(1)

Where the first summation is over pairs of adjacent spins

displays that *i* and *j* are nearest neighbors

is the magnetic moment

The Boltzmann distribution gives us the probability of a certain configuration as shown in (2):

(2)

Where is the normalization constant

The functioning

This showcases that an increase in temperature directly affects the possibility of finding a system in a particular configuration in a negative manner.

This leads into the characterization of a magnetic system and the most important expected values to be calculated. The first is mean energy as seen in (3).

(3)

Where signifies the number of possible configurations

The second is mean magnetization as seen in (4).

(4)

Where is represented for all configurations

The third is magnetic susceptibility as seen in (5).

(5)

The fourth is heat capacity as seen in (6).

(6)

These reduced equations form the basis for the project and allow for the specific model to be deduced. The Ising model can be classified as having a ferromagnetic interaction if and for the purposes of this simulation, this case will be used.

2.2 Simplified Ferromagnetic Ising Model

To ease the complexity of this project, the general Ising model as described in *Section 2.1* will be simplified. The first step is to assume that there is no external magnetic field in play and with this assumption, (1) has it second summation go to zero, leaving (7):

(7)

The coupling constant J is assumed to be a constant which enables the removal of it from within the summation forming (8).

(8)

The final assumption is that the model is ferromagnetic in nature which means that neighboring spins are aligned, resulting in lower energies.

2.3 Case: L = 2 × 2 Ising Model

To prepare for later comparisons with numerical results, the analytical expressions for an L = 2 x 2 case are explored. Using (3), (4), (5), and (6); the following, Table 1, shows all of possible states for an L = 2 x 2 case.

|  |  |  |  |
| --- | --- | --- | --- |
| State | Symmetries | Energy (J) | Mean Magnetization |
|  | 1 | -8 | 4 |
|  | 4 | 0 | 2 |
|  | 4 | 0 | 0 |
|  | 2 | 8 | 0 |
|  | 4 | 0 | -2 |
|  | 1 | -8 | -4 |

**Table 1: Case L = 2 x 2 all 16 Possible Configurations**

To compute all the associated physical quantities as shown above, the closed form expressions can be given by the following partition function (9):

(9)

This leads to the expected energy forming into (10):

(10)

Using (4) the mean magnetization for this particular system can be computed into (11)

(11)

This leads to the expected absolute magnetization coming out as (12):

(12)

The expected value for specific heat is characterized in (13):

(13)

By inserting (10) into (13) the expected specific heat analytical equation (14):

(14)

Rewriting (5) into (15) gives the basis for the susceptibility of magnetism:

(15)

By combining (11) and (12) into (16), the second integral part for the susceptibility of magnetism can be derived:

(16)

Finally, by inserting (16) into (15), (17) can be formulated:

(17)

These analytical results are the basis for future comparisons with numerically obtained results.

3. Discussion of Algorithms

3.1 Metropolis

One of the key facets of this project was to explore object oriented programming and applying it to the algorithmic process. Since object oriented program structures perform incredibly poorly with respect to scientific programing, all computations were performed on arrays. The program uses Python coupled with the powerful array calculation library numpy to access memory efficiently to ensure reasonable runtime. The object orientation came forth in the use of a class structure to make expandability clean when calling the functions. This allowed a compact format as seen below that showcases the benefits of applying object orientation to this projects structure.

**Class Structure**

**class** solar\_system:

**def** \_\_init\_\_(self):

self.n\_planets = 0

self.distance = 0

**def** create\_planetary\_object(self, x0, y0, vx0, vy0, mass, adjust\_sun=True):

**if** self.n\_planets == 0:

self.planetary\_positions = nmp.array( [[x0, y0]] , dtype=nmp.float64)

self.planetary\_velocities = nmp.array( [[vx0, vy0]] , dtype=nmp.float64)

self.planetary\_masses = nmp.array( mass , dtype=nmp.float64)

**else**:

self.planetary\_positions=nmp.append(self.planetary\_positions, [[x0, y0]], axis=0 )

self.planetary\_velocities=nmp.append(self.planetary\_velocities, [[vx0, vy0]], axis=0)

self.planetary\_masses=nmp.append(self.planetary\_masses, mass )

**if** adjust\_sun:

self.repop\_sun()

self.n\_planets += 1

The *solar\_system* class contains all pertinent information about the initial conditions of the system with *self* containing the instance attributes of the method. This method allows for ease of customization when implementing the functions that follow.

3.2 Flip

The function containing the Forward Euler integrator derived from (14) and (15) can be seen below.

**Forward Euler Implementation**

**def** forward\_euler(self, P, V, p\_new, V\_new, dt, acc\_method):

length = **len**(P)

**for** n **in** **xrange**(length):

p\_new[n] = P[n] + V[n]\*dt

V\_new[n] = V[n] + acc\_method(P, V[n], n,self.planetary\_masses)\*dt

The derivations found in (19) and (20) are implemented into the Velocity Verlet integrator as showcased below.

3.2 Monte Carlo

The function containing the Forward Euler integrator derived from (14) and (15) can be seen below.

**Velocity Verlet Implementation**

**def** velocity\_verlet(self, P, V, p\_new, V\_new, dt, acc\_method):

length = **len**(P)

acc\_p = nmp.zeros((length, 2), dtype = nmp.float64)

**for** n **in** **xrange**(length):

acc\_p[n] = acc\_method(P, V[n], n, self.planetary\_masses)

p\_new[n] = P[n] + V[n]\*dt + 0.5\*acc\_p[n]\*dt\*\*2

**for** n **in** **xrange**(length):

acc\_p\_new = acc\_method(p\_new, V[n], n, self.planetary\_masses)

V\_new[n] = V[n] + 0.5\*(acc\_p[n] + acc\_p\_new)\*dt

4. Discussion of Results

4.1 System Specifications

A few notes about the system running the algorithms before analyzing the results should be helpful in gauging performance between differing hardware as seen in *Table 1*.

|  |  |
| --- | --- |
| Operating System | Mac OS X |
| Processor | i7 - 2.2 GHz |
| Memory | 8 GB 1600 MHz DDR3 |
| Hard drive | 128 GB SSD |

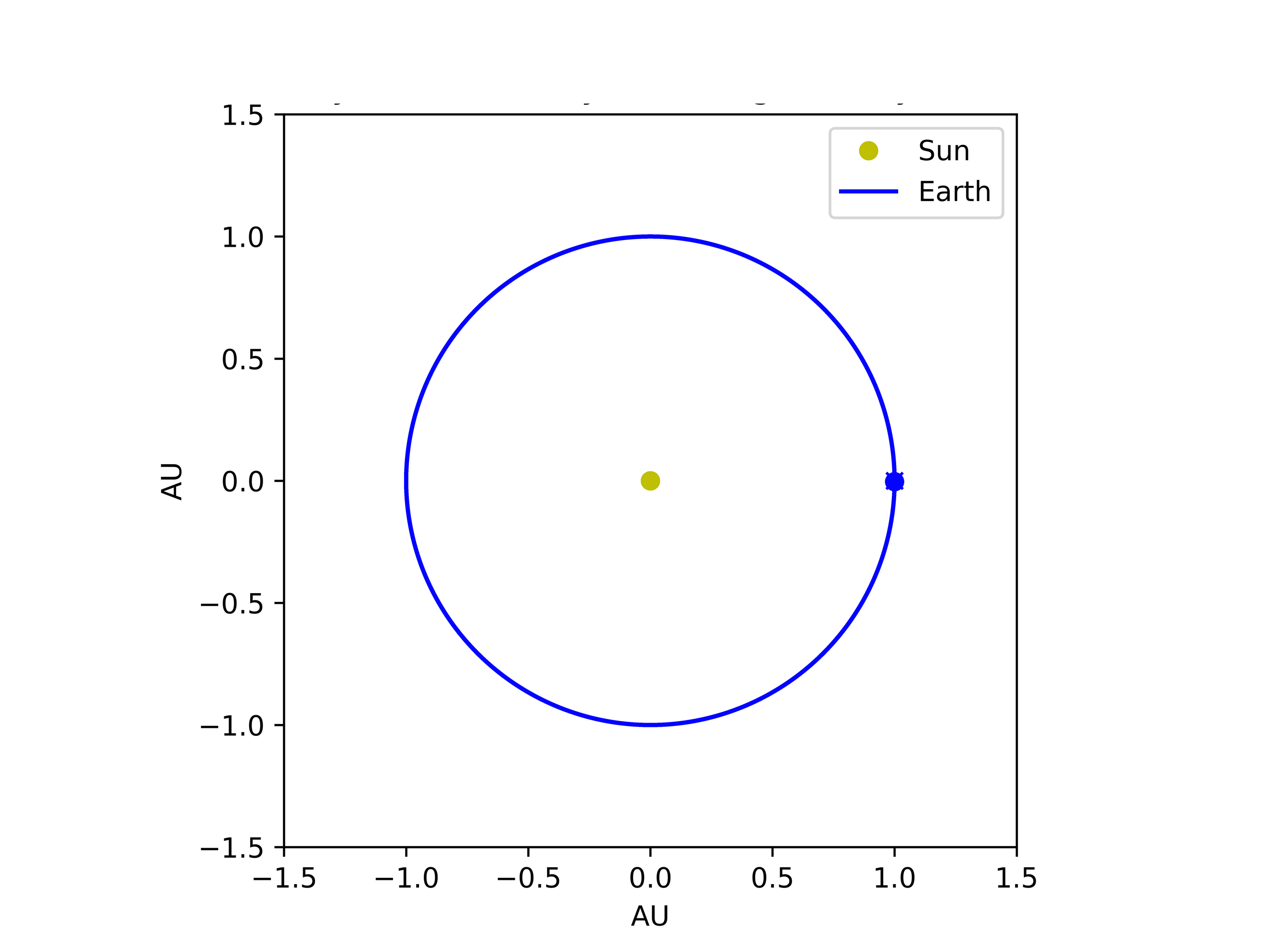
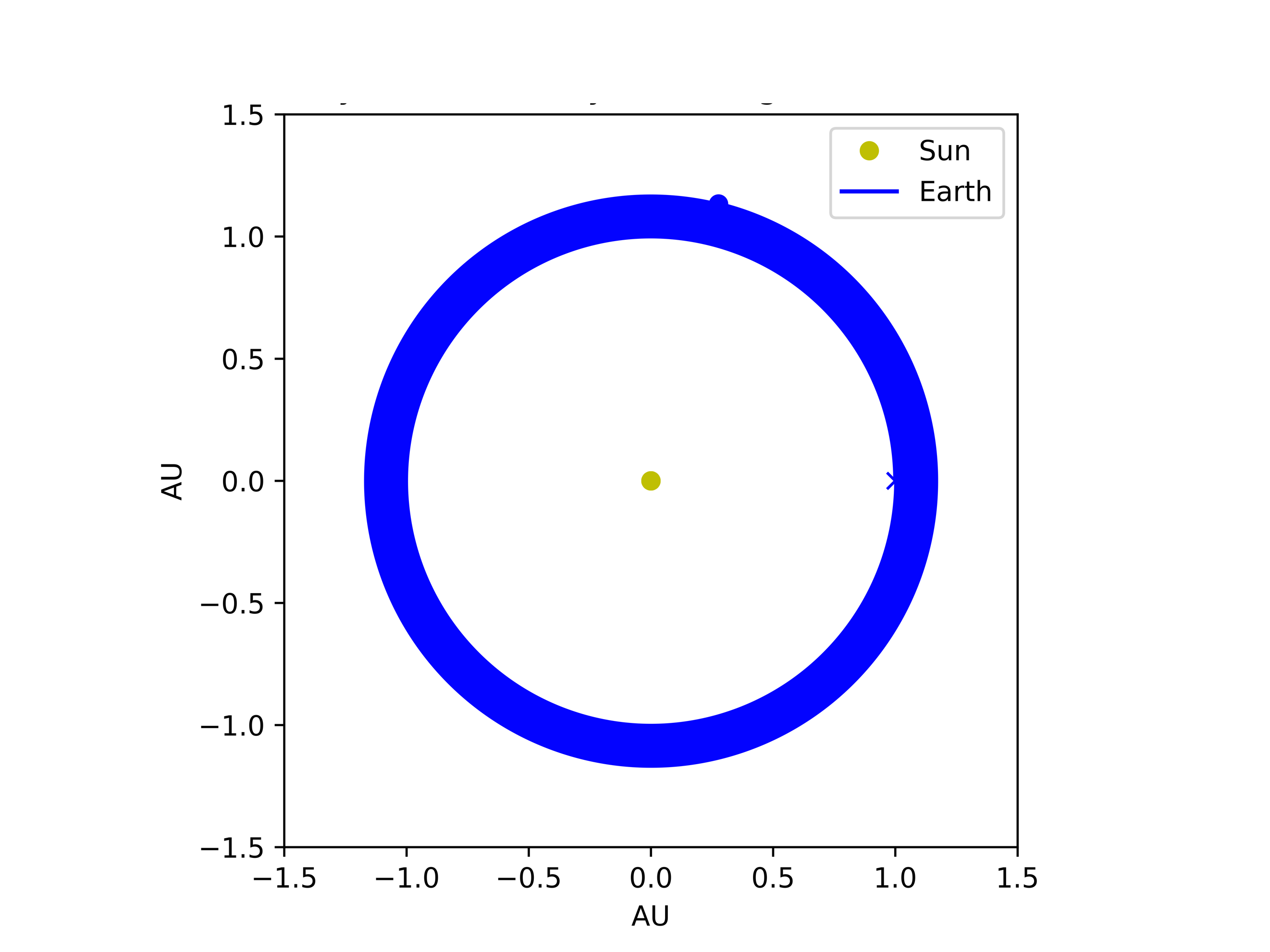
**Table 1: System Specifications**

With the testing conditions being established we can now dive into the data recorded.

4.2 Ising Model with L = 2 x 2

The first test of the code was to simulate a simple Earth-Sun system using the two integrators developed coupled with classical acceleration. To compare the accuracy of the Forward Euler and Velocity Verlet integrators were run over 50 years.

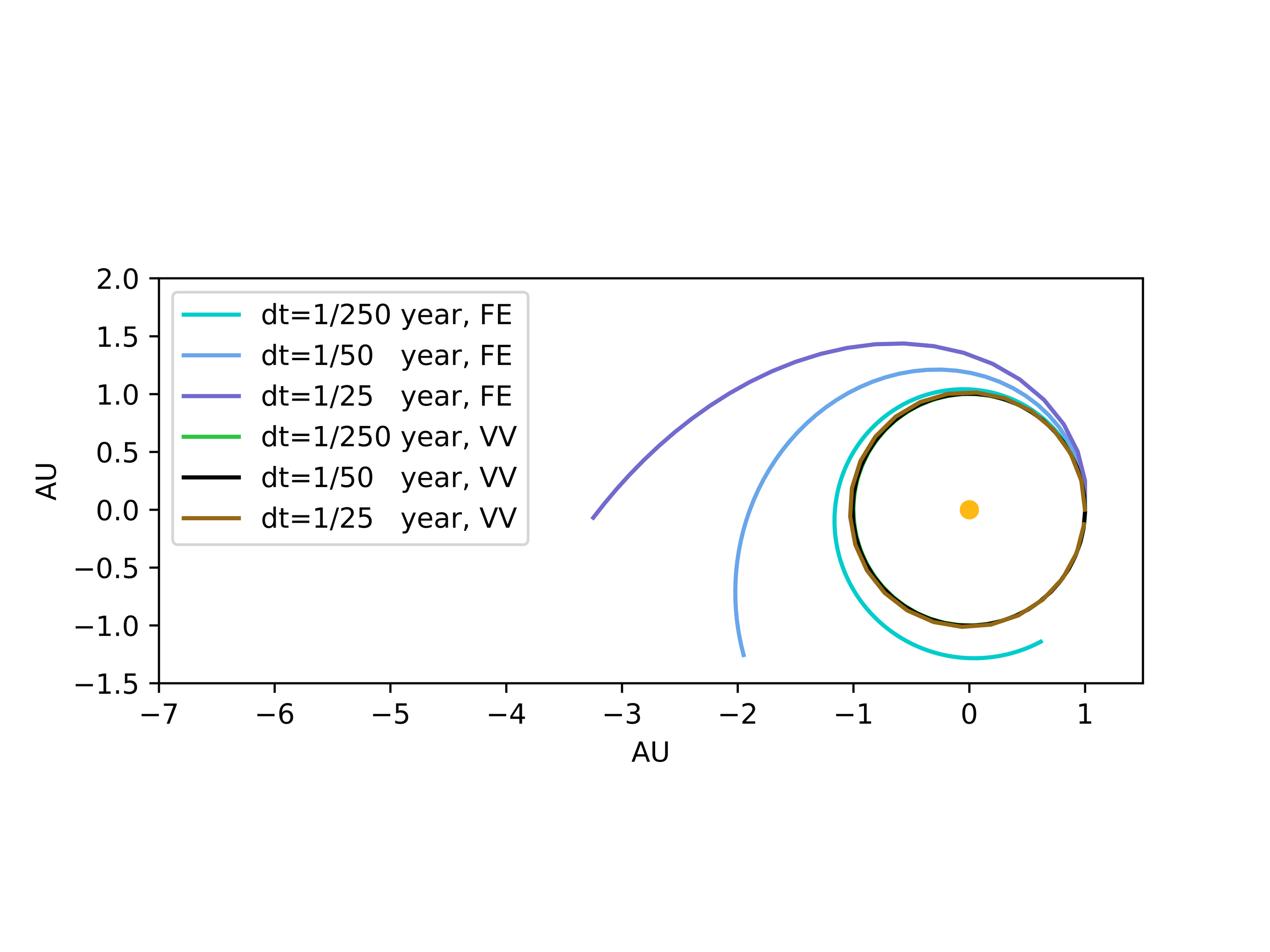
**Figure 1: Both Integrators ran over a period of 50 years with dt = 5e-05 as shown with Forward Euler (left) and Velocity Verlet (right)**



It is fairly obvious that the Velocity Verlet algorithm is superior with stability as seen in *Figure 1* with respect to the minimal drift over the time period. It is important to not immediately discredit a solver without proper testing, so the stability of the integrators with various time steps were tested.

**Figure 2: Differing Time steps (dt) with**

**Forward Euler (FE) and Velocity Verlet (VV) Integrators**



It can be seen in *Figure 2* that the orbit of the Earth-Sun system is stable and does not escape or drift with the Velocity Verlet integrator. The Forward Euler integrator does not fare as well and can barely hold its own with the smallest time steps. This is massive drawback, because it means Forward Euler would require small time steps, which drastically increases computation time. This leads into a comprehensive computation timing test being performed as seen below in

*Table 2*.

|  |  |  |
| --- | --- | --- |
| dt | Integrator | Computational Time (s) |
| 0.1 | Forward Euler | 0.023478 |
| 0.1 | Velocity Verlet | 0.052323 |
| 0.01 | Forward Euler | 0.223417 |
| 0.01 | Velocity Verlet | 0.474764 |
| 0.001 | Forward Euler | 2.289643 |
| 0.001 | Velocity Verlet | 4.652022 |
| 0.0001 | Forward Euler | 22.421817 |
| 0.0001 | Velocity Verlet | 46.441356 |

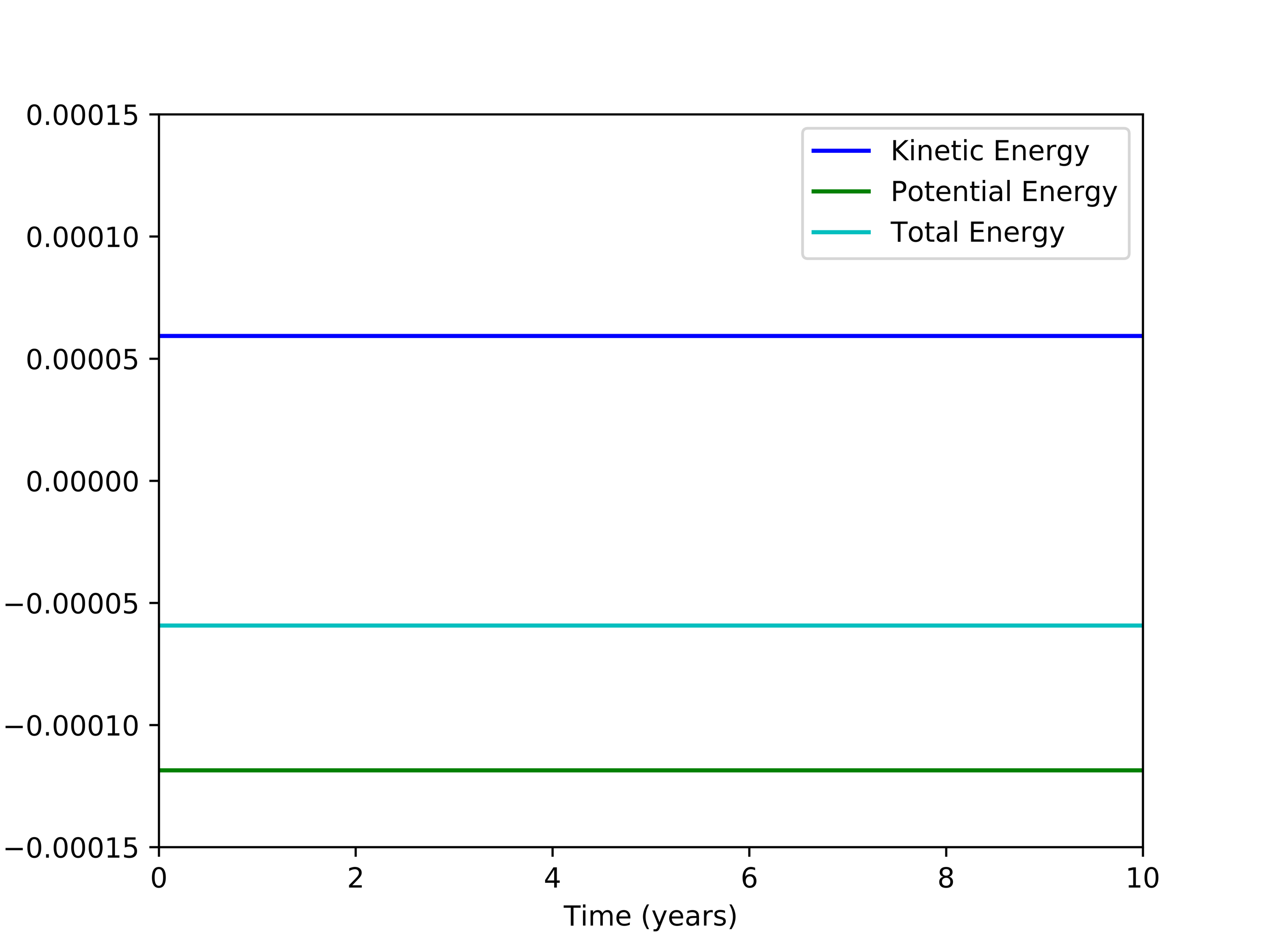
**Table 2: Comparing Computational Time of the Forward Euler and Velocity Verlet Integrators using varying time steps (dt)**

It is observed that the Velocity Verlet integrator uses approximately twice the amount of computational time as Forward Euler. This corresponds with the FLOPs analysis completed in Section 3.2.3 estimating the same time. At this point the drawbacks of the Forward Euler integrator are obvious and even though it has a better runtime, that cannot be traded for accuracy in the simulation. Following the outline set by the Problem Statement, all following simulations and tests are run solely with the Velocity Verlet integrator.

4.3 Ising Model with L = 20 x 20

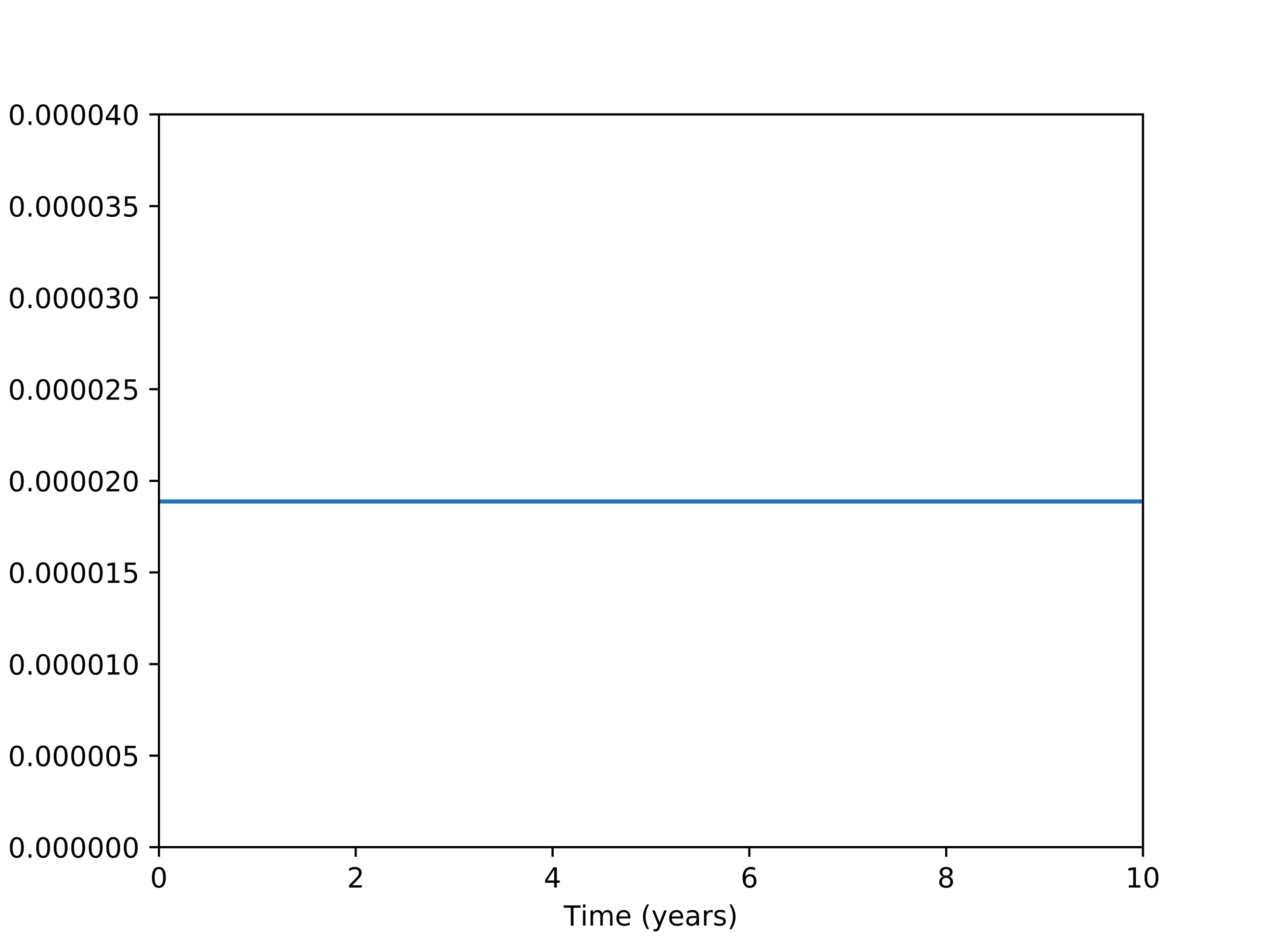
Since this simulation is isolated without external forces influencing the celestial bodies, it is expected that the total mechanical energy and angular momentum of the system is conserved throughout. Also with a circular orbit the associated kinetic and potential energy are expected to be conserved by the orthogonality of the sun with respect to the planets velocity. This in turn means that zero work is done on the planets, meaning that the potential and kinetic energy states should be unchanged.

**Figure 3: Conservation of Energy within Earth-Sun System over 10 years**



It can be seen in *Figure 3* that the conservation of energy has been preserved to a high degree of certainty and accuracy. Extending this towards the conservation of angular momentum, *Figure 4* was generated.

**Figure 4: Conservation of Momentum within Earth-Sun System over 10 years**



It is observed that the conversation of angular momentum has also been held within a fine degree. The relative error was then calculated for the angular momentum over a 10 years period and was found to be -1.504766e-13. This validates that the simulation is producing negligible error and successfully simulating the solar system.

Just as insurance that the solar system is indeed behaving as expected, the center of mass over a period of ten years shifted as seen in *Table 3*.

|  |  |  |
| --- | --- | --- |
|  | Beginning of Simulation | End of Simulation |
| Coordinates [x, y] | [0, 0] | [-3.549e-19, -6.091e-18] |

**Table 3: Center of mass at start of simulation, then after 10 years**

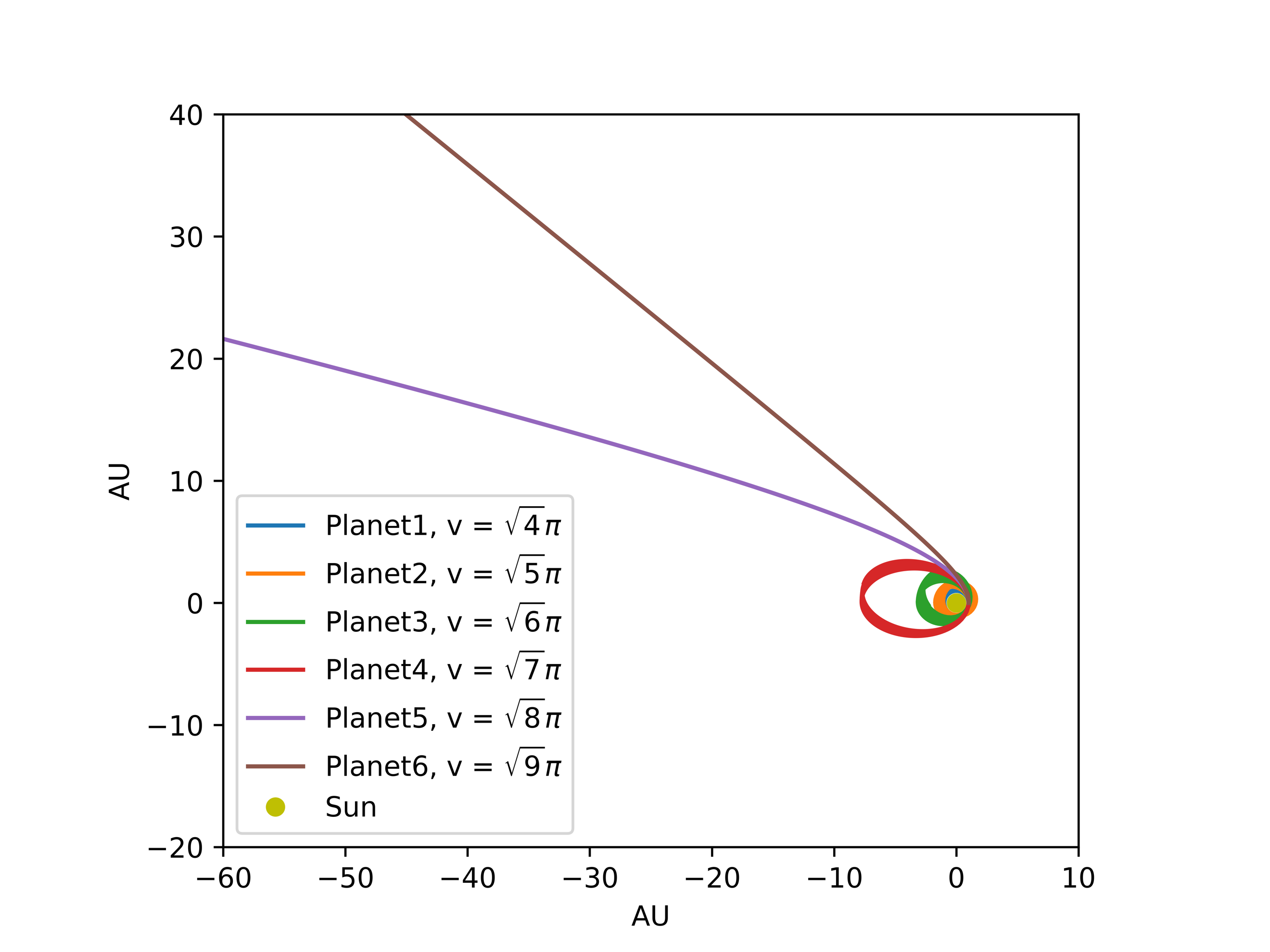
The shift in center of mass is essentially zero and validates that the simulation is stable and performing as expected.

4.4 Analyzing Probability Distribution

To maintain an orbit around a celestial body, a planet must not have to high of a velocity or it might escape from the gravitational forces of the celestial body. This is appropriately named the escape velocity of a planet and to understand this quantifiable phenomenon, *Figure 5* was produced.

**Figure 5: Varying Velocity (*v*) of Planet to Escape the**

**Gravitational Effects of the Sun**

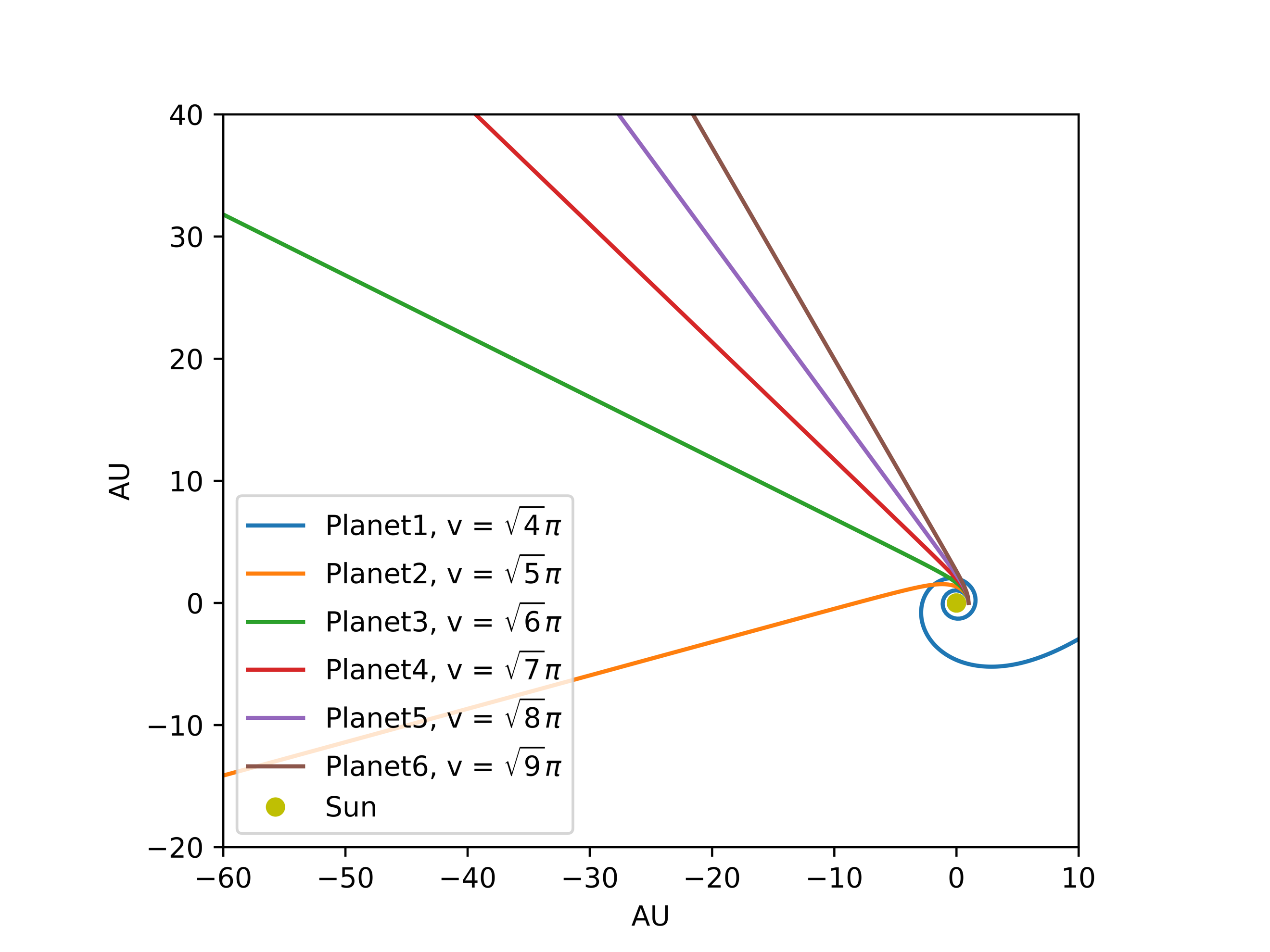
****

It can be seen that when , the planet fully escapes the gravitational effects of the Sun. Further testing could be completed to determine the exact point of release, although for this projects’ purpose the concept can be understood with this approximate velocity.

To further explore the concept of escape velocity, the modified equation developed in Section 3.3.3, (21), is graphed with the same velocities.

**Figure 6: Varying Velocity (*v*) of Planet to Escape the**

**Gravitational Effects of the Sun with Modified Gravitational Equation**



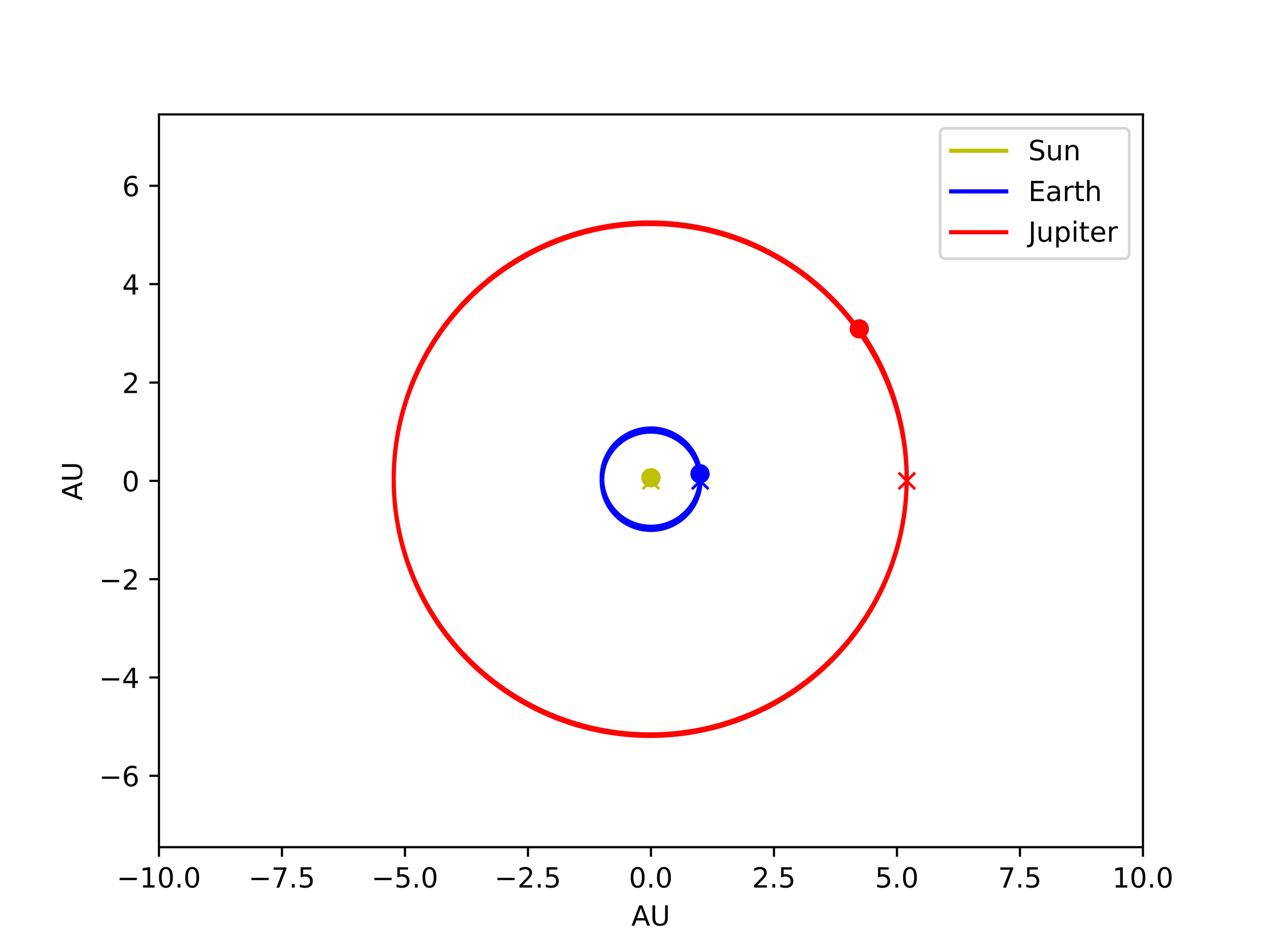
With the modified term falling in the exponent on the lower side of the fraction, this means the associated gravitational force and acceleration are reduced. Through common logic it can be deduced that this would result in a weaker pull on the planet by the Sun. By using the same test velocities as *Figure 5*, *Figure 6* shows the planets all failing to maintain an orbit and hitting their escape velocities very quickly. This confirms the hypothesis and shows another facet of modifying physics based equations.

4.5 Numerical Analysis of Phase Transitions

To continue towards a complete model of the solar system, the three-body problem is explored next. By including Jupiter into the Earth-Sun system, the simulation can be progressively tested. The automatic adjustment of the Sun was disabled for this simulation, as it depends on other celestial bodies having smaller masses. The first test is to simulate Jupiter with its real mass and the result as shown in *Figure 7* shows stable orbital paths.

**Figure 7: Simulation of Three-Body System with Jupiter**

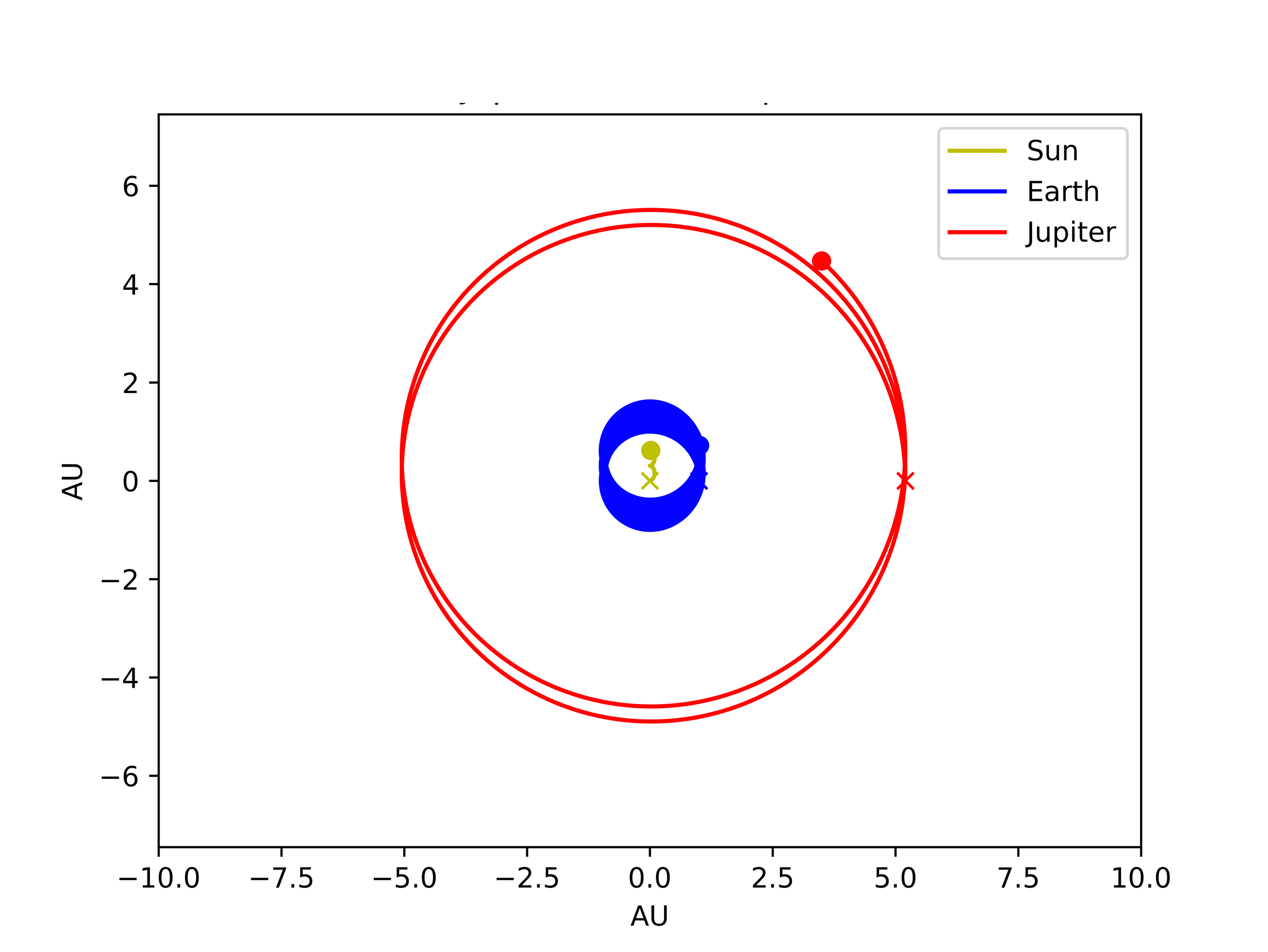
**Having its Real Mass**



To visualize the effects of a larger mass affecting the orbits of other celestial bodies, Jupiter’s mass was increased by a multiplier of 10 resulting in *Figure 8*.

**Figure 8: Simulation of Three-Body System with Jupiter**

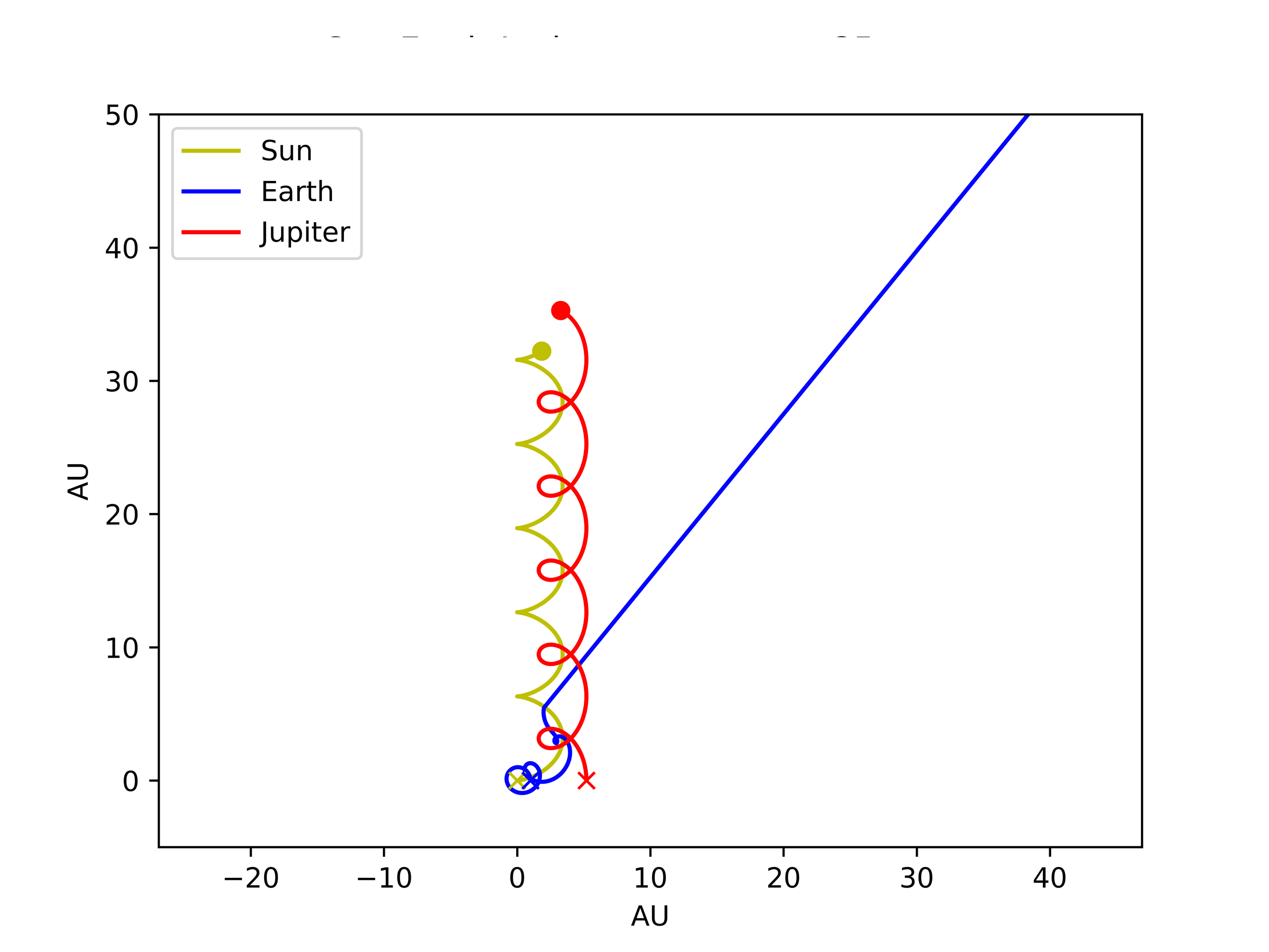
**Having a Mass Multiplier of 10**



It is evident that there is a slight effect on the Sun and Earth being pulled because of Jupiter’s 10x mass. Also, it can be observed that Jupiter itself is starting to drift because of its large mass effecting its orbit. To take this effect to the extreme, Jupiter’s mass was increased by a multiplier of 1000 resulting in *Figure 9*.

**Figure 9: Simulation of Three-Body System with Jupiter**

**Having a Mass Multiplier of 1000**

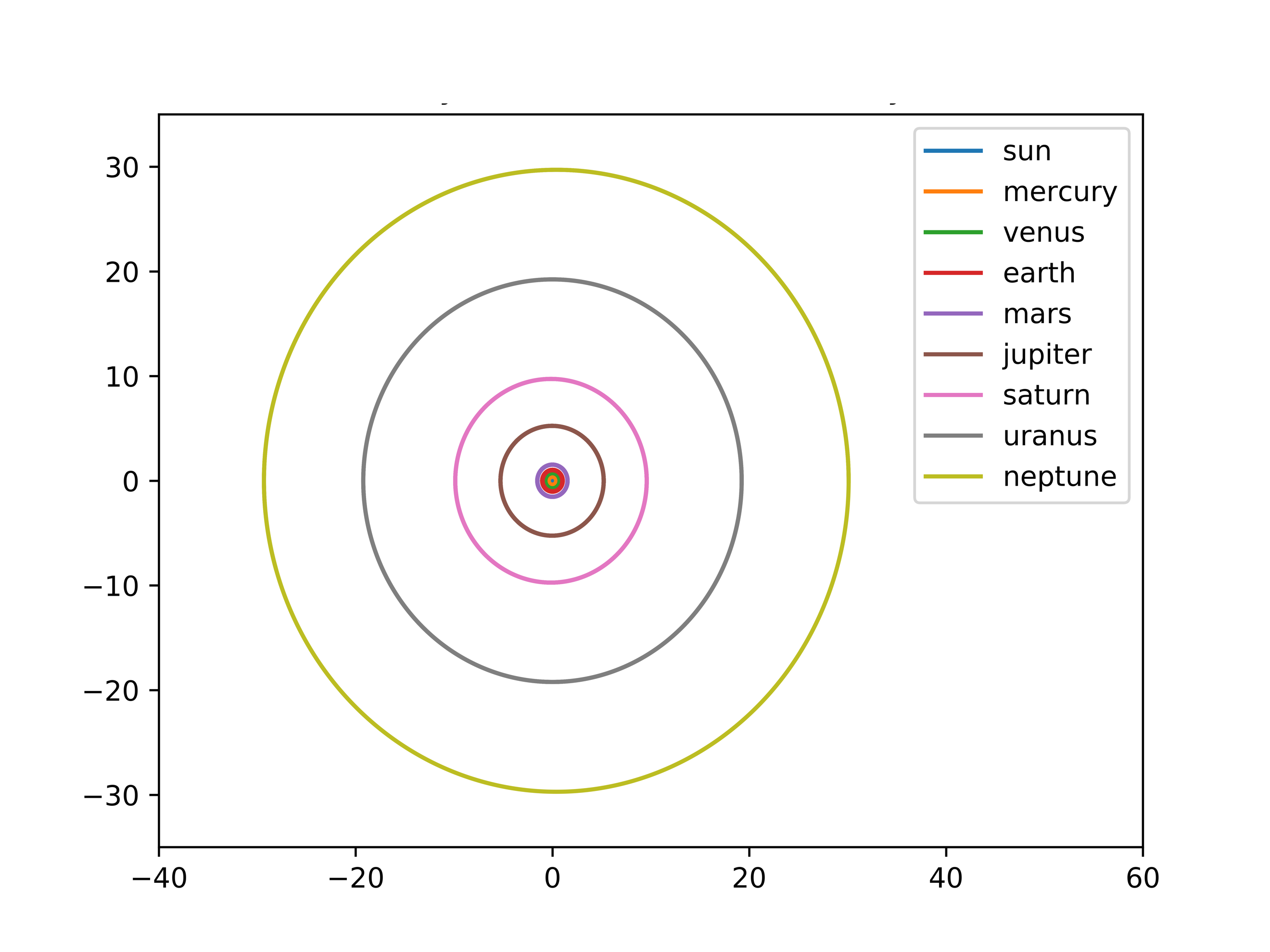


Since this multiplier places the mass of Jupiter around that of the Sun’s, they actually start orbiting each other while Earth escapes eventually into a path free of outside influence. These tests show the factors at play when a celestial bodies’ mass is altered. The observed orbits were as expected and validated that even under extreme cases, the simulation is still valid.

4.6 Critical Temperature Validation

After verifying the algorithm through extensive testing, the complete model of the solar system was simulated. The use of the NASA HORIZONS web database allowed for proper initial conditions of the planets to be obtained [2]. It can be seen in Figure 10 that the system is stable over the selected period of 250 years and validates our simulation.

**Figure 10: Simulation of the Complete Solar System with dt = 1e-06**



AU

AU

5. Conclusion

Through this project, a successful simulation of the solar system was created with an object-oriented framework. When programming the algorithm, object orientation was implemented to ease code reusability and allow for smoother development. The two integrators initially chosen, Forward Euler and Velocity Verlet, were comprehensively tested against each other. These tests showed that Forward Euler has the benefit of running two times the speed of Velocity Verlet. This however comes at the cost of very poor accuracy and an inability to hold orbital paths unless a tiny step size is chosen. With these weights taken into consideration, Velocity Verlet was chosen for all remaining tests, since the precision of the simulation was integral to the success of this project.

The conservation of angular momentum and energy were validated to be preserved within the simulation to a high degree of accuracy. The center of mass of the Sun was also observed to be stable across long periods of time. In an effort to observe a three-body system taken to the extreme, Jupiter’s mass was altered until the system collapsed. This effect was interesting when Jupiter’s mass has a multiplier of 1000 with the resulting mass being close to that of the Sun, resulting in them actually orbiting each other.

With the simulation verified to be running correctly, a complete model of the solar system was ran, which resulted in the expected orbits being produced. It was then observed that the precession of Mercury’s perihelion can only be accurately explained with the introduction of general relativity. The project concludes in an effective simulation of the solar system using ordinary differential equations executed using an object-oriented structure.

6. Future Work and Thoughts

Since my passion is in robotics and their interactive design, this project gave me an idea. It is fairly trivial to calculate a navigational path using a circular path planning algorithm [3]. Another class I am taking here at UiO is INF3490 and we just completed a Machine Learning (ML) project. By combining ML with a weighting system for swarm robots, a collaborative network could be generated with rotating bodies being fused into the ML algorithm. Each planet being represented by a robot, they can generate circular paths through their relative orbits and weights being assigned by the ML to achieve the goal at hand. This could be for example a particular search pattern in rescue operations. If for example a certain planet finds a target, the weight in the ML would be increased, and the remaining “planet” robots would start being attracted to the circular path of the best performing robot.

I thoroughly enjoyed the project and found it to be enjoyable, yet challenging at the same time. One future idea to possibly implement into the project is allowing the students to create their own solar system to simulate a science fiction environment. This would entail creating unique planets and their associated properties to characterize unique orbital paths. I usually find that allowing for some creativity even in math and science heavy projects allows for students to show their individuality.

7. References

[1] Hjorth-Jensen, Morten. "Project 3." *Computational Physics FYS3150*, Fall 2018. University of Oslo. Lectures.

[2] “HORIZONS Web-Interface.” NASA, NASA, ssd.jpl.nasa.gov/horizons.cgi#top.

[3] Han, Sung-Min, et al. “Mobile Robot Navigation by Circular Path Planning Algorithm Using Camera and Ultrasonic Sensor.” 2009 IEEE International Symposium on Industrial Electronics, 2009, doi:10.1109/isie.2009.5213204.