Computational Physics

Project 3

UiO : University of Oslo

Hunter Wilhelm Phillips (*hunterp*)

October 24, 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 Newtonian Gravitation

When studying the solar system, the core of movement is held within gravity and the associated equations. The classical law of gravitation is given by Newton's law as seen in (1):

$$F_G = \frac{GM_1M_2}{r^2} \tag{1}$$

Where F_G is the gravitational force between two bodies

G is the gravitational constant

M_1 and M_2 are the masses of the two bodies

r is the distance between the two bodies

This equation can be broken into components when viewed in three dimensions. By utilizing Newton's second law of motion, the following individual equations can be derived for the acceleration due to gravity on a chosen body i [1].

$$\frac{d^2x}{dt^2} = \frac{F_{G,x}}{M_i}, \quad \frac{d^2y}{dt^2} = \frac{F_{G,y}}{M_i}, \quad \frac{d^2z}{dt^2} = \frac{F_{G,z}}{M_i} \tag{2}$$

These set of equations seen in (2) can be individually integrated to determine the position of the celestial bodies at any provided time. It can also be seen that converting (1) into a vectorial equation in multiple dimensions, shown in (3), allows for discretized calculations in two dimensions as chosen for this project.

$$\boldsymbol{F}_G = \frac{GM_1M_2}{r^3}\boldsymbol{r} \tag{3}$$

where $\boldsymbol{r}$ is the position of the body in the cartesian coordinate system

2.2 General Relativity

The theory of general relativity introduced by Albert Einstein is currently the description of gravity accepted by modern physics. One such anomaly that cannot be explained is the perihelion precession of Mercury's orbit being 43 arc-seconds per century. This unique case was explained through general relativity by Einstein.

One important aspect of Newton's law of gravitation is the factor $\frac{1}{r^2}$ used in closed elliptical orbits. When changing this factor, the associated orbit is altered as well. Instead of computing a space-time manifold to make the proper correction, a general relativistic error term is added to (1), resulting in (4).

$$F_G = \frac{GM_{Sun}M_{Mercury}}{r^2}\left[1 + \frac{3l^2}{r^2c^2}\right] \tag{4}$$

where M_{Sun} is the mass of Sun

where $M_{Mercury}$ is the mass of Mercury

$\boldsymbol{r}$ is the distance between Mercury and the Sun

$l = |\boldsymbol{r}\ x\ \boldsymbol{v}|$ is the orbital angular momentum per unit mass

c is the speed of light in vacuum

By using (4) and coupling it with (5) allows for easier location of Mercury's perihelion by using the arctangent of the cartesian coordinate frame to calculate the angle of perihelion.

$$\theta_p = \arctan\left(\frac{y_p}{x_p}\right) \tag{5}$$

2.3 Units

Being that this problem is heavily based on an abstract set of units it was decided to expand upon the classifications used in this project. All celestial body masses are expressed as a fraction of solar masses as seen in an example below:

$$M_{Sun} = 2\ x\ 10^{30}\ kg$$
$$M_{Earth} = 6\ x\ 10^{24}\ kg$$
$$M_{Earth,\ frac} = \frac{6\ x\ 10^{24}\ kg}{2\ x\ 10^{30}\ kg} = 3\ x\ 10^{-6}$$

All units related to time in the algorithm other than benchmarking will be in Earth years (yr). All units of distance will be in astronomical units (AU). This in turn identifies the velocity to be $\frac{AU}{yr}$ for this project.

By changing the variables units, the gravitational equations must be modified to accommodate altered velocities v and gravitational constant G. A simple example case can be seen below using the Sun and Earth as the celestial bodies.

$$F_G = \frac{G M_{Sun} M_{Earth}}{r^2} \tag{6}$$

Modification of (6) assuming Earth's orbit is circular:

$$F_G = \frac{M_{Earth} v^2}{r} \tag{7}$$

Combining (6) and (7) produces:

$$v^2 r = g M_{Sun} = 4\pi^2 AU^3 yr^{-2} \tag{8}$$

Knowing that $M_{Sun,\ frac} = 1$ with unit statement above and $r = 1\ AU$, (8) can be reduced into the set of derived units showcased in (9):

$$\begin{aligned} G &= 4\pi^2 AU^3 yr^{-2} \\ v_{Earth} &= 2\pi AU yr^{-1} \end{aligned} \tag{9}$$

2.3 Discretizing Differential Equations

To derive the foundation for the associated numerical methods used in the algorithm, the following Taylor expansions are formed.

$$\begin{aligned} x(t+h) &= x(t) + x'(t)h + \frac{x''(t)}{2}h^2 + (h^3) \\ &= x(t) + v(t)h + \frac{a(t)}{2}h^2 + (h^3) \end{aligned} \tag{10}$$

$$\begin{aligned} v(t+h) &= v(t) + v'(t)h + \frac{v''(t)}{2}h^2 + (h^3) \\ &= v(t) + a(t)h + \frac{a'(t)}{2}h^2 + (h^3) \end{aligned} \tag{11}$$

2.3.1 Forward Euler's

Using the leading terms of (10) and (11) coupled with making $dt = h$, (12) and (13) can be obtained:

$$\vec{x}(t+) = \vec{x}(t) + \vec{v}(t) \tag{12}$$

$$\vec{v}(t+) = \vec{v}(t) + \vec{a}(t) \tag{13}$$

The derived equations (12) and (13) can be discretized into (14) and (15):

$$x_{i+1} = x_i + v_i \tag{14}$$

$$v_{i+1} = v_i + a_i \tag{15}$$

2.3.2 Velocity Verlet

Using forward difference, $a'(t)$ can be approximated as:

$$a'(t) \approx \frac{a(t+h) - a(t)}{h} \tag{16}$$

By placing (16) into (11) and coupling with $dt = h$, (17) and (18) are produced:

$$\vec{x}(t+) = \vec{x}(t) + \vec{v}(t) + \frac{\vec{a}(t)^2}{2} \tag{17}$$

$$\vec{v}(t+) = \vec{v}(t) + \frac{\vec{a}(t) - \vec{a}(t+)}{2} \tag{18}$$

Due to the nature of the problem only dealing with gravitational forces, (15) can be used to find $a(t+)$ resulting in the ability to use (16) for solving.

The derived equations (17) and (18) can be discretized into (19) and (20):

$$x_{i+1} = x_i + v_i + \frac{{a_i}^2}{2} \tag{19}$$

$$v_{i+1} = v_i + \frac{a_i + a_{i+1}}{2} \tag{20}$$

3. Discussion of Algorithms

3.1 Program Structure

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 Velocity

3.2.1 Forward Euler

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
```

3.2.2 Velocity Verlet

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

3.2.3 Floating Point Operations Comparison

When calculating the floating-point operations per seconds (FLOPs):

Forward Euler requires $4n$ vector OPs. Considering that the vector length is two, this translates into $8n$ FLOPs. There is also a single call to the acceleration function. Velocity Verlet requires $6n$ vector OPs and $3n$ FLOPs. Taking the vector length of two into consideration for the OPs coupled with the extra $3n$ FLOPs, this translates into $15n$ FLOPs. There are also two calls to the acceleration function. By inspection it would then be expected that Velocity Verlet would roughly require twice the computational time of Forward Euler.

3.3 Acceleration

3.3.1 Classical

The implementation of Newton's classical acceleration can be seen below. It can be noted that for this project it was chosen that $z = 0$ thus causing the norm to be simplified into x and y components only when finding the magnitude of the distance.

Classical Acceleration Implementation

```
# Classical Acceleration
def Acc(self, Positions, Velocity, target, Masses ):
    x_acc = 0
    y_acc = 0
    for i in xrange(self.n_planets):
        if i != target:
            x_distance = Positions[target,0] - Positions[i,0]
            y_distance = Positions[target,1] - Positions[i,1]
            distance = math.sqrt( x_distance**2 + y_distance**2 )
            self.distance = distance
            x_acc -= G*Masses[i]*x_distance/distance**3
            y_acc -= G*Masses[i]*y_distance/distance**3
    return nmp.array([x_acc, y_acc])
```

3.3.2 Relativistic

The implementation of Einstein's relativistic acceleration can be seen below.

Relativistic Acceleration Implementation

```
# relativistic acceleration
def acc_rel(self, Positions, Velocity, target, Masses):
    x_acc = 0
    y_acc = 0
    for i in xrange(self.n_planets):
        if i != target:

            x_distance = Positions[target,0] - Positions[i,0]
            y_distance = Positions[target,1] - Positions[i,1]
            distance = math.sqrt( x_distance**2 + y_distance**2 )
            self.distance = distance
            l = Positions[target,0]*Velocity[1] \
                - Positions[target,1]*Velocity[0]
            rel_fac = 1 + ( (3*l**2) / (distance**2*c**2) )
            x_acc -= G*Masses[i]*x_distance/distance**3*rel_fac
            y_acc -= G*Masses[i]*y_distance/distance**3*rel_fac
    return nmp.array([x_acc, y_acc])
```

3.3.3 Modifications for Specific Case with β

To simulate a different gravitational situation, (6) was modified into (21) as seen below:

$$F_G = \frac{GM_{Sun}M_{Earth}}{r^2} \rightarrow F_G = \frac{GM_{Sun}M_{Earth}}{r^{\beta}} \quad with\ \beta \in [2,3] \tag{21}$$

By analyzing the various points held within the bounds, it is trivial to modify the classical acceleration function to account for the end points of the set.

Classical Acceleration Function Modified with β

```
# classical acceleration with modified beta in set [2,3] testing end
condition {3}
def acc_beta(self, Positions, Velocity, target, Masses ):
    x_acc = 0
    y_acc = 0
    for i in xrange(self.n_planets):
        if i != target:
            x_distance = Positions[target,0] - Positions[i,0]
            y_distance = Positions[target,1] - Positions[i,1]
            distance = math.sqrt( x_distance**2 + y_distance**2 )
            x_acc -= G*Masses[i]*x_distance/distance**4 # change
            y_acc -= G*Masses[i]*y_distance/distance**4 # change
    return nmp.array([x_acc, y_acc])
```

This change lowers the force and corresponding acceleration, causing an effect on the planets which will be examined in the results.

3.3 Preservation of Centre of Mass and Linear Momentum

The program makes the assumption that the first celestial body added to the solar system in the simulation is the Sun, and the associated initial condition of speed is zero. When the remaining celestial bodies are added, the Sun is adjusted with an argument condition so that the center of mass lies in the origin and the total linear momentum is zero. The planets axes were simplified into x and y components along the z plane. This should affect the simulation minimally due to the fact that the celestial body orbits are almost planar.

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

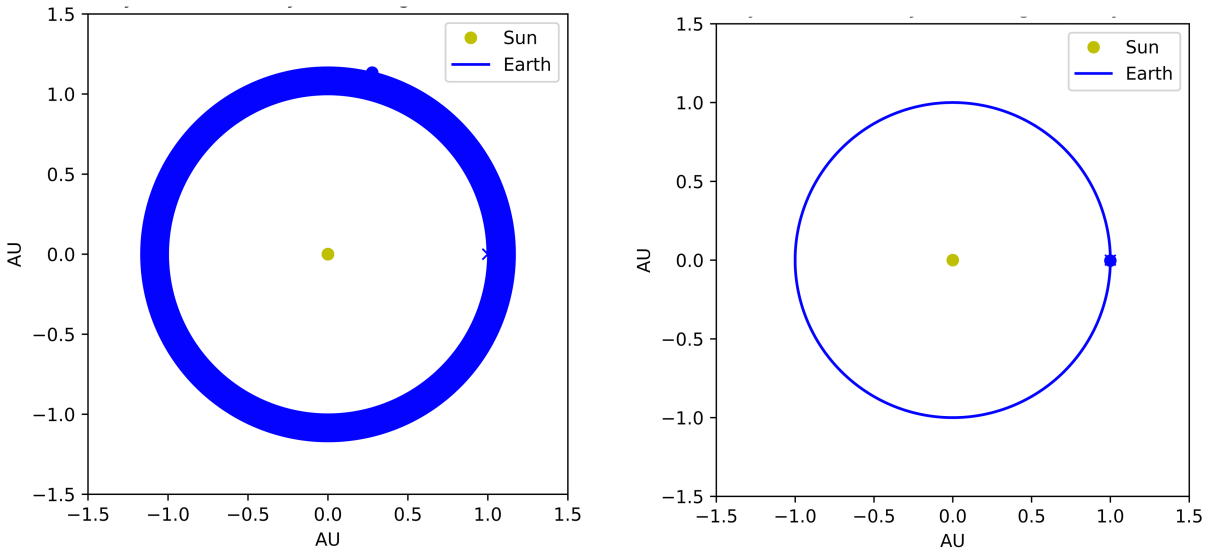
Table 1: System Specifications

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

4.2 Earth-Sun System

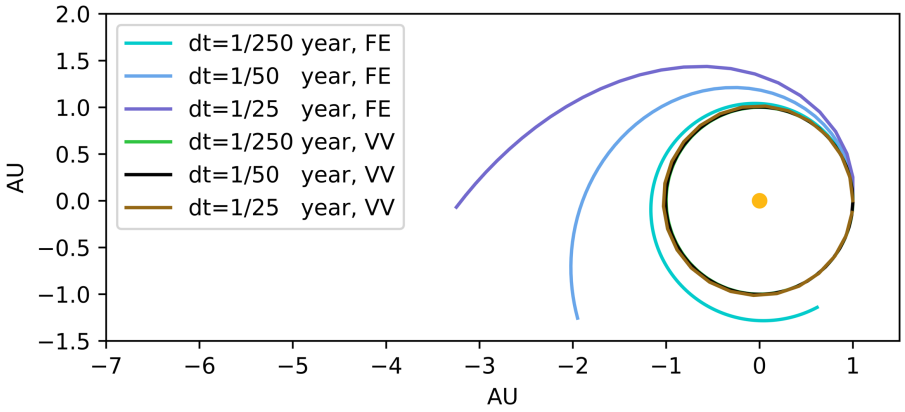
The first test of the code was to simulate a simple Earth-Sun system using the two integrators developed coupled with classical acceleration. To compare the accuracy of the Forward Euler and Velocity Verlet integrators were run over 50 years.

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

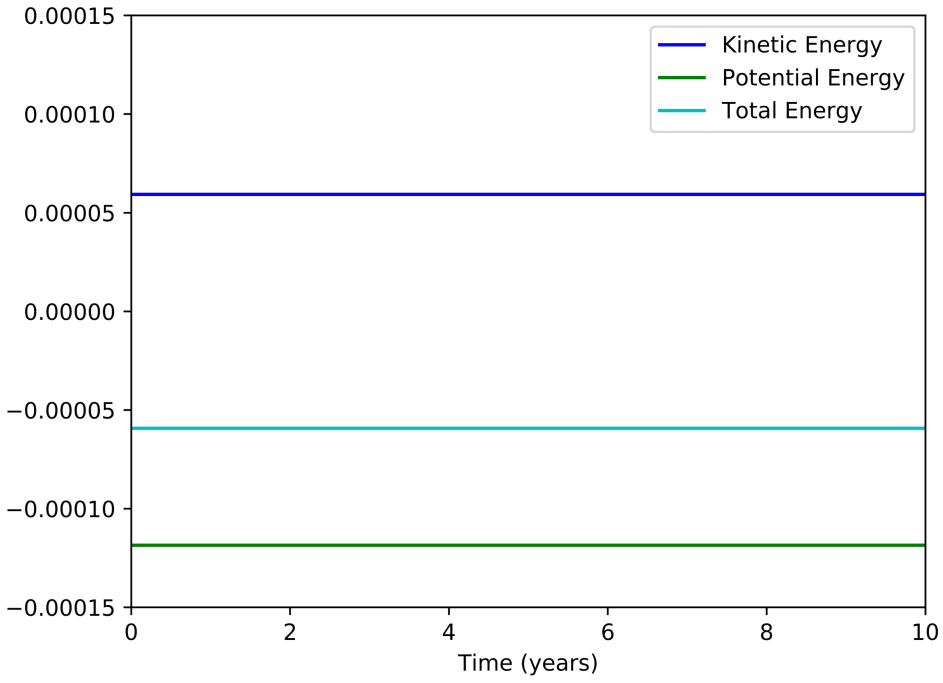
Table 2: Comparing Computational Time of the Forward Euler and Velocity Verlet Integrators using varying time steps (dt)

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 Velocity Verlet integrator.

4.3 Conservation of Energy and Angular Momentum

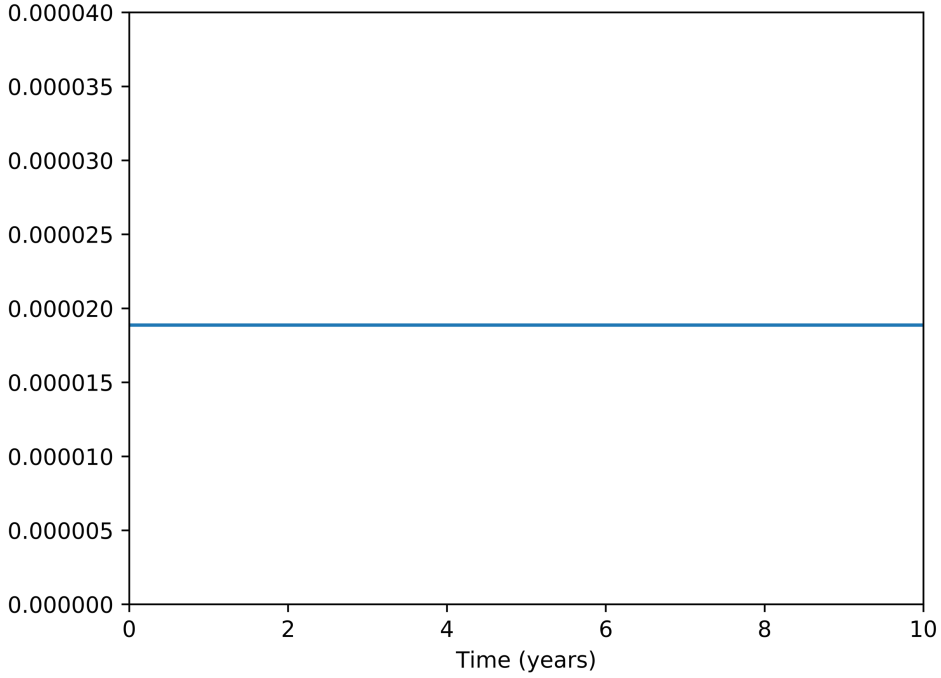
Since this simulation is isolated without external forces influencing the celestial bodies, it is expected that the total mechanical energy and angular momentum of the system is conserved throughout. Also with a circular orbit the associated kinetic and potential energy are expected to be conserved by 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]

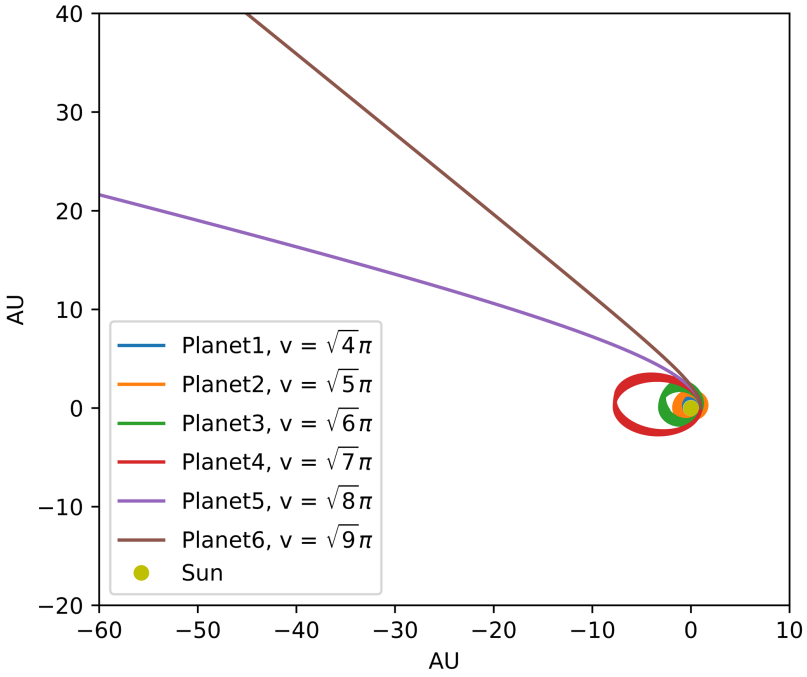
Table 3: Center of mass at start of simulation, then after 10 years

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

4.4 Escape Velocity

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.

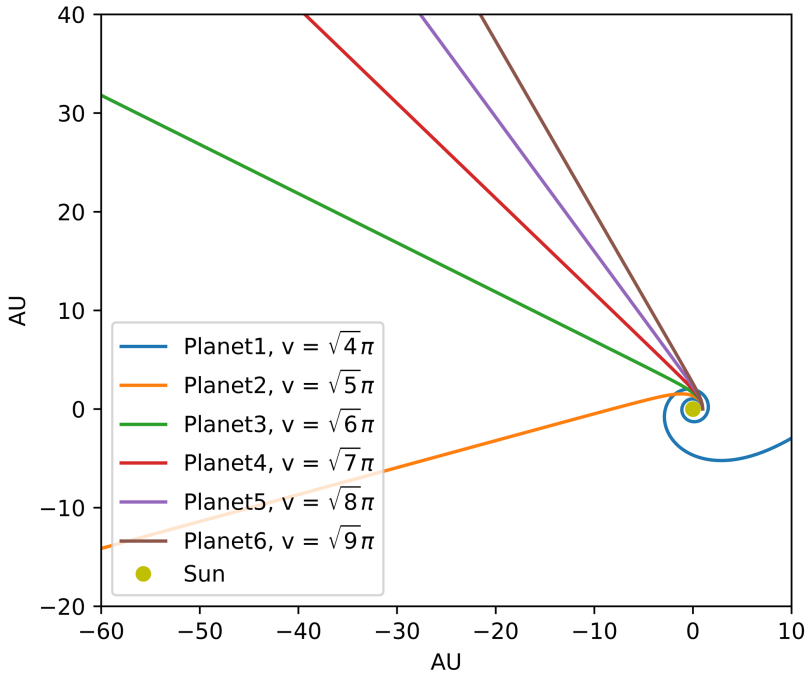
Figure 5: Varying Velocity (v) of Planet to Escape the Gravitational Effects of the Sun



It can be seen that when $v = \sqrt{8\pi}$, 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

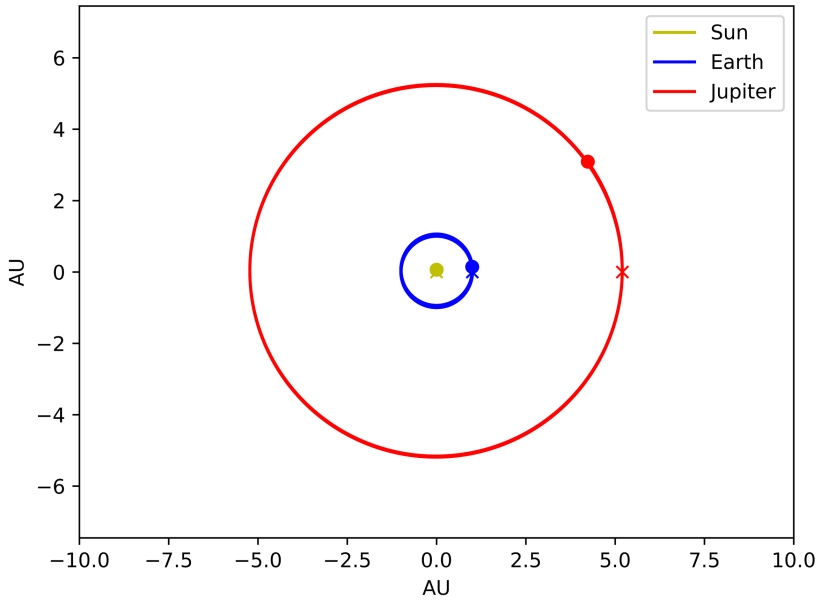


With the modified β term falling in the exponent on the lower side of the fraction, this means the associated gravitational force and acceleration are reduced. Through common logic it can be deduced that this would result in a weaker pull on the planet by the Sun. By using the same test velocities as *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 Three-Body Problem

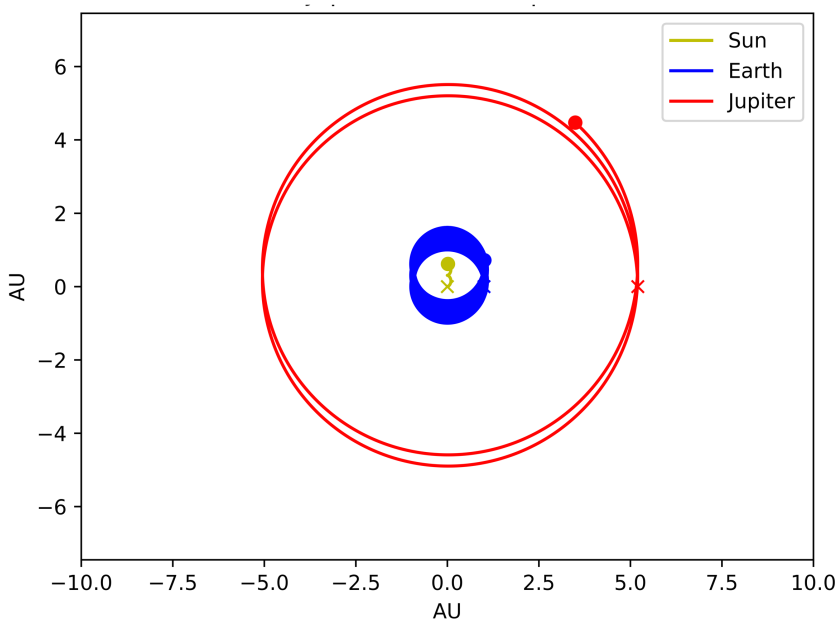
To continue towards a complete model of the solar system, the three-body problem is explored next. By including Jupiter into the Earth-Sun system, the simulation can be progressively tested. The automatic adjustment of the Sun was disabled for this simulation, as it depends on other celestial bodies 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



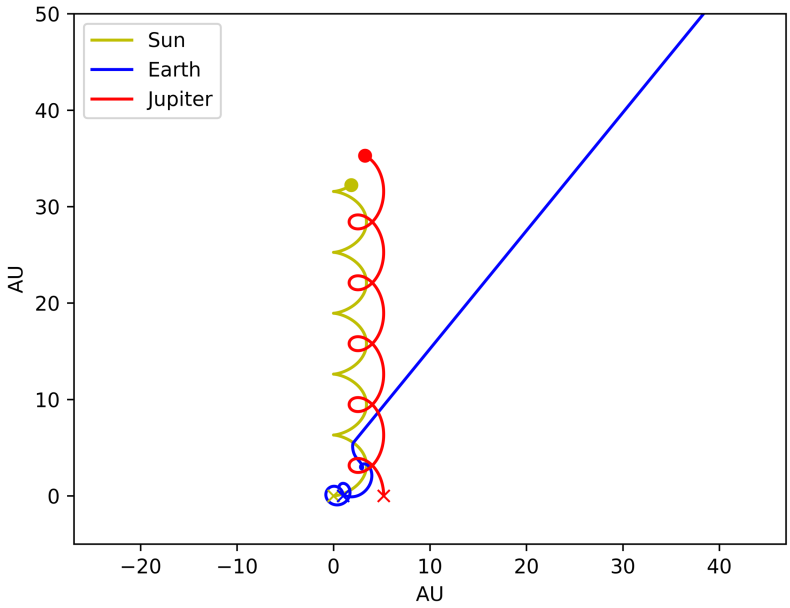
To visualize the effects of a larger mass affecting the orbits of other celestial bodies, Jupiter's mass was increased by a multiplier of 10 resulting in *Figure 8*.

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

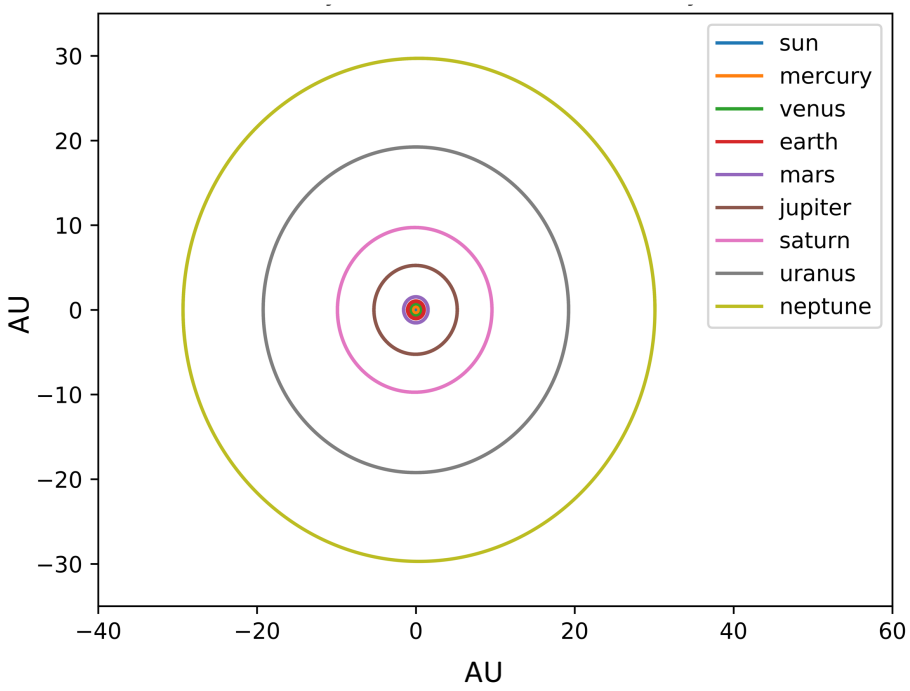


Since this multiplier places the mass of Jupiter around that of the Sun's, they actually start orbiting each other while Earth escapes eventually into a path free of outside influence. These tests show the factors at play when a celestial bodies' mass is altered. The observed orbits were as expected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

4.5 Complete Model of the Solar System

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



4.5 Perihelion Precession of Mercury

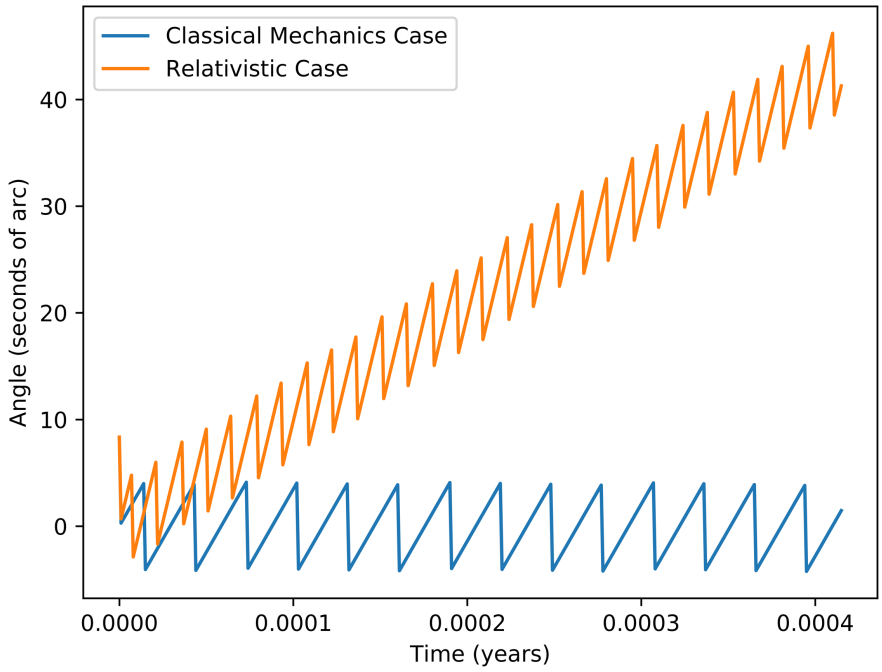
In a solar system with only the Sun and Mercury, there is a precession in the perihelion that cannot be explained by classical mechanics, but can be explained by the relativistic laws of gravity. The expected precession is 43 arc-seconds over the span of a century [1]. To validate this, two simulations were ran comparing classical and relativistic acceleration over a century. The results of this test can be seen in *Table 4* below.

Precession per 100 years (Classical)	1.45938
Precession per 100 years (Relativistic)	41.2757
Observed perihelions (Classical)	416
Observed perihelions (Relativistic):	416

Table 4: Results of Perihelion Precession of Mercury Simulation using dt = 1e-06 with Classical and Relativistic Acceleration over a Century

It is important to note an aspect of the counted perihelions, there is an initial blip in the data being counted as a perihelion, so that would mean there is actually only 415 perihelions. This however is minor and holding true to science with the fact that the tiny jump does exist, 416 perihelions were counted for the entirety of this project. The results *in Table 4* showcase that classical mechanics cannot appropriately estimate the precessions of mercury with the result of $1.45938 \neq 41$. Relativistic acceleration can however estimate with a very close estimate being produced of $41.2757 \approx 41$. There is a such a significant different between the two, that a plot was produced as seen in *Figure 11* to observe this visually.

Figure 11: Simulation of the Perihelion Precession of Mercury through Comparing the Classical Mechanics and Relativistic Cases with a dt = 1e-06



This helps clarify the faults in classical mechanics visually as seen by the differing slops of the curves. One important aspect to note about this algorithm is the very small step size needed for a successful simulation as it will not pick up the precessions. When initially running the program, a $dt = 1\ x\ 10^{-4}$ was the maximum amount of precision allowed with a reasonable computation time (over an hour). Through extensive optimization of the code a $dt = 1\ x\ 10^{-6}$ was successfully ran with a computation time of 2.6 hours. The results were worth the wait, as observing the data provides another part in the successful completion of this project.

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.

5. 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.

6. 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.