FYS4150 Project 3
Building a Model for the Solar System Using Ordinary Differential Equations

Tiffany Chamandy and Andri Spilker

October 24, 2016

Contents

Abstract

In this project a model of our Solar system is built by solving a set of coupled differential equations for the time evolution of the positions and velocities of the planets and Pluto. This is done using the Euler and Verlet methods for solving ordinary differential equations numerically. From a comparison of the methods, we find that the Verlet method conserves energy and angular momentum and is the most accurate. Three different systems are examined; the first with just the Earth and Sun, then again with Jupiter included, and finally with all planets in the Solar system adding Pluto for historical reasons. This is done in order to understand how the masses of the planets affect each other and how the Solar system works. In the end the Newtonian force of gravity is modified to include the effects of general relativity, and we confirm that the perihelion precession of Mercury requires a general relativistic gravity.

1 Introduction

In this project, we build a model for the Solar system using coupled ordinary differential equations. The aim of the project is to familiarise ourselves with classes and object orientation, as well as the Euler and Verlet method for solving coupled differential equations numerically. With these tools, we build a model of the solar system, and can investigate the stability of the system with time. For stability, conservation of energy and angular momentum are needed, and this project tests the ability of the Euler and Verlet methods to produce the Solar system as we know it.

As we know from Newtonian gravity, the force on two objects in orbit is given by the equation:

$$F_G = \frac{GM_{Sun}M_{Earth}}{r^2} \tag{1}$$

Where G is the gravitational constant, M_{Sun} is the mass of the sun, M_{Earth} is the mass of the earth and r is the distance between them. This formula can be used to find the force of gravity between any two celestial bodies. Using this equation for gravitational force, and knowing that the second derivative for position is acceleration; we can find the gravitational acceleration in the x and y directions. These can be written in terms of second order differential equations.

$$\frac{d^2x}{dt^2} = \frac{F_{G,x}}{M_{Earth}} \qquad \frac{d^2y}{dt^2} = \frac{F_{G,y}}{M_{Earth}} \tag{2}$$

Where $F_{G,x}$ and $F_{G,y}$ are the force components in the x and y planes. Since the Earth is in orbit around the Sun, and we are assuming a fixed Sun in the first Earth-Sun system, we can extrapolate that the first of the circular rotation of the Earth is equal to the centripetal force. From this we can find the circular velocity, or orbital velocity. The centripetal force is given as the following relation. Solving for the velocity in the left hand side of equation (3) we get the circular velocity.

$$F_G = \frac{v^2 M_{Earth}}{r} = \frac{GM_{sun}M_{Earth}}{r^2} \tag{3}$$

$$v^2 r = GM_{sun} = 4\pi^2 AU^3/yr^2 \tag{4}$$

These equations will become very useful in this project when solving the differential equations using Euler's method and the velocity Verlet method [4]. We use Astronomical units for distance, Solar masses for mass, and years for time, and from equation (4) we see that G in these units is $4\pi^2$.

Euler's method of forward integration is a way of solving a discretised differential equation. A typical second order differential equation of this form looks like equations (2)- $m\frac{d^2x}{dt^2} = F(x,t)$, which can be written in terms of two coupled differential equations for position and velocity. The equations are said to be coupled because they depend on each other, and need to be calculated simultaneously.

$$\frac{dx}{dt} = v(x,t) \qquad \frac{dv}{dt} = \frac{F(x,t)}{m} = a(x,t) \tag{5}$$

The Verlet Method states that if we now perform a Taylor expansion, we find discretised equations that we can solve. First, we perform a Taylor expansion for x(t+h). We denote nth order derivatives by $^{(n)}$, and hence the second term for this Taylor expansion, $x^{(2)}(t)$ can be substituted for the second order differential in equation 5, $\mathrm{x}^{(2)}(t) = a(x,t)$.

$$x(t+h) = x(t) + hx^{(1)}(t) + \frac{h^2}{2}x^{(2)}(t) + O(h^3) \tag{6}$$

$$x(t \pm h) = 2[x_i - x_{i-1}] + h^2 x_i^{(2)} + +O(h^4) \tag{7}$$

Using equation 6 and adding the $x(t-h)$ portion, we get equation (7). This is formally solved for the Taylor expansions for both positions and velocity using these equations as a base. The Taylor expansion for position and velocity are given by

$$x_{i+1} = x_i + hx_i^{(1)} + \frac{h^2}{2}x_i^{(2)} + O(h^3) \tag{8}$$

$$v_{i+1} = v_i + hv_i^{(1)} + \frac{h^2}{2}v_i^{(2)} + O(h^3). \tag{9}$$

The equation for velocity can be broken down further by stating that $v_{i+1}^{(1)} = v_i^{(1)} + hv_i^{(1)} + O(h^2)$ and solving for $hv_i^{(2)} \approx v_{i+1}^{(1)} - v_i$. We substitute this into the expansion for velocity above and get the finalised equation:

$$v_{i+1} = v_i + \frac{h^2}{2}(v_{i+1}^{(1)} + v_i^{(1)}) + O(h^3). \tag{10}$$

Solving equation (8) and (10) give solutions for position and velocity using Verlet's Method. Euler's forward method is very similar [4]. Using the coupled equations above, we show that $\frac{dx}{dt} = v(x,t)$ and $\frac{dv}{dt} = \frac{F(x,t)}{m} = a(x,t)$. We may re write the second equation as $y'(t_i) = F(t_i, y_i)$, where $t_{i+1} = t_i + h$ and

$$y_{i+1} = y(t_i) + hF(y_i, t_i) + O(h^2). \tag{11}$$

We want to write this equation in terms of position and velocity as follows with $y_1^{(0)}$ being the position and $y_1^{(1)}$ being the velocity.

$$y_1^{(0)} = y_0 + hv_0 + O(h^2) \tag{12}$$

$$y_1^{(1)} = v_0 - \frac{hy_0k}{m} + O(h^2) \tag{13}$$

Re writing these formulas in terms of a step size N, we get the final solutions to Euler's method to be as follows (with a_n is the acceleration).

$$y_{n+1}^{(0)} = y_n^{(0)} + hy_{n+1}^{(1)} + O(h^2) \tag{14}$$

$$y_{n+1}^{(1)} = y_n^{(1)} + ha_n + O(h^2) \tag{15}$$

Using the NASA site provided, **http://ssd.jpl.nasa.gov/horizons.cgi**, we found the mass, in Solar masses (1 $M_{sun} = 2x10^{30}$ kg), and distances in AU (astronomical units, 1AU= $1.5x10^{11}$ m) for each of the 8 planets and one dwarf planet, Pluto. Throughout the report we use units of AU for distance, years for time and Solar masses for mass. For the Earth-Sun system and the Jupiter-Earth-Sun system, we used the values displayed in the below table, while for the remainder of the project the data from Nasa were used. We obtained values for x, y and z positions and velocities from the Nasa website at 14.October 2016 at 00.00.0000.

Planet	Mass(M_{sun})	Distance(AU)	Velocity (AU/Yr)
Earth	3e-06	1AU	2π
Jupiter	9.5e-04	5.20 AU	2.75425
Mars	3.3e-07	1.52 AU	5.07782
Venus	2.45e-06	0.72 AU	7.35357
Saturn	2.75e-04	9.54 AU	2.03554
Mercury	1.2e-07	0.39 AU	10.1743
Uranus	4.4e-05	19.19 AU	1.43512
Neptun	5.15e-05	30.06 AU	1.14613
Pluto	6.55e-09	39.53 AU	0.995132

2 Method and Results

All C++ program files and Python plotting programs as well as resultant images and plots can be found at **https://github.com/akspilke/PROJECT3-Solar-System**.

2.1 The Euler Method

In the introduction we found that the necessary equations for position and velocity for the Euler method were:

$$x_{i+1} = x_i + hv_i + O(h^2) \tag{16}$$

$$v_{i+1} = v_i + ha_i + O(h^2). \tag{17}$$

Higher order terms were ignored, and these equations were implemented into our code as shown below, where the timestep h is written as dt, and $a = \frac{F}{m}$ from Newton's $F = ma$.

```
#include "euler.h"
#include "solarsystem.h"

Euler::Euler(double dt) :
    m_dt(dt)
{
}

void Euler::integrateOneStep(SolarSystem &system)
{
    system.calculateForcesAndEnergy();

    for(CelestialBody &body : system.bodies()) {
        body.position += body.velocity*m_dt;
        body.velocity += body.force / body.mass * m_dt;
    }
}
```

Figure 1: Code for Solving Euler's Forward Method

2.2 The Verlet Method

The equations necessary for solving the coupled ordinary differential equations with the Verlet method were found to be:

$$x_{i+1} = x_i + hv_i + \frac{h^2}{2}a_i + O(h^3) \tag{18}$$

$$v_{i+1} = v_i + \frac{h}{2}(a_{i+1} + a_i) + O(h^3). \tag{19}$$

The factor a_{i+1} was not entirely trivial to implement into the code. Therefore we needed an intermediate step to solve for this factor. We did this by calculating the velocity at $i + \frac{1}{2}$ (for half time steps).

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

$$x_{i+1} = x_i + hv_{i+\frac{1}{2}} \tag{21}$$

The forces of the system then needed to be calculated again before we calculated a_{i+1}, because the acceleration and force are coupled. This step is shown as:

$$v_{i+\frac{1}{2}+\frac{1}{2}} = v_{i+1} = v_{i+\frac{1}{2}} + \frac{h}{2}a_{i+1}. \tag{22}$$

We have included the Verlet Class from our c++ script below. This class is similar in structure to the Euler class and solves for the position and velocity using the methods just described, see figure 2.

```
#include "verlet.h"
#include "solarsystem.h"

Verlet::Verlet(double dt) :
    m_dt(dt)
{

}

void Verlet::integrateOneStep(SolarSystem &system)
{
    system.calculateForcesAndEnergy();

    for(CelestialBody &body : system.bodies()) {
        body.velocity += (m_dt/2)*(body.force / body.mass); //Calculate velocity at half step
        body.position += body.velocity*m_dt;
    }
    system.calculateForcesAndEnergy();

    for(CelestialBody &body : system.bodies()) {
        body.velocity += (m_dt/2)*(body.force / body.mass);
    }
}
```

Figure 2: Code for the Verlet method.

2.3 Object Oriented Code

Object orientation is very useful when the program contains a lot of variables and vectors, as is the case for this project. Object orientation is used to organise classes of variables and vectors belonging to the same functions, and these classes can be understood as "groups" of variables and vectors that have similar properties or definitions. These classes then make it possible to define multiple variables in a similar fashion at the same time, and removes unneccessary

definitions and calculations from the main file. This makes the program more efficient and often also more readable.

The program for this project is based on a program written by Anders Hafreager, which was found on **https://github.com/andeplane/solar-system**. We used this object oriented code to solve for the Euler's method as described above. The code consists of header files and cpp files for the different classes, where the header files are used to define all the variables needed to solve the equations in the cpp files.

In the vec3 class positions, velocities and forces of the solar system are defined as three vectors in x, y and z coordinates. Vector operations are then also defined, such that component wise vector addition, subtraction, multiplication and division of vectors can be done easily throughout the program.

The Celestial Bodies class defines which vectors and variables are associated with the different celestial bodies in the system. This connects the positions, velocities, forces and masses to the different celestial bodies as the program runs through the bodies in the system. Celestial Bodies therefore consists of three variables as vectors, as well as doubles for the mass.

The Solar System class then uses the vectors and variables defined in vec3 and tied to a celestial body in Celestial Bodies, to calculate the forces, potential energies, kinetic energies and angular momenta of the bodies in the system. The forces are needed to find the positions of the celestial bodies in the next time step, because they determine the orbit of the celestial bodies. The potential and kinetic energies and angular momenta are included in order to investigate the conservation of these quantities, which are necessary for stable systems.

The Euler class defines the Euler integration method and implements the necessary equations from the introduction, as seen in the code from the previous section. A new class file for the Verlet Method was then created in a similar fashion, in order to investigate the differences between the two algorithms and improve our results. In the main file, the solar system is created using the Solar System class, celestial bodies are added using the Celestial Body class, and integrated over all timesteps using either the Euler method or the Verlet method, producing files of the positions and velocities of the different planets as the solar system evolves with time.

2.4 The Earth Sun System

Using the Euler and Verlet method, we can solve for the velocity of an object in orbit by solving the discretised differential equations. Our solar system program, explained above, makes a model for the Earth-Sun system. With Earth at an x distance of 1 AU and a y distance of 0 AU. The code works for time steps of h $h = (t_{final} - t_0)/N$, where t_{final} the final time and the initial time t_0 have to be specified. In the below plots we ran the program for one year, with timesteps of 0.001 (1000 timesteps). The velocity of the Earth was calculated assuming a perfectly circular orbit, with the formula

$$v_{orbit} = \frac{CircumferenceCircle}{Periodoforbit} = \frac{2\pi r}{Period}. \tag{23}$$

For Earth at $r = 1AU$ and period=1year, this gives an initial velocity of 2π in our units. The Euler Method was inefficient at plotting and did not complete the circular orbit because energy was not conserved in that case. On the contrary, the Verlet method executed these positions perfectly and we were left with a perfect circular orbit, showing that this method was more effective and conserved energy in the system. Our results for these plots can be seen in figures 3 and 4.

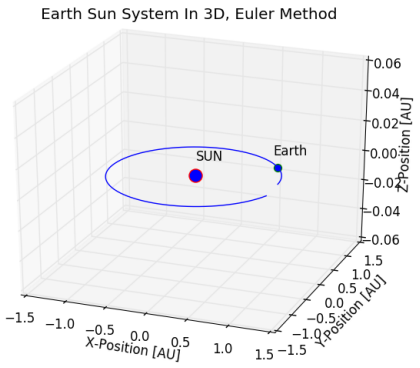

Figure 3: A three dimensional plot of the Earth-Sun system solved with the Euler algorithm.

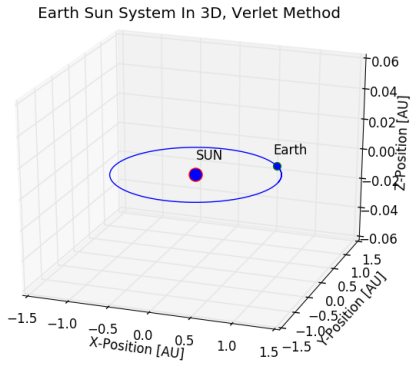

Figure 4: A three dimensional plot of the Earth-Sun system solved with the Verlet algorithm.

2.5 Test of the Algorithm

We now investigate the stability of the Earth's orbit around the Sun with the two different methods, by plotting the Earth- Sun system in 3D. In a text file, the positions along the orbit of Earth were saved, with time-steps of Δt and the initial conditions were specified as above. The results and specified timesteps can be seen in figures 5, 6, 7, 8 and 9.

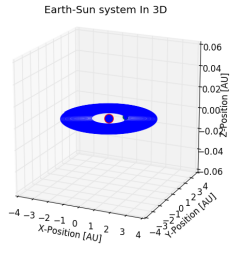

Figure 5: Euler method for Earth, 10^5 timesteps of size 10^{-3}, hundred years.

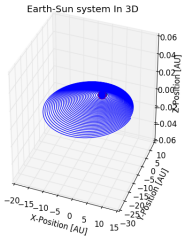

Figure 6: Euler method for Earth, 10^5 timesteps of size 10^{-2}, thousand years.

From these figures we see that the Verlet method clearly gives much better results than the Euler method. We also compared the number of floating point operations (flops) and the timings of the two methods. The Euler method contains 5 flops and the CalculateForcesandEnergy function, and so in total contains 19 flops per timestep. The Verlet method contains 8 flops and two CalculateForcesandEnergy, giving a total of 36 flops per timestep. Timing the two functions we found that for 10^5 timesteps the Euler method uses on average 0.298638 seconds, while the Verlet method uses 0.320229 seconds. Hence the time difference is not very large even though the number of floating points are quite different, and the Verlet method gives much more stable results for fewer timesteps, and we will therefore use the Verlet method for the remainder of the project. It should be noted that these measurements were taken while writing to file, and so the difference is probably larger than what these values indicate.

We also tested our code by seeing if our system satisfies conservation of energy. As we know the total energy of a system is $E_{tot} = K + U$ where K is the kinetic energy $K = \frac{1}{2} M_E V^2$ and U is the potential energy $U = -\frac{GM_S M_E}{R}$. The code for this is displayed in figure 10.

We solved for the kinetic and potential energies using the velocity we found with the two methods. We found that while the kinetic energy remained constant for each time-step, the potential energy was not conserved in the Euler method, while in the Verlet method both are conserved. This explains figures 5, 6, 7, 8 and 9 which showed instability for the Euler method for large timesteps.

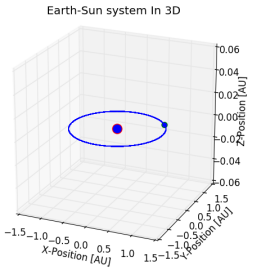

Figure 7: Verlet method for Earth, 10^5 timesteps of size 10^{-3}.

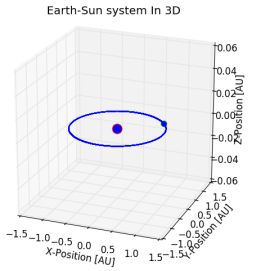

Figure 8: Verlet method for Earth, 10^5 timesteps of size 10^{-2}.

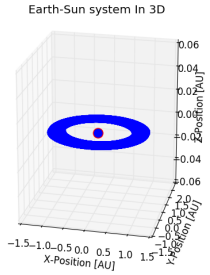

Figure 9: Verlet method for Earth, 10^5 timesteps of size 10^{-1}.

```
    m_angularMomentum+=body1.mass*body1.velocity.lengthSquared()*dr ;

    m_potentialEnergy = -(m_G*body1.mass*body2.mass)/dr;
}
m_kineticEnergy += 0.5*body1.mass*body1.velocity.lengthSquared();
```

Figure 10: Code for Kinetic and Potential Energy and Angular Momentum

By the laws of physics, angular momentum is conserved in an orbit. By Kepler's Laws, an object in orbit must cover the same area in the same time. Angular momentum is defined as $L = MVr$, which we solved in our code. We found that the values of the angular momentum varied, and it is not conserved for the Euler method, but is constant for Verlet. We concluded that Euler's method is not accurate enough and has a significant error unless the timesteps are very small.

2.6 Finding the Escape Velocity

When the velocity of a body in orbit increases, the orbit grows until the body can escape the system all together; going from a circular orbit all the way to hyperbolic, where it is not bound to its orbit anymore. The equation for escape velocity is shown below, from [5].

$$v_{esc} = \sqrt{\frac{2GM_E}{R}} \tag{24}$$

Though trial and error with our code and increasing the initial velocity of the Earth, we were able to find the velocity which is needed for an Earth mass planet at 1 AU to escape the Sun's gravity. We found that a velocity of 3π satisfies this condition, and makes the Earth escape from the Solar system.

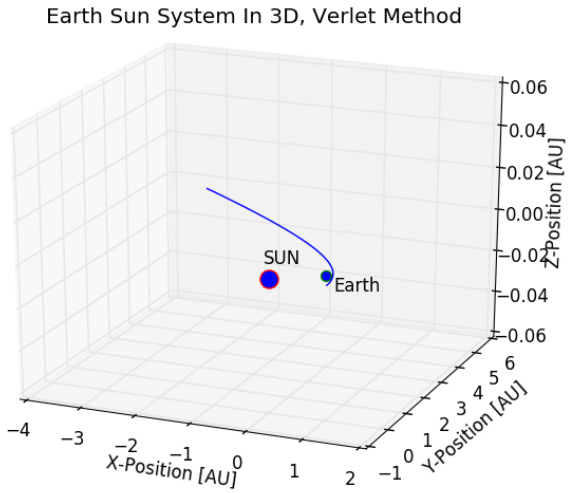

Figure 11: Earth-Sun System with Verlet method, and Earth with initial velocity of 3π (the escape velocity).

From figure 11, we see that Earth's orbit is hyperbolic, making the Earth escape the Sun's gravitational influence. It is interesting to see that only a small change in the initial velocity of the Earth, completely alters the conditions on our planet.

2.7 Three Body Problem; Adding Jupiter

We now add the most massive planet of the Solar system to our Earth-Sun system, to see how a significant third mass affects the system. Here we have still assumed circular orbits, even though this is not strictly true for Jupiter. The orbit velocities were calculated with

$$v_{orbit} = \frac{CircumferenceCircle}{Periodoforbit} = \frac{2\pi r}{Period}. \tag{25}$$

For the Earth we see that in our units, this simply gives $v = \frac{2\pi * 1AU}{1year} = 2\pi AU/year$. The orbit of Jupiter was added in a similar fashion to the Earth, with the distance between the Sun and Jupiter as the position x coordinate (5.2 AU), and the y and z coordinates as zero. The velocity calculated with the above equation was then set as the velocity in the y direction, while the other coordinates were set to zero. We then ran our Verlet code with these initial conditions and found the position elements for this three body problem. The result can be seen in figure 12.

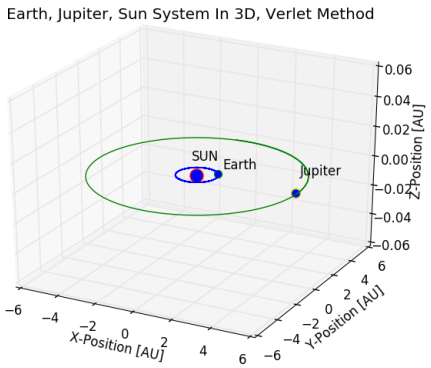

Figure 12: Earth-Jupiter-Sun System with Verlet method.

From examining the positions and velocities of the objects in the system we can see that the system has changed. The Earth's orbit appears more elliptical due to the pulling of Jupiter's gravity with the Sun's gravity. To see this more clearly, we increase the mass of Jupiter. The mass was first increased by a factor of 10, see figure 13, and then by a factor of 1000 (figure 14). For a mass 10 times the real mass of Jupiter, we see that the orbits are altered. When Earth is at perihelion with Jupiter, both orbits are pulled toward each other due to the force of gravity between them. At this point, the system is becoming unstable due to the constant displacement of both orbits. For a thousand times the real mass of Jupiter, we see that the system is completely unstable. When Earth

orbits around the Sun and it is at perihelion with Jupiter, the orbit becomes completely distorted and the Earth makes a series of orbital encounters with the now Sun sized Jupiter until it is sling shotted out of the Solar system via a strange Hohmann transfer [1]. This strange encounter that causes the Earth to be shot out of the Solar system causes Jupiter to spiral out as well until it is also shot out of our solar system.

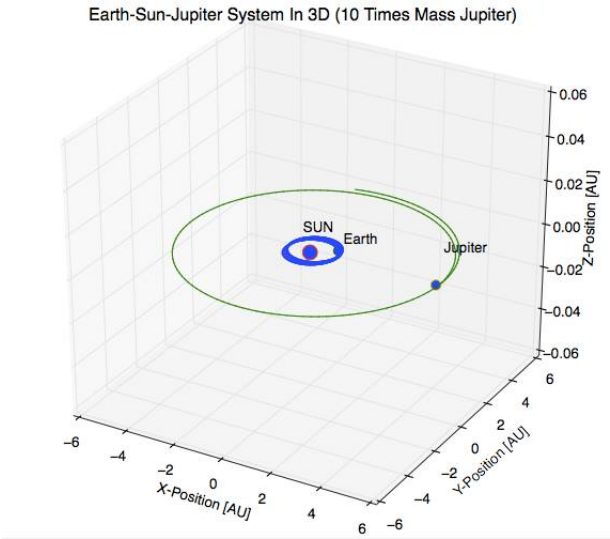

Figure 13: Earth-Jupiter-Sun System with Verlet method, 15 000 timesteps.

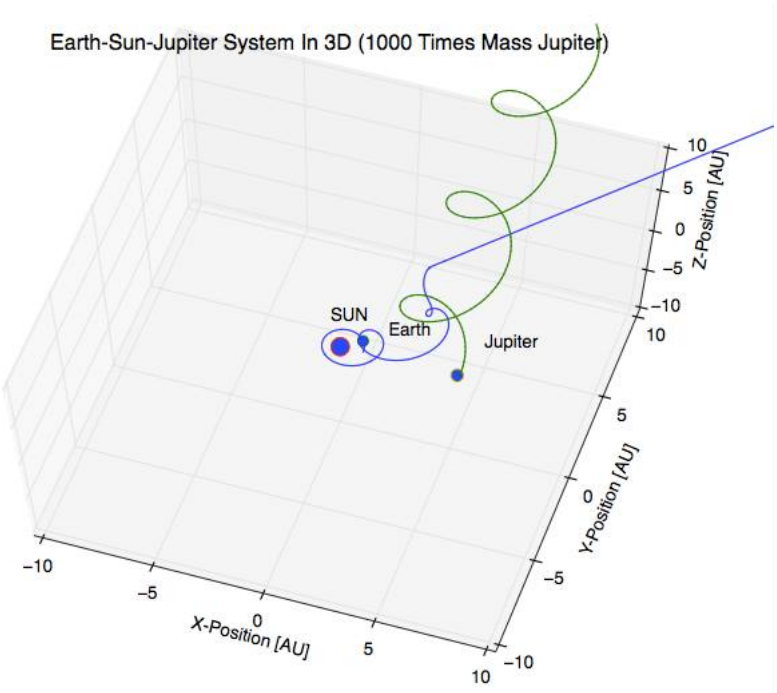

Figure 14: Earth-Jupiter-Sun System with Verlet method, 15 000 timesteps.

2.8 Final Model for all Planets of the Solar System

From the previous section, we have seen that the mass of Jupiter affects the Solar system, and in order to solve for the whole Solar system with all planets, we therefore change the centre of mass of the system to take account of the mass of Jupiter. The formula for the centre of mass between two bodies is as follows:

$$x_{COM} = \frac{m_1 x_1 + m_2 x_2}{m_1 + m_2}. \tag{26}$$

We insert this formula into our object oriented program, so the centre of mass is calculated in x, y and z coordinates, for all pairs of bodies in the system. We then move the origin of our coordinate system to be at this point rather than in the centre of the now slightly circling Sun. This is done by subtracting the total momentum of the system from the velocity of the Sun, in order to make the total momentum of the system zero and hence fix the centre of mass. We then used the Nasa data for all celestial bodies and plotted their orbits with respect to the centre of mass, as seen in figures 16 and 17. The orbits of Netune and Pluto could not be completed due to the to their long orbital periods and limitations in the size of the file allowed in Python notebook.

It was cool to see our own Solar system in 3D and how the planets affect each other about their centre of mass.

```
void SolarSystem::removeTotalMomentum()
{
    vec3 P;
    P.zeros();

    for(int i=1; i<numberOfBodies(); i++) {
        CelestialBody &body = m_bodies[i];

        P += body.mass*body.velocity;
    }
    CelestialBody &sun = m_bodies[0];
    sun.velocity -= P/totalmass;
}
```

Figure 15: Code for calculating and removing total momentum of the system.

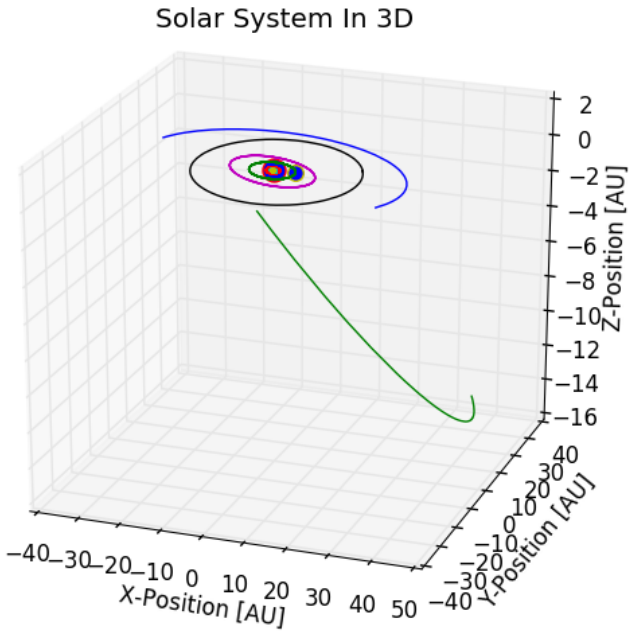

Figure 16: Solar System with all planets and Pluto.

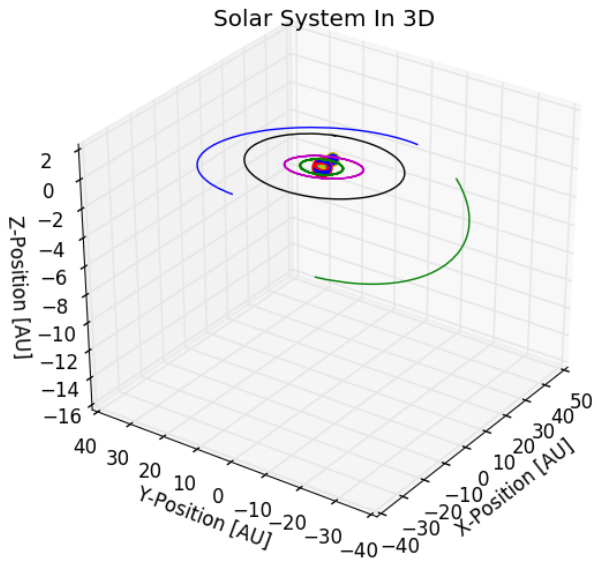

Figure 17: Solar System with all planets and Pluto.

2.9 The Perihelion Precession of Mercury

We first modified the force equation to include the relativistic correction term $[1 + \frac{3l^2}{r^2c^2}]$ to the original force equation, so it becomes:

$$F_G = \frac{GM_{Sun}M_{Jupiter}}{r^2}[1 + \frac{3l^2}{r^2c^2}]. \tag{27}$$

Where l is the magnitude of the orbital angular momentum of Mercury per unit mass ($l = |\vec{r}x\vec{v}|$). We then solved for the orbital angle that Mercury makes it ints orbit with respect to the x and y position elements $\theta = \arctan \frac{x_p}{y_p}$. In order to do this we found the position elements where Mercury is in perihelion with the Sun (meaning at its closest distance). We found these points by integrating over all points and looking at the distance between the Sun and Mercury, then compare with the previous point and if the current distance is smaller than the previous distance we count that point and proceed until we find the closest point to the Sun (where r has the smallest value). The perihelion distances and coordinates were found, and the perihelion angle calculated for each year in a period of 10 years. The code for this problem is shown below.

The perihelion angle (thetaprev) and the time of this measurement (time) were saved in a text file to plot the correlations in Python, as seen in figure 19.

```
// PERIHELION OF MERCURY
CelestialBody &mercury = solarSystem.createCelestialBody(vec3(0.3075, 0, 0), vec3(0, 12.44, 0), 1.65e-7, string("Mercury") );

ofstream myfile;
string filename = "PerihelionMercury10YR10.txt";
if(!myfile.is_open()) {
    myfile.open(filename.c_str(), ios_base::out);
}
vector<CelestialBody> &bodies = solarSystem.bodies();

double rOld = 0;
double rPrev = 0;
double r = 0;
double thetaPrev = 0;

Verlet integrator(dt);
for(long int time = 0; time < numTimesteps; time++){
    integrator.integrateOneStep(solarSystem);

    double x = bodies[1].position.x() - bodies[0].position.x();
    double y = bodies[1].position.y() - bodies[0].position.y();
    double thetaCurrent = atan2(y, x);
    double rCurrent = (bodies[1].position - bodies[0].position).length();

    if(rCurrent > rPrev && rPrev < rOld){
        myfile << thetaPrev << endl;
        myfile << time << endl;
    }
    rOld = rPrev;
    rPrev = rCurrent;
    thetaPrev = thetaCurrent;
 }
```

Figure 18: Code for calculating the precession of the perihelion angle of Mercury.

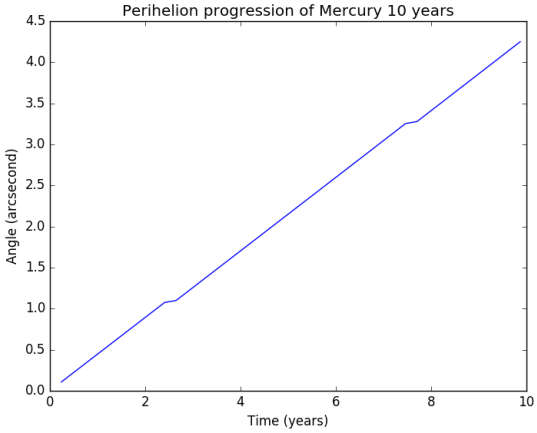

Figure 19: Angle vs Time for Perihelion of Mercury

Figure 19 shows the angle difference for 10 years with time steps of $1x10^{-8}$. It can be seen that there is fluctuations within the angle difference at various

perihelion positions. Such as at 2.4 years and 8 years. This indicates changes in the angle of Mercury at perihelion due to the effects of general relativity. We expect these fluctuations to occur many more times in a 100 year time span. From our model, the perihelion angle difference is 4.25 arcseconds, which projected to 100 years would give 42.5 arcseconds, which is close to the observed perihelion precession of 43 arcseconds over 100 years. We attempted to run our code for more years and smaller timesteps, but our program took more than three days to run (for timesteps of $1e-10$ and 20 years) and then crashed-making it impossible. The fact that the results indicate an angle very close to 43 arcseconds for 100 years is a confirmation of General Relativity, showing that the observed angle of the perihelion precession of Mercury can not be explained by classical Newtonian dynamics, we need General Relativity! This has also been confirmed by numerous observations to a high degree of accuracy [2], [3], [6]. And so Einstein's crazy theory that space and time are curved by gravity is true ♡

3 Analysis and Conclusion

We conclude with that many body problems are much better solved numerically than analytically, and that discretisation of differential equations and application of methods such as the Euler and Verlet methods are extremely useful for such systems. We found that the Verlet method was more efficient than the Euler method, in spite of having a higher order error and more floating point operations. This was due to the higher order Taylor expansion term included. We also found that by altering the masses or the velocities of the planets in the Solar system, the system could completely change, making the conditions on Earth and our lives quite different. In the final section of the project the effects of general relativity were included, and it was shown that general relativity is required to explain some of the features observed in our Universe in detail, such as the perihelion precession of Mercury.

In future projects, more integration methods could be investigated, in order to make more efficient programs. More of the program could probably also be written in classes, and we could have specified forces and methods and timesteps in the command line rather than commenting and uncommenting in the file. Nasa data could have been used for a larger part of the project, to make the systems more similar to reality, and more timesteps could have been added to the perihelion of Mercury calculation to investigate the dependancy on general relativity more closely.

Through completing this project we have learnt how to make use of classes in C++, how to treat large text files and how not to give up when a program has been running for a whole weekend just to crash a few minutes before completion. It has been interesting and fun to work with the Solar system, and to see how small changes are able to completely alter the conditions of our planet. It was also informative to see general relativity at work, and how this determines the movement of the planets.

References

[1] Howard Curtis. *Orbital mechanics for engineering students*. Butterworth-Heinemann, 2013.

[2] Frank W Dyson, Arthur S Eddington, and Charles Davidson. A determination of the deflection of light by the sun's gravitational field, from observations made at the total eclipse of may 29, 1919. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 220(571-581):291–333, 1920.

[3] Albert Einstein. Erklarung der perihelionbewegung der merkur aus der allgemeinen relativitatstheorie. *Sitzungsber. preuss. Akad. Wiss., vol. 47, No. 2, pp. 831-839, 1915*, 47:831–839, 1915.

[4] Merlin L James, Gerald M Smith, and JC Wolford. *Applied numerical methods for digital computation*, volume 2. Harper & Row New York, 1985.

[5] Emmanuel Kanambaye. Concept of smallest length having a physical sense according to the general relativity especially in schwarzschild metrics. 2015.

[6] Karl Schwarzschild. On the gravitational field of a mass point according to einstein's theory. *arXiv preprint physics/9905030*, 1999.