FYS4150 Project 3

Alex Ho
Lars Frogner

October 31, 2016

Abstract

We consider the equations of motion governing a gravitational many-body system, and discuss numerical methods for solving these. The methods are applied to simulate the motion of solar system bodies. We also add a general relativistic correction in order to reproduce the observed precession of Mercury's perihelion.

Contents

1 Introduction

The subject of this text is solving the equations of motion of solar system objects using many-body simulations. We will derive the relevant equations of motion using Newton's laws of gravitation and motion, and discuss some simple numerical methods, in particular the Velocity Verlet method, for solving the resulting system of ordinary differential equations. Our implementations of these methods will be tested by running various scenarios involving different solar system bodies. We will also consider how the Newtonian description can be extended to include general relativistic effects, and use this to reproduce the observed precession of Mercury's orbit.

We have used two independent implementations for solving the many-body problem; one in C++ and one in Fortran 2008. The respective source codes can be found in the following GitHub repositories:

C++:
`https://github.com/AHo94/FYS3150_Projects/tree/master/Project3`

Fortran:
`https://github.com/lars-frogner/FYS4150-Projects/tree/master/Project%203`

For initial conditions of the solar system objects we have used ephemeris data from NASA, available from this link:

`http://ssd.jpl.nasa.gov/horizons.cgi`

We derive the Newtonian and general relativistic equations of motion in section 2.1 and 2.2 respectively, while section 2.3 discusses how these can be efficiently implemented. The numerical methods are presented in section 2.4. In section 3 we give an overview of the workings of the C++ and Fortran implementations used to produce the test results, which are presented in section 4. Finally, our findings are summarised in section 5.

2 Methods

2.1 The equations of motion for a gravitational many-body system

In this section we will formulate the physical problem that we aim to solve, by setting up the equations of motions that we will integrate numerically. Let us start by considering the two-body problem of the sun and Earth. The force $\mathbf{F}_{21}$

on Earth from the sun is given by Newton's law of gravitation:

$$\mathbf{F}_{21} = -Gm_1 m_2 \frac{\mathbf{r}_{21}}{|\mathbf{r}_{21}|^3}$$

where G is the gravitational constant, m_1 and m_2 are respectively the masses of the sun and Earth, and $\mathbf{r}_{21} = \mathbf{r}_2 - \mathbf{r}_1$ is the position of Earth relative to the sun. The acceleration $\mathbf{a}_2 = \mathbf{a}_{21}$ of Earth due to the force from the sun is then found from Newton's second law:

$$\mathbf{a}_2 = \mathbf{a}_{21} = -Gm_1 \frac{\mathbf{r}_{21}}{|\mathbf{r}_{21}|^3}$$

The corresponding acceleration of the sun due to the force from the Earth is

$$\mathbf{a}_1 = \mathbf{a}_{12} = Gm_2 \frac{\mathbf{r}_{21}}{|\mathbf{r}_{21}|^3}.$$

Note that $\mathbf{r}_{21} = -\mathbf{r}_{12}$. Let us add Jupiter (with mass m_3) to the mix. The total acceleration of the sun now gets a new contribution caused by the force from Jupiter:

$$\mathbf{a}_1 = \mathbf{a}_{12} + \mathbf{a}_{13} = -Gm_2 \frac{\mathbf{r}_{12}}{|\mathbf{r}_{12}|^3} - Gm_3 \frac{\mathbf{r}_{13}}{|\mathbf{r}_{13}|^3}$$

The same applies to Earth,

$$\begin{aligned}\mathbf{a}_2 = \mathbf{a}_{21} + \mathbf{a}_{23} &= -Gm_1 \frac{\mathbf{r}_{21}}{|\mathbf{r}_{21}|^3} - Gm_3 \frac{\mathbf{r}_{23}}{|\mathbf{r}_{23}|^3} \\ &= +Gm_1 \frac{\mathbf{r}_{12}}{|\mathbf{r}_{12}|^3} - Gm_3 \frac{\mathbf{r}_{23}}{|\mathbf{r}_{23}|^3},\end{aligned}$$

and Jupiter,

$$\begin{aligned}\mathbf{a}_3 = \mathbf{a}_{31} + \mathbf{a}_{32} &= -Gm_1 \frac{\mathbf{r}_{31}}{|\mathbf{r}_{31}|^3} - Gm_2 \frac{\mathbf{r}_{32}}{|\mathbf{r}_{32}|^3} \\ &= +Gm_1 \frac{\mathbf{r}_{13}}{|\mathbf{r}_{13}|^3} + Gm_2 \frac{\mathbf{r}_{23}}{|\mathbf{r}_{23}|^3}.\end{aligned}$$

We should now be able to convince ourselves that if we have M bodies, the total acceleration of body i becomes

$$\mathbf{a}_i = \sum_{j\neq i} \mathbf{a}_{ij} = \sum_{j=1}^{i-1} \left(Gm_j \frac{\mathbf{r}_{ij}}{|\mathbf{r}_{ij}|^3} \right) + \sum_{j=i+1}^{M} \left(-Gm_j \frac{\mathbf{r}_{ij}}{|\mathbf{r}_{ij}|^3} \right),$$

or more explicitly,

$$\frac{d^2\mathbf{r}_i}{dt^2} = \sum_{j=1}^{i-1} \left(Gm_j \frac{\mathbf{r}_i - \mathbf{r}_j}{|\mathbf{r}_i - \mathbf{r}_j|^3} \right) + \sum_{j=i+1}^{M} \left(-Gm_j \frac{\mathbf{r}_i - \mathbf{r}_j}{|\mathbf{r}_i - \mathbf{r}_j|^3} \right). \qquad (1)$$

Equation (1) represents a set of n coupled ordinary differential equations for each of the three spatial dimensions. These are thus the equations of motion for a many-body problem governed by Newtonian gravity.

2.2 Adding a general relativistic correction

While Newtonian gravity is a very good approximation to reality (i.e. general relativistic gravity) in most scenarios involving solar system bodies, one important exception is the motion of Mercury. The orientation of its orbit around the sun changes slowly with time, by roughly 43 arc seconds per century. This discrepancy is large enough that we can observe it. In section 4.5 we use or developed code to reproduce this result. The precession is a consequence of the gravitational potential close to the sun being a bit steeper according to GR than according to Newton, meaning that Mercury will move slightly faster (especially at perihelion) than Newtonian gravity predicts. The same effect also happens for all the other bodies in the solar system, but the discrepancies are less significant. We can take this effect into account in the equations of motion by adding a correction term to the Newtonian acceleration:

$$\mathbf{a}_{ij}^{\mathrm{GR}} = \mathbf{a}_{ij}\left(1 + \eta_{ij}\left(\mathbf{r}_i, \mathbf{r}_j, \mathbf{v}_i, \mathbf{v}_j\right)\right)$$

The expression for the correction is

$$\eta_{ij} = \frac{3}{m_i c^2}\frac{|\mathbf{l}_{ij}|^2}{|\mathbf{r}_{ij}|^2}, \tag{2}$$

where c is the speed of light and

$$\begin{aligned}\mathbf{l}_{ij} &= m_i\left(\mathbf{r}_{ij} \times \mathbf{v}_{ij}\right)\\ &= m_i\left(\mathbf{r}_i - \mathbf{r}_j\right) \times \left(\mathbf{v}_i - \mathbf{v}_j\right)\end{aligned}$$

is the angular momentum of body i relative to body j. We can write out (2) more explicitly as

$$\eta_{ij} = \frac{3}{c^2}\frac{|\left(\mathbf{r}_i - \mathbf{r}_j\right) \times \left(\mathbf{v}_i - \mathbf{v}_j\right)|^2}{|\mathbf{r}_i - \mathbf{r}_j|^2}.$$

This is a more practical form to use for calculating the correction numerically. To better understand when the correction is significant, we use that

$$|\mathbf{r}_{ij} \times \mathbf{v}_{ij}| = |\mathbf{r}_{ij}||\mathbf{v}_{ij}||\sin\theta| = |\mathbf{r}_{ij}|v_{ij}^{\perp},$$

where θ is the angle between the two vectors and $v_{ij}^{\perp} = |\mathbf{v}_{ij}||\sin\theta|$ is the relative speed in the direction perpendicular to $\mathbf{r}_{ij}$, to write the correction as

$$\eta_{ij} = 3\left(\frac{v_{ij}^{\perp}}{c}\right)^2.$$

So the size of the correction only depends on the size of the circular velocity relative to the speed of light. For Mercury, $\eta \approx 10^{-7}$ at perihelion. The corresponding value for Earth is $\eta \approx 3 \cdot 10^{-8}$.

Note that since $\eta_{ji} = \eta_{ij}$, we can implement the correction by just modifying the common factor $\mathbf{d}_{ij}$;

$$\mathbf{d}_{ij}^{\mathrm{GR}} = \mathbf{d}_{ij}\left(1 + \eta_{ij}\right),$$

rather than the separate accelerations $\mathbf{a}_{ij}$ and $\mathbf{a}_{ji}$.

2.3 Implementing the equations of motion

Let us consider some practical aspects of computing the accelerations $\mathbf{a}_i$ from (1). The straightforward approach is to loop over all other bodies and sum up every contribution to the total acceleration of body i. But we can achieve this more efficiently if we make use of the fact that in calculating $\mathbf{a}_{ij}$ we have already laid the ground work for calculating $\mathbf{a}_{ji}$, since they only differ by a constant factor. More precisely, if we define

$$\mathbf{d}_{ij} \equiv G\frac{\mathbf{r}_i - \mathbf{r}_j}{|\mathbf{r}_i - \mathbf{r}_j|^3},$$

then we have

$$\mathbf{a}_{ij} = -m_j \mathbf{d}_{ij} \tag{3}$$

and

$$\mathbf{a}_{ji} = m_i \mathbf{d}_{ij}. \tag{4}$$

So, starting with body 1, we should calculate the contributions $\mathbf{a}_{1j}$ using (3) and add them to get $\mathbf{a}_1$, and at the same time calculate each $\mathbf{a}_{j1}$ using (4) and add it to $\mathbf{a}_j$. When we then get to body 2, $\mathbf{a}_{21}$ has already been added to $\mathbf{a}_2$, so we only have to consider the contributions $\mathbf{a}_{2j}$ for $j > 2$. For these we proceed just like we did with body 1. The procedure continues until we get to body M, and at this point the acceleration $\mathbf{a}_M$ is already determined, since it has recieved all the contributions from the other bodies. The following pseudo code illustrates a function that calculates the accelerations using this approach:

```
define function a(r[1:M])

   for i = 1 to M
      a[i] = 0
   end for

   for i = 1 to M-1
      for j = i+1 to M

         d = G*(r[i] - r[j])/norm(r[i] - r[j])^3
         a[i] = a[i] - m[j]*d
```

```
            a[j] = a[j] + m[i]*d

        end for
    end for

    return a[1:M]

end function
```

How much do we gain in terms of number of floating point operations (FLOPS) by doing it this way compared to the straightforward way? The straightforward approach uses an outer loop over M bodies and an inner loop over $M-1$ bodies, while the improved approach uses an outer loop over $M-1$ bodies and an inner loop over $M-i$ bodies. This gives a total of $M(M-1)$ inner loops for the former approach and $M(M-1)/2$ inner loops for the latter. Since the number of FLOPS performed in an inner loop should be roughly the same for both cases, we see that the improved approach only requires about half the amound FLOPS.

One final matter to consider is the units that we use for time, length and mass. On solar system scales, SI units is impractical since it will yield very large values, which makes the code more susceptible to numerical problems like overflow. A more natural choice is to use years for time, astronomical units (AU) for length and solar masses ($M_\odot$) for mass. We then need the value for the gravitational constant G in terms of these units. Our particular combination of units gives a very simple numerical value for G. To see this, consider the circular velocity of Earth:

$$v_{\text{circ}} = \sqrt{\frac{GM_\odot}{1\ \text{AU}}}$$

This gives

$$G = \frac{1\ \text{AU}}{1\ M_\odot} \cdot {v_{\text{circ}}}^2.$$

We can also write the circular velocity as

$$v_{\text{circ}} = \frac{2\pi \cdot 1\ \text{AU}}{1\ \text{yr}},$$

which we insert into the above expression for G:

$$\begin{aligned} G &= \frac{1\ \text{AU}}{1\ M_\odot} \cdot \left(\frac{2\pi \cdot 1\ \text{AU}}{1\ \text{yr}}\right)^2 \\ &= 4\pi^2\ \text{AU}^3\ {M_\odot}^{-1}\ \text{yr}^{-2} \end{aligned}$$

So in units of years, AU and solar masses, the gravitational constant is just $4\pi^2$.

2.4 Solving the equations of motion numerically

Consider an equation of motion in 3D on the general form

$$\frac{d^2\mathbf{r}}{dt^2} = \mathbf{a}.$$

This second order differential equation can be decomposed into two first order differential equations by including the velocity $\mathbf{v}$:

$$\begin{aligned}\frac{d\mathbf{r}}{dt} &= \mathbf{v}\\ \frac{d\mathbf{v}}{dt} &= \mathbf{a}\end{aligned}$$

A common approach for solving these equations numerically is to approximate the derivatives in terms of finite differences obtained from Taylor expansions. Suppose we want the solution for $N+1$ discrete time points n between $t_0 = 0$ and $t_N = T$. The time points are separated by a time $h = T/N$. We denote the position and velocity at time t_n as $\mathbf{r}(t_n) \equiv \mathbf{r}_n$ and $\mathbf{v}(t_n) \equiv \mathbf{v}_n$. We can Taylor expand the velocity and position around $t = t_n + h$ to get

$$\mathbf{v}_{n+1} = \mathbf{v}_n + \mathbf{a}_n h + \mathcal{O}(h^2) \tag{5}$$

$$\mathbf{r}_{n+1} = \mathbf{r}_n + \mathbf{v}_n h + \mathcal{O}(h^2). \tag{6}$$

This gives rise to the *Euler* method, where the velocity and position is advanced to the next time using the following expressions:

$$\mathbf{v}_{n+1} = \mathbf{v}_n + h\mathbf{a}_n \tag{7}$$

$$\mathbf{r}_{n+1} = \mathbf{r}_n + h\mathbf{v}_n \tag{8}$$

The error introduced in each step is $\mathcal{O}(h^2)$, so after N steps it will be $\mathcal{O}(h)$ (since $N \propto 1/h$). If we replace the current velocity $\mathbf{v}_n$ in (8) with the next velocity $\mathbf{v}_{n+1}$ given by (7), we obtain the *Euler-Cromer* method:

$$\mathbf{v}_{n+1} = \mathbf{v}_n + h\mathbf{a}_n \tag{9}$$

$$\mathbf{r}_{n+1} = \mathbf{r}_n + h\mathbf{v}_{n+1} \tag{10}$$

This method also has an $\mathcal{O}(h)$ global error, but conserves energy better than the standard Euler method.

We typically want a better global accuracy than $\mathcal{O}(h)$. To obtain a method that achieves this, we can start by including another term in the Taylor approximations (5) and (6) that we used to derive the Euler method:

$$\mathbf{r}_{n+1} = \mathbf{r}_n + \mathbf{v}_n h + \mathbf{a}_n \frac{h^2}{2} + \mathcal{O}(h^3) \tag{11}$$

$$\mathbf{v}_{n+1} = \mathbf{v}_n + \mathbf{a}_n h + \left[\frac{d\mathbf{a}}{dt}\right]_n \frac{h^2}{2} + \mathcal{O}(h^3) \tag{12}$$

To keep the $\mathcal{O}(h^3)$ error, we need an expression for the h^2 term in (12) that is at least third order accurate. To obtain such an expression we can do a Taylor expansion of the acceleration:

$$\begin{aligned}
\mathbf{a}_{n+1} &= \mathbf{a}_n + \left[\frac{d\mathbf{a}}{dt}\right]_n h + \mathcal{O}(h^2) \\
\Rightarrow \quad \left[\frac{d\mathbf{a}}{dt}\right]_n h &= \mathbf{a}_{n+1} - \mathbf{a}_n + \mathcal{O}(h^2) \\
\Rightarrow \quad \left[\frac{d\mathbf{a}}{dt}\right]_n \frac{h^2}{2} &= (\mathbf{a}_{n+1} - \mathbf{a}_n)\frac{h}{2} + \mathcal{O}(h^3)
\end{aligned}$$

Inserting this back into (12) gives

$$\begin{aligned}
\mathbf{v}_{n+1} &= \mathbf{v}_n + \mathbf{a}_n h + (\mathbf{a}_{n+1} - \mathbf{a}_n)\frac{h}{2} + \mathcal{O}(h^3) \\
&= \mathbf{v}_n + (\mathbf{a}_{n+1} + \mathbf{a}_n)\frac{h}{2} + \mathcal{O}(h^3).
\end{aligned}$$

This together with (11) defines the *Velocity Verlet* method:

$$\mathbf{r}_{n+1} = \mathbf{r}_n + h\left(\mathbf{v}_n + \frac{h}{2}\mathbf{a}_n\right) \tag{13}$$

$$\mathbf{v}_{n+1} = \mathbf{v}_n + h\left(\frac{\mathbf{a}_{n+1} + \mathbf{a}_n}{2}\right) \tag{14}$$

The local error for this method is $\mathcal{O}(h^3)$, and the global error is $\mathcal{O}(h^2)$. Note that (14) requires the acceleration $\mathbf{a}_{n+1}$, which in turn depends on the position $\mathbf{r}_{n+1}$ given by (13). So the new acceleration must be evaluated in between the calculations of $\mathbf{r}_{n+1}$ and $\mathbf{v}_{n+1}$. But this requirement causes problems if the acceleration depends on velocity (which is the case if the relativistic correction is being used), since $\mathbf{v}_{n+1}$ is not known when $\mathbf{a}_{n+1}$ is being evaluated. A way around this is to calculate a value for $\mathbf{v}_{n+1}$ with a simple Forward Euler step, and use this together with (13) when evaluating $\mathbf{a}_{n+1}$.

Notice that the quantity $\mathbf{v}_n + h\mathbf{a}_n/2$ is can be found in both (13) and (14). This is just a Forward Euler approximation of the velocity at time $t_n + h/2$. So we can formulate the method in the following alternative way:

$$\mathbf{v}_{n+1/2} = \mathbf{v}_n + \frac{h}{2}\mathbf{a}_n \tag{15}$$

$$\mathbf{r}_{n+1} = \mathbf{r}_n + h\mathbf{v}_{n+1/2} \tag{16}$$

$$\mathbf{v}_{n+1} = \mathbf{v}_{n+1/2} + \frac{h}{2}\mathbf{a}_{n+1} \tag{17}$$

This is a slightly more effective way of performing the calculations, since we have isolated a quantity that occurs in both expressions, and thus only have to calculate it once per time step.

Let us consider how to implement the Velocity Verlet method for solving the equation of motions for the many-body problem. Since we have M bodies, each with their corresponding position, velocity and acceleration, equations (15), (16) and (17) must be applied once for each body. But since the acceleration of each body depends on the positions of all the other bodies, we need to advance the positions of all the bodies before calculating the accelerations. This boils down to the following pseudo code, where the accelerations are obtained by calling the function defined in section 2.3:

```
a[1:M] = a(r[1:M])

for n = 1 to N

   for i = 1 to M
      v[i] = v[i] + a[i]*h/2
      r[i] = r[i] + v[i]*h
   end for

   a[1:M] = a(r[1:M])

   for i = 1 to M
      v[i] = v[i] + a[i]*h/2
   end for

end for
```

The position and velocity arrays are assumed to contain the initial conditions at the beginning of the pseudo code.

We have already determined that the accuracy of the Velocity Verlet method is one order better than that of the Euler-Cromer method, but how does the required number of FLOPS compare for the two methods? For the Euler-Cromer method, each advancement of a position and velocity vector requires 12 FLOPS (2 additions and 2 multiplications for each of the 3 spatial dimensions). For the Velocity Verlet method this increases to 18, due to the inclusion of an additional Euler step ($h/2$ is assumed to be pre-calculated). In each time step, we must apply the solution method M times; one for each body, in addition to evaluating the acceleration function once (which requires $\mathcal{O}(M^2)$ operations). So in total we get

$$\text{FLOPS for Euler-Cromer} = N\left(12M + \mathcal{O}(M^2)\right)$$
$$\text{FLOPS for Velocity Verlet} = N\left(18M + \mathcal{O}(M^2)\right).$$

Although the Velocity Verlet method in itself requires 50 % more operations than the Euler method, we can see that this difference quickly becomes insignificant when we have more bodies and most of the CPU time is spent on evaluating the accelerations. So for many-body problems there is a big incentive for using the Velocity Verlet method rather than the Euler-Cromer method.

3 Implementation

3.1 C++

For the C++ program, it is divided in two parts. Creating the system and calculation the forces between the planets are done in C++. The data will then be saved to a .txt file which is then read and plotted in Python. All planetary data (with the exception for the escape velocity of a planet), which is used to determine the initial conditions of the planets, were obtained from NASA's Horizons page.

This program is an object oriented program, in which we use classes to create objects and use them to calculate. The C++ program has multiple classes which is as follows:

- `vec3`: This class is a vector class which allows us to do basic vector operations. I.e vector dot products, vector cross products, find the length of the vector, additions of two vectors and so on.
- `celestials`: A class that sets the initial conditions and masses for each planet.
- `odesolvers`: Contains all the algorithms that we will use for the project. That is the Verlet, Euler and Euler Cromer method.
- `solarsystem`: This class sets up the whole system of the program. It will use the `celestials` class to set the initial conditions and masses of a given planet. This class will also calculate the forces when we call one of the solvers from `odesolvers`. In addition to that, this class will save the positions of all objects to a file.

  Unit tests for energy and angular momentum conservation applies to every system. The C++ program will first calculate the total energy, for one time step, of the whole system and then save it to a variable. In the next time step, it will again calculate the total energy and then compare it to the previous total energy. That is

  $$\Delta E = |E_{new} - E_{old}|$$

  This will therefore be the total difference between the energies. The same idea applies for the total angular momentum. The absolute value of the difference between the old and new total energy/total angular momentum should ideally be zero. However, due to round off errors in programming (from float numbers), this will not be the case, so we will have to set a certain error threshold ϵ. In the C++ program, we have selected $\epsilon = 3 \cdot 10^{-4}$.

The energy of the system must be conserved because gravity as a force is a conservative force. This is also the only force we take into account in the project. The kinetic energy and potential energy will therefore not vanish due to forces like friction (which we have ignored). Angular momentum for the system must also be conserved because we assume that the system is not affected by an external torque. This is also because we assume that our system is isolated, and therefore not affected by external forces.

3.2 Fortran

The Fortran program is structured around a superclass `bodies` that represents a general collection of bodies with properties like mass, position and velocity. All the values for these quantities are contained in arrays bound to each class instance. The class contains a method `solve_equations_of_motion` that integrates the equations of motion for the objects using a given solution method (like Velocity Verlet) and writes the results to an output file. There is also a unit test that can be used to check whether energy, linear momentum and angular momentum is conserved for the system of bodies. This is achieved by printing out the relative difference of each quantity between the beginning and end of the simulation.

Note that the equations of motion are given by how the accelerations are calculated, and this is not specified in the class in order to keep it general. The method calculating the accelerations, `get_accelerations`, must rather be provided by a subclass inheriting `bodies`. The subclass `celestial_bodies` does exactly this, and implements the routine `get_accelerations` similarly to the one outlined in section 2.3. It also has a method for initialising the celestial bodies with the masses and initial conditons obtained from the NASA ephemeris. There is also a class `gr_celestial_bodies` that inherits `celestial_bodies` and overrides the ordinary acceleration routine with one that includes the general relativistic correction. The actual instantiation of class objects and calling of the `solve_equations_of_motion` method is performed by the interface program `solar_system.f90`, which is controlled by command line arguments. The Python script `plot_orbits.py` is a more user friendly wrapper for `solar_system.f90` that also handles the preparation of ephemeris data and visualisation of the results.

4 Results

4.1 Testing the algorithm

One can calculate the initial velocity to, which ensures circular orbits, analytically. The only force in the system that we will think of is the gravitational force. Newton's second law says that

$$M_{earth}a = \frac{GM_{earth}M_{\odot}}{r^2}$$

With circular orbits, the acceleration is given as $a = \frac{v^2}{r}$, plugging this in and doing some simple manipulations gives

$$v = \sqrt{\frac{GM_{\odot}}{r}}$$

From the theory section we have shown that $GM_{\odot} = 4\pi^2$ and if Earth starts exactly 1 AU away from the Sun, then $r = 1$ and the initial velocity to get circular orbits will be

$$v = 2\pi$$

For now, we will assume that Earth starts 1 AU away from the Sun in x-direction, and the velocity will be directed only in the y-direction.

We will now test the Euler and Verlet method, presented earlier, to solve the Earth-Sun system. We will also see how stable these methods are for different values of Δt. For the C++ program, we have tested Δt values of 10^{-4} and 10^{-3}. The number of mesh points used for each Δt values is $N = 100000$ and $N = 10000$ respectively. This is to ensure that the time elapsed, for both simulations, are the same.

Figure 1 shows the Euler method with $\Delta t = 10^{-4}$ and figure 2 also shows the Euler method, but for $\Delta t = 10^{-3}$. As we can see, the stability is a lot better for smaller time steps. However, it appears that Earth is going further and further away from the Sun. The Euler method is not very good at calculating circular orbits, which is why the results are not exactly what we want. Euler method calculates the tangent on the current point and it does not take into account the curvature of the points.

Let us now look at the Verlet method. Figure 3 shows the Verlet method using $\Delta t = 10^{-4}$. From this figure, we already see that the Verlet method gives a much better result compared to the ones using the Euler method. Increasing the step length to $\Delta t = 10^{-3}$ would give identical results, so for the Earth-Sun system, Verlet method is very stable up to $\Delta t = 10^{-3}$. A plot for the Verlet method with $\Delta t = 10^{-3}$ can be found in the appendix in figure 13.

For the C++ program, the time difference between these two methods are almost negligible for a small number of time steps. As seen in the blue text

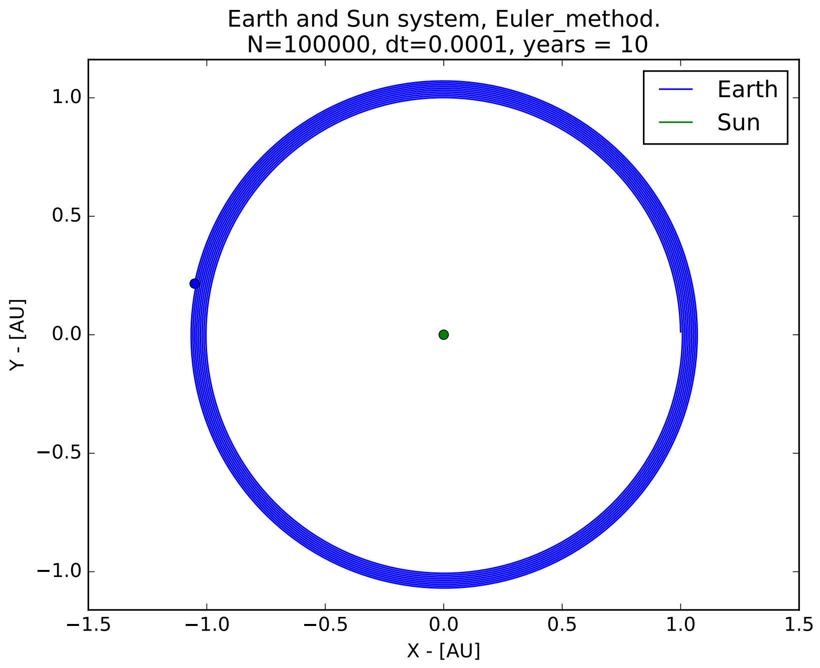

Figure 1: Euler method using $\Delta t = 10^{-4}$

below, the Verlet method had to run roughly 2 times longer compared to the Euler method, which is not surprising due to the increased number of FLOPS required to solve one time step. However, as we saw from the plots, the extra time it took for the Verlet method gave a much better result than the one using Euler method. Also, as mentioned earlier in the methods sections, when we increase the number of planets we calculate, the computation time between Euler and Verlet method will be negligible. We will therefore stick with the Verlet method for the remainder of the project.

```
Running Euler method
Time elapsed for Euler method:0.1s
Data saved to: Earth_Sun_sys_euler.txt

Running Euler Cromer method
Time elapsed for Euler Cromer method:0.097s
Data saved to: Earth_Sun_sys_eulercromer.txt

Running Verlet method
```

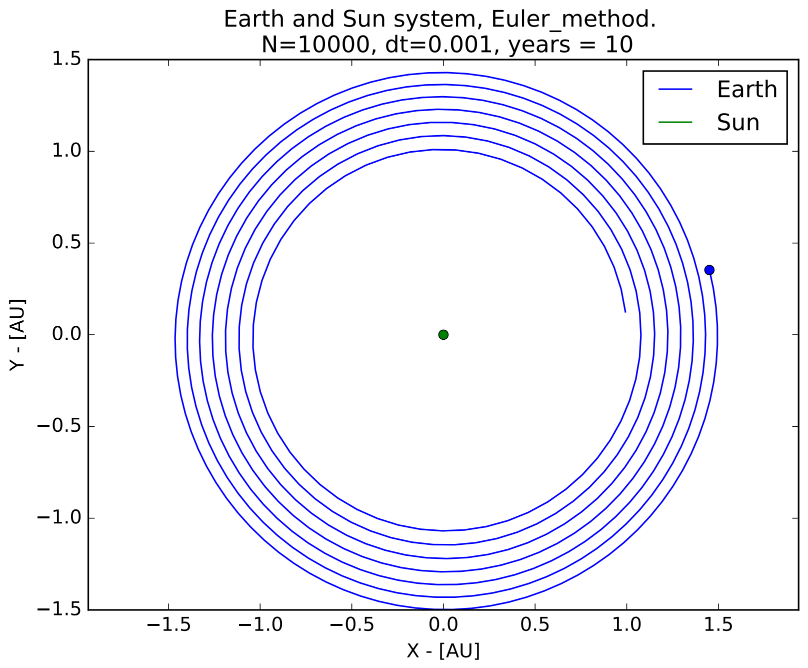

Figure 2: Euler method using $\Delta t = 10^{-3}$.

```
Time elapsed for Verlet method:0.202s
Data saved to: Earth_Sun_sys_verlet.txt
```

Both the total energy and total angular momentum are conserved for the given tolerance, $\epsilon = 3 \cdot 10^{-4}$, for all three algorithms, as the output gave no warnings of them.

One can also use the Euler Cromer method to solve these systems. Figure 14 shows a plot of the Earth-Sun system using the Euler Cromer method. As we can see, the results practically identical to the Verlet method. The computation speed of the Euler Cromer method is almost identical as the computation speed of the Euler method.

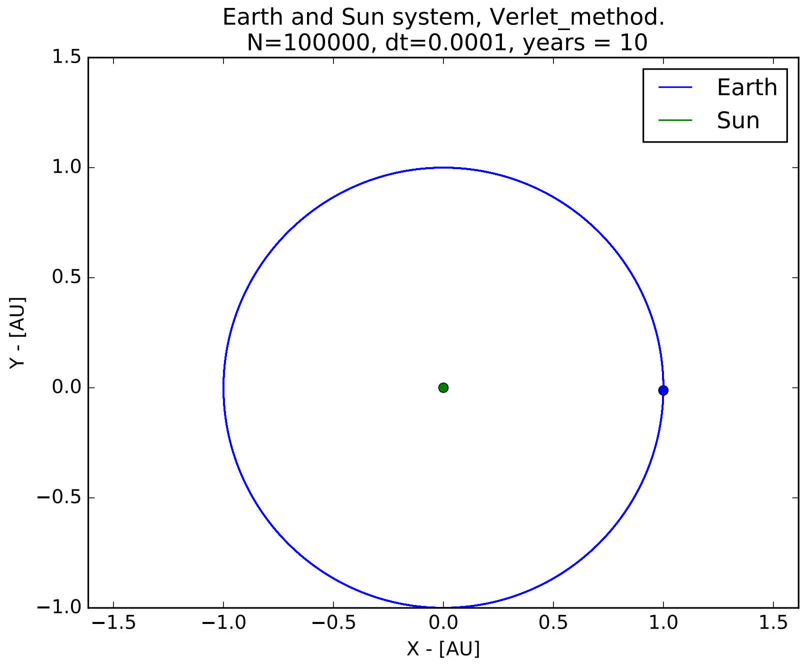

Figure 3: Verlet method using $\Delta t = 10^{-4}$

4.2 Escape velocity

One can calculate the escape velocity of a planet analytically. The escape velocity of an object with mass M_1 is the velocity when the object's kinetic energy is equal to the potential energy. That is

$$\frac{1}{2}M_1 v = \frac{GM_1M_2}{r}$$

Let us consider a planet with mass M orbiting around the Sun. The escape velocity for this planet is then

$$v_{escape} = \sqrt{\frac{2GM_\odot}{r}}$$

In terms of AU per year, we have that $GM_\odot = 4\pi^2 \text{AU}^3/\text{yr}^2$, so

$$v_{escape} = \sqrt{\frac{8\pi^2}{r}}\text{AU/yr}$$

It is now quite easy to implement the escape velocity. In the C++ program, we have assumed, for simplicity, that the planet has the mass of Earth and start 1 AU away from the Sun in the x direction. The velocity will be the escape velocity directed in the y-direction.

Figure 4 shows a plot of a the planet with mass M_{earth} and how the orbit is if the initial velocity is equal to the escape velocity. As we can see, starting with the escape velocity allows the planet to break free from it's intended orbit and leave the system. The orbit of Earth, with circular orbit used previously, is also plotted as a comparison.

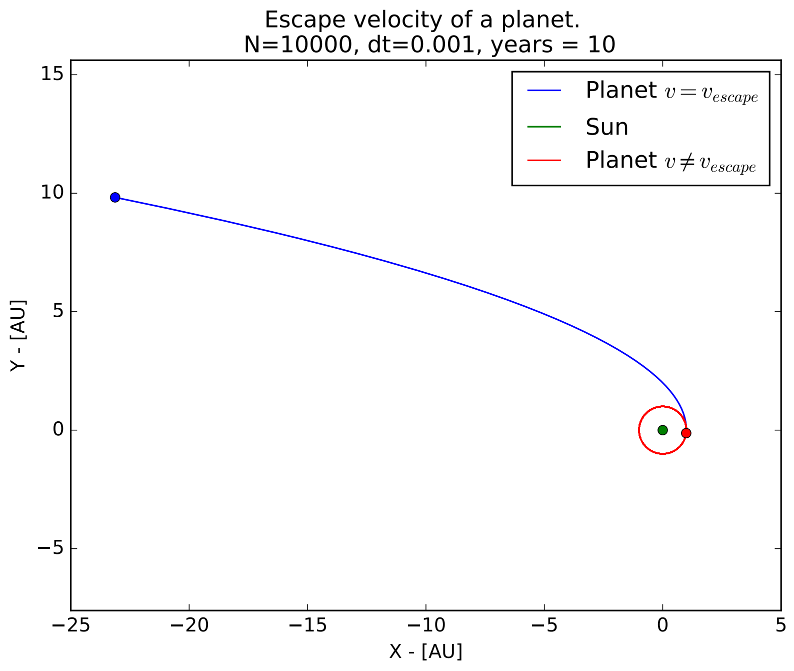

Figure 4: Plot shows a planet with the mass of Earth with and without escape velocity.

4.3 The three-body problem

From now on, all the initial conditions, i.e positions and velocities of both the Sun and Earth will be obtained from NASA. The date of the data is 2016.10.05. We already know that Earth's orbit will change, from the theory we previously introduced, when we add Jupiter to the system. Figure 5 shows the orbit of the Earth and Jupiter, as well as the Sun. The orbits themselves does not look particularly odd in any way, but it is hard to see how Jupiter affects the orbit of the Earth. Instead, we only plot the orbit of the Earth and the Sun (still taking Jupiter into account during the calculations) which can be found in figure 6.

As we can see from figure 6, the curves tracing the orbits of Earth have changed

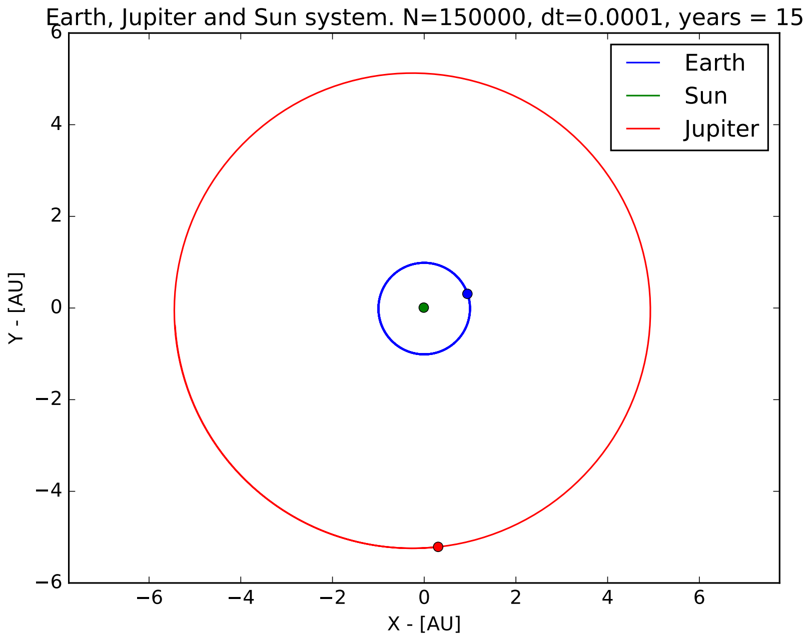

Figure 5: Orbits of the Earth, Jupiter and the Sun.

slightly with respect to each other, since the line appears to be a bit thicker. This indicates that the orbit has become slightly more elliptical.

The Verlet method for this system is still quite stable, given that $\Delta t = 10^{-4}$. We can also test the stability of the algorithm by increasing the time step to $\Delta t = 10^{-3}$. At the same time, we will have to reduce the number of steps N to match the simulation time. As seen in figure 15 in the appendix, the result is practically identical to the one with $\Delta t = 10^{-4}$.

Increasing the mass of Jupiter will change the orbits of Earth, the Sun, as well as Jupiter itself even more. Figure 7 shows the system when we increase the mass of Jupiter by a factor of 10. We can already see that Earth's orbit has changed a little bit, but the effect of a 10 times more massive Jupiter is not very significant to the whole system. Both planets, even after 20 years, are still in orbit around the Sun.

Figure 8 shows the system when the mass of Jupiter is increased by a factor of 1000, and the result is very interesting. At this point, Jupiter has practically the same mass the Sun, so the gravitational effects from that should be quite dramatic. As we see from the figure, the Sun starts to move due to the increased mass of Jupiter. We also notice that Earth is getting flung around

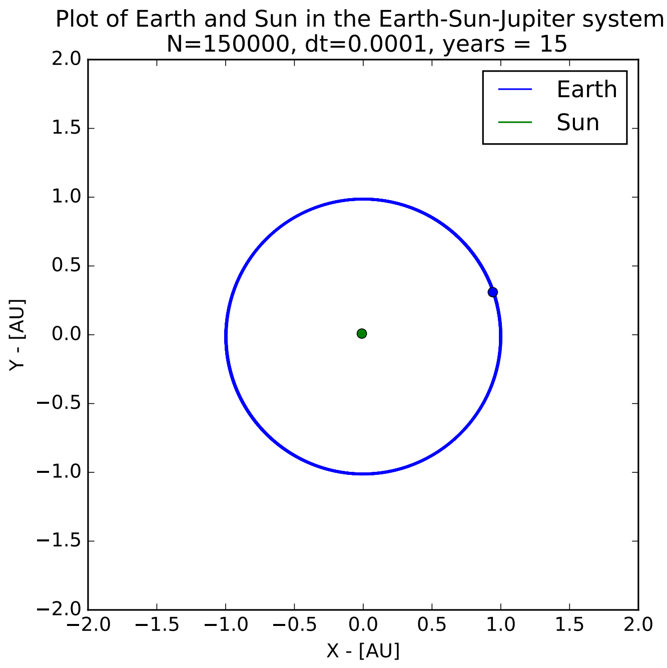

Figure 6: Plot of the Earth and the Sun in the Earth-Sun-Jupiter system.

between these two massive objects.

The center of mass (CM) also starts to move because of the increased mass of Jupiter. The movement of the CM depends on the momentum of each object. Now that Jupiter has an increased mass, and the initial velocity is unchanged, the sudden increase of momentum will shift the CM as a whole.

3D Plots of this system with unchanged Jupiter mass and 1000 times the Jupiter mass can be found in the appendix in figure 16 and 17 respectively.

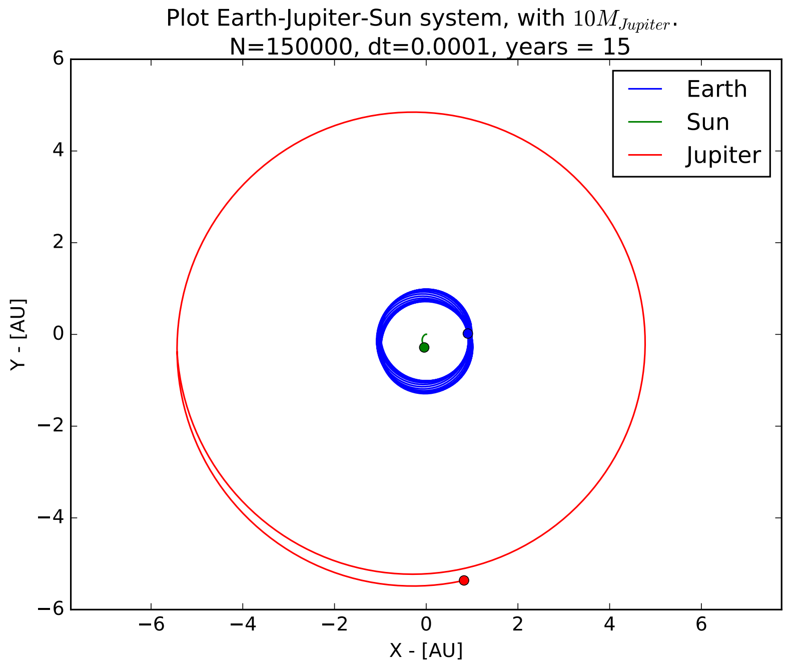

Figure 7: Earth-Sun-Jupiter system for $10M_J$.

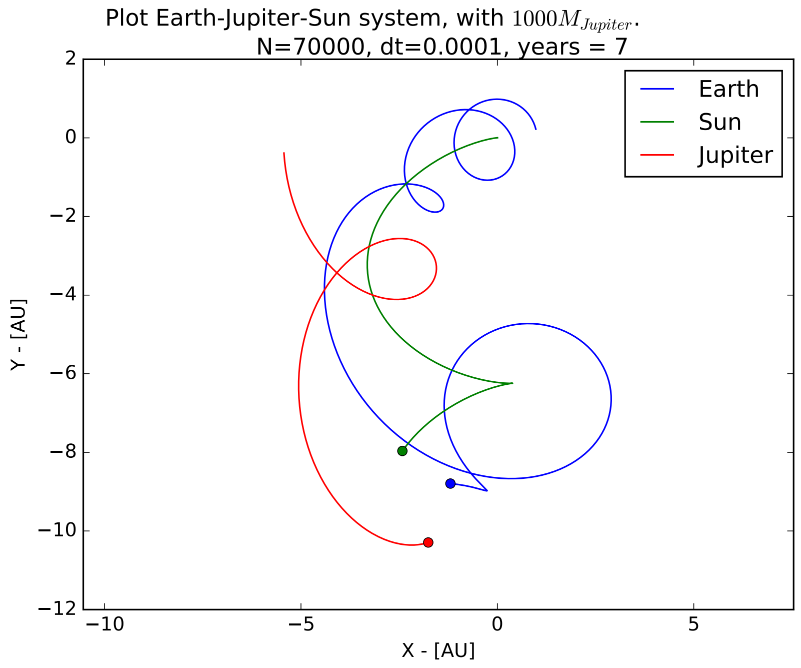

Figure 8: Earth-Sun-Jupiter system for $1000M_J$.

4.4 All planets

We will now add all planets, including Pluto, into our system. This is quite straight forward to do. In the C++ program, all we have to do is to add an extra force to each planet. Figure 9 shows the orbits of all planets in 3 dimensions, for 250 years. The initial velocity of the Sun is set, so that the total momentum of the system is conserved, that is

$$M_{\odot}v_{\odot} + \sum_i M_i v_i = 0 \tag{18}$$

Because of the orbit sizes of the outer planets (i.e Jupiter, Saturn, Uranus,

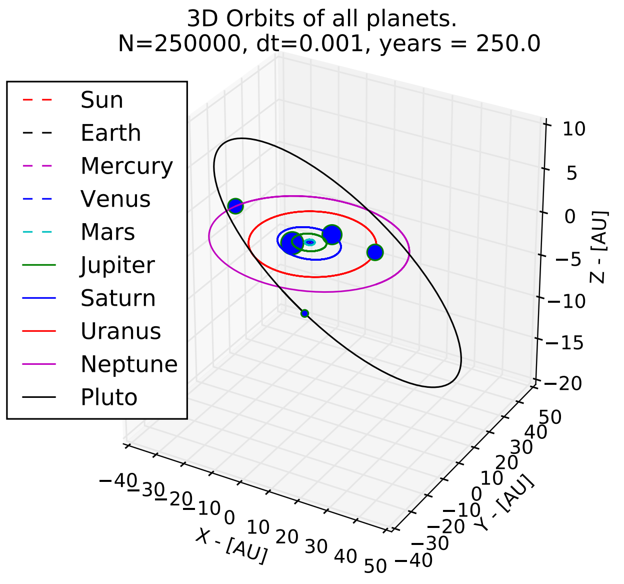

Figure 9: Plot of all planets in 3D. Note: the scatter plot (spheres) of the planet does not scale with it's real size.

Neptune and Pluto), the orbits of the first four planets will be packed in the center of the system, which is quite hard to analyse. What we can do is to only plot the first four planets, while still taking the effect of the gravitational effects of the other planets into account.

However, plotting 250 years worth of orbits of the first four planets are quite redundant so we have decided to reduce the number of time steps, Δt. This is to, again, mainly to reduce the number of simulated years, but it will also increase the stability of the Verlet method. The orbits of the first four planets under these settings can be found in figure 10. One thing to note is that the outer planets will no longer have full orbits, like in figure 9. However, the effect from the gravitational pull of these planets are still present, which is the most important part.

The results are promising. The orbits of all planets resembles the orbits we

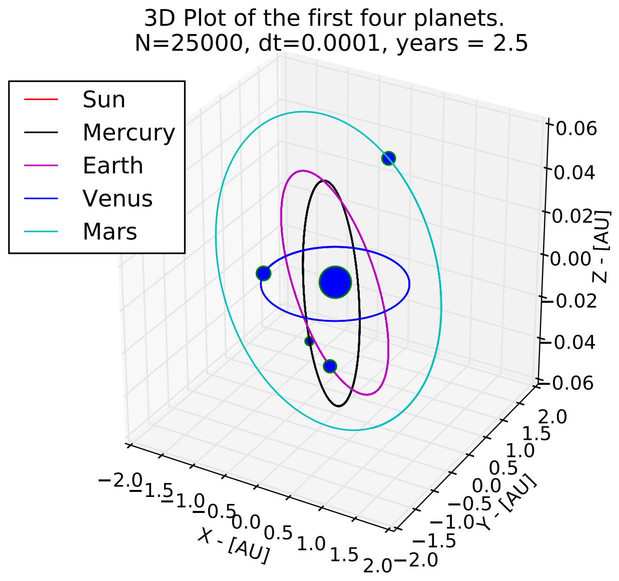

Figure 10: 3D plot of the first four planets orbit. Note: the scatter plot (spheres) of the planet does not scale with it's real size.

observe. When looking at the plot for the first four planets, one may quickly be confused by the orbits, as it seems like the planets (with the exception of Earth) goes very high up in the z-axis. But this movement is at most up to $\pm 0.04 AU$, which is a very small movement. By looking at figure 9, the vertical movement of the first four planets are practically zero, when we include Pluto to the system.

4.5 Precession of Mercury's perihelion

In this section we discuss our simulation of Mercury's perihelion precession due to general relativity, which uses the correction to Newtonian gravitation described in section 2.2. There are several effects that cause the orientation of Mercury's orbit to change over time, the most important of which being the gravitational forces from the other planets. When trying to reproduce the much smaller precession caused by general relativity, we therefore limited ourselves to the two-body system of the sun and Mercury.

We initialised a simulation with the center of mass at the origin, and Mercury in the perihelion position $\mathbf{r}_{21} = (0.3075, 0, 0)$ with the perihelion velocity $\mathbf{v}_{21} = (0, 12.44, 0)$ (note that the units are still AU for length and years for time). The sun was given a corresponding speed in the opposite direction in order to make the center of mass stationary.

A high degree of precision was required to resolve the minute variations in the perihelion position. As mention in section 2.2, the orientation of the orbit changes by about $43''$, or $2.08 \cdot 10^{-4}$ rad, per century. At perihelion this corresponds to a physical distance of 0.3075 AU $\cdot\, 2.08 \cdot 10^{-4}$ rad $= 6.4 \cdot 10^{-5}$ AU $=$ 9570 km. Mercury traverses this distance in $6.4 \cdot 10^{-5}$ AU/12.44 AU yr^{-1} $=$ $5.14 \cdot 10^{-6}$ yr $= 162$ s. How finely we resolve Mercury's movement will decide the precision to which we can determine the precession angle. If the angular distance between two subsequent positions near the perihelion is $\Delta\theta$, our measurement of the perihelion angle should have a precision of $\pm\Delta\theta/2$. If we require $\Delta\theta = 1''$, we need a distance of 0.3075 AU $\cdot\, 1'' = 0.3075$ AU $\cdot\, 4.85 \cdot 10^{-6}$ rad $= 1.49 \cdot 10^{-6}$ AU, or about 223 km, between each measurement. The required temporal resolution for this is $1.49 \cdot 10^{-6}$ AU/12.44 AU yr^{-1} $= 1.20 \cdot 10^{-7}$ yr, which is just below 4 seconds. If we use a constant time step size, this corresponds to over 800 million time steps for a 100 year simulation. To avoid having to handle such a large amount of output data, we adjusted the code to let us write the results to file only for the final orbit, since this is all we need to measure the final perihelion position.

Figure 11 is a plot of a section of the first and last orbit of a 100 year simulation, with the perihelion positions of the orbits indicated.

The angle between the two lines connecting the sun and the perihelions is the precession angle. The time step size used for this simulation was about $6 \cdot 10^{-8}$ yr, giving a precision of about $\pm 0.2''$. We see that our result of $43.2''$ agrees with the observed precession angle. Figure 12 gives a closer view to better show the difference between the two orbits.

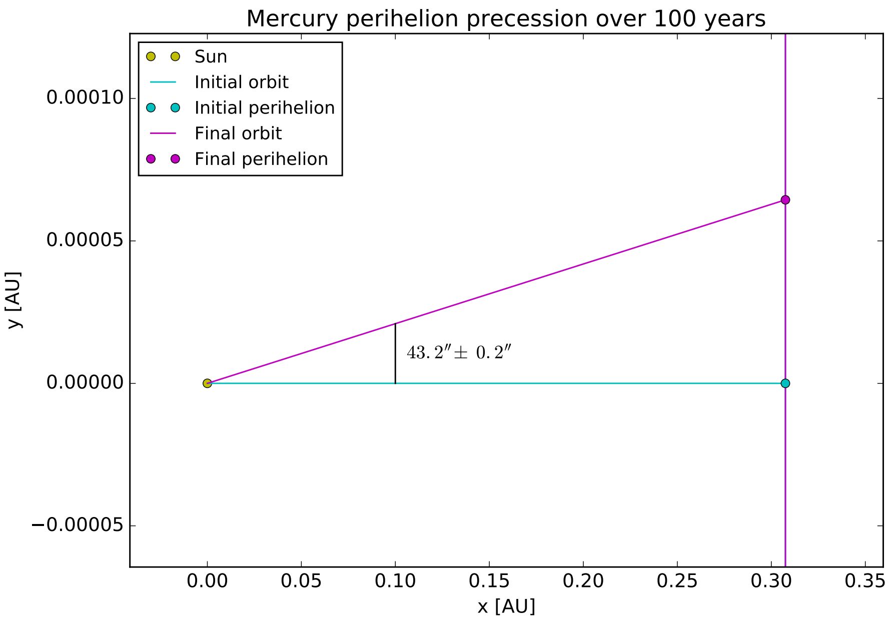

Figure 11: Plot of two orbits of Mercury separated by 100 years. Note that the plot is strongly zoomed in the y-direction to show the minute difference between the two perihelions, causing the orbits to appear to overlap completely.

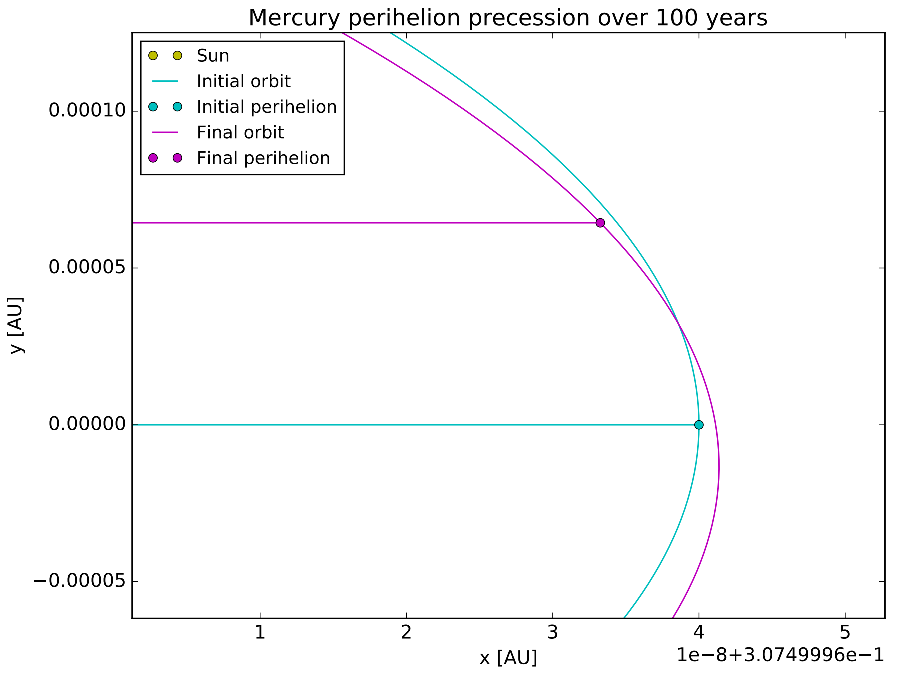

Figure 12: The same orbits as in figure 11 are here shown with a very strong zoom in the x-direction to make the different orientations of the orbits visible.

5 Conclusions

As we saw when we simulated the Earth-Sun system, the Verlet method was clearly the algorithm of choice, as the Euler method did not give stable circular systems. Even though the Verlet method requires more FLOPS to run, the computation time between it and the Euler method is negligible when we start to add more objects to the system.

All the planetary orbits, whether it was for Earth-Sun system, Earth-Sun-Jupiter system or even the whole Solar system, gave promising results that resembles real orbits we observe. Increasing the mass of one of the Planets, as seen in the three body problem, changed the orbits of all the celestial objects. If we increase the mass enough, then the system would be very chaotic, which we saw when we increased Jupiter's mass by 1000.

With the general relativistic correction we were also able to reproduce the observed precession of Mercury's orbit.

6 Appendix

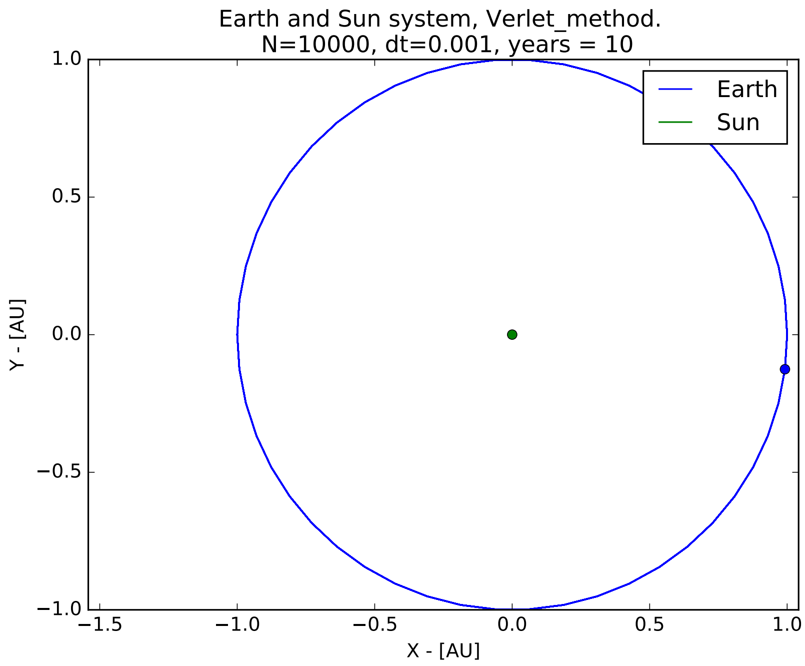

Figure 13: Verlet method using $\Delta t = 10^{-3}$

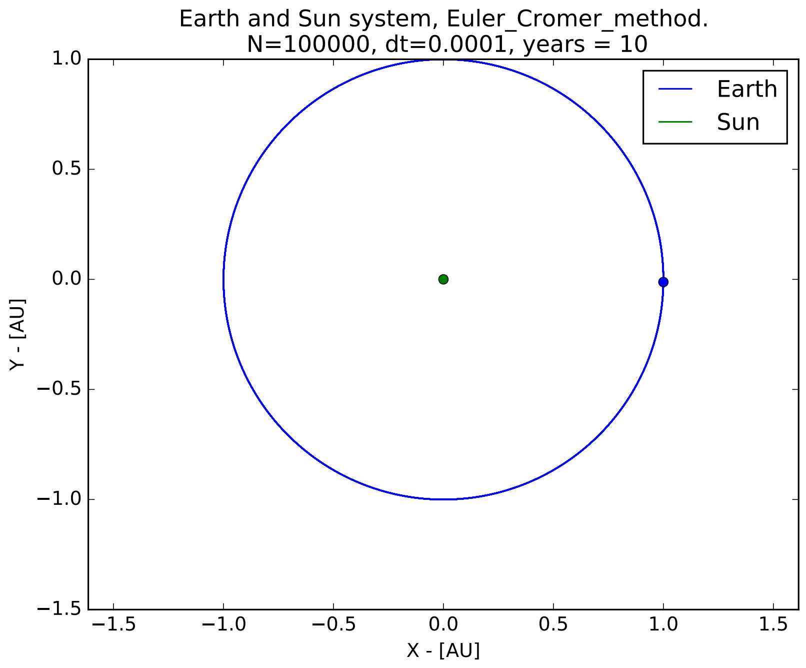

Figure 14: Earth-Sun system using Euler Cromer method

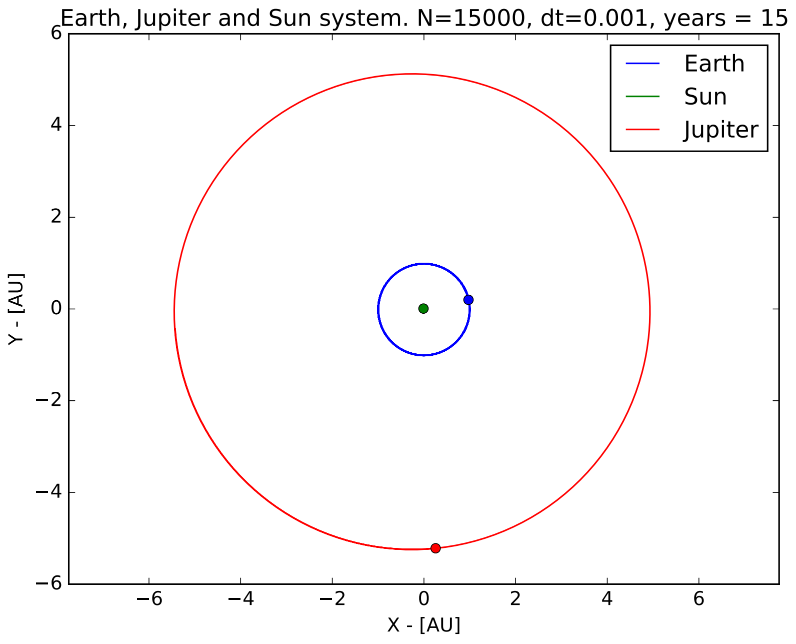

Figure 15: Earth-Sun-Jupiter system for $\Delta t = 10^{-3}$. The result is identical to the system with $\Delta t = 10^{-4}$.

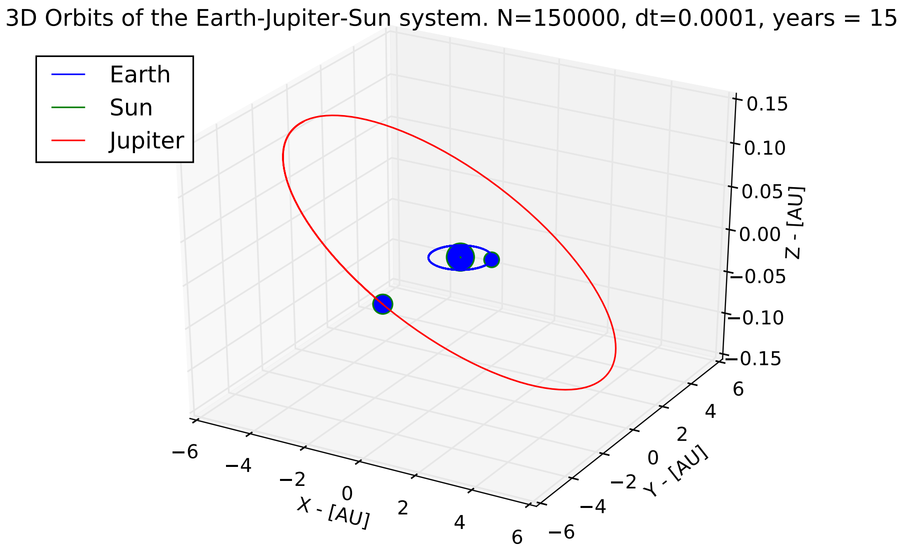

Figure 16: 3D Plot of the Earth-Sun-Jupiter system.

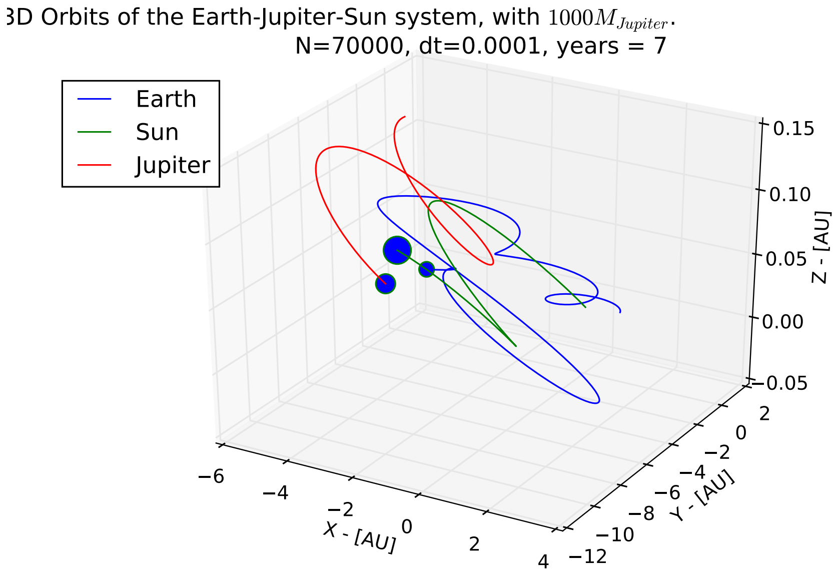

Figure 17: 3D plot for Earth-Sun-Jupiter system with the mass of Jupiter as $1000M_{Jupiter}$

References

[1] M. Hjorth-Jensen, *Computational Physics*, 2015, 551 pages