

# FYS4150 - Computational Physics

## Project 3

Alex Ho

October 19, 2016

### Implementation

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 <http://ssd.jpl.nasa.gov/horizons.cgi>.

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.
- *celestial*s: 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 *celestial*s class to

Unit test for energy conservation applies to every system. The 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. The absolute value of the difference between these two total energies should ideally be zero. However, due to round off errors in programming, this will not be the case, so we will have to set a certain error threshold  $\epsilon$ . In the C++ program, we have selected  $\epsilon = 10^{-4}$ .

**3a)**

**3b)**

Implementing the Euler and Verlet method is quite easy. We have the Euler method as

$$\begin{aligned}x_{i+1} &= x_i + v_i dt \\v_{i+1} &= v_i + a_i dt\end{aligned}$$

One needs to calculate the acceleration  $a_i$  using the discretized equations that we have previously shown. One can also implement a more precise method, the Euler Cromer's method, which is given as

$$\begin{aligned}v_{i+1} &= v_i + a_i dt \\x_{i+1} &= x_i + v_{i+1} dt\end{aligned}$$

Which is almost the same as the Euler method, however, the position now depends on the new velocity  $v_{i+1}$ . Unlike the Euler method, we now need to calculate the new velocity before calculating the new position. In the Euler method, we did not have to take this into account. In the C++ program, the ordering of the position and velocity calculation are swapped for these two methods, and we will see later that this will give two different results.

The next method we will look at is the Verlet method. It is given as

$$\begin{aligned}x_{i+1} &= x_i + v_i dt + \frac{dt^2}{2} a_i \\v_{i+1} &= v_i + \frac{dt}{2} (a_{i+1} + a_i)\end{aligned}$$

Implementing this may be a little tricky. Calculating the position is straight forward, but the velocity now depends on  $a_{i+1}$  and  $a_i$ . To bypass this problem, we will have to save  $a_i$  so its own variable and then calculate the new acceleration  $a_{i+1}$  using the new calculated position  $x_{i+1}$ .

### 3c) - Testing the algorithm

The initial velocity of Earth is retrieved by data from NASA.

One can assume that the total energy for the whole system is conserved as the total energy will be confined within this system. That is, we think of this system that is in a box. The energy for this system will not leave the box, thus the energy is conserved. We are also not taking into account of gravitational waves, which have some energy as well.

Another way to determine that the energy must be conserved is because the forces in this system are conservative.

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  $\Delta t$ . For the C++ program, we have tested  $\Delta t$  values of  $10^{-3}$  and  $10^{-4}$ . The number of mesh points used for each  $\Delta t$  values is  $N = 300000$  and  $N = 30000$  respectively. This is to ensure that the time elapsed, for both simulations, are the same.

Figure 1 shows a comparison of the Euler method with the given  $\Delta t$  values respectively. The system is a little more stable for a lower  $\Delta t$  value, but Earth seems to go closer and closer to the Sun for every circular orbit. This does not exactly fit with what we observe, so the Euler method is clearly not the most ideal algorithm we can use for these kinds of systems.

Let us now look at the Verlet method. Figure 2 compares the Verlet method of the given  $\Delta t$  values. As we can see, the system for  $\Delta t = 10^{-4}$  is incredibly stable, giving an almost perfect circular orbit for the Earth. The orbit is not perfectly circular is due to very small fluctuations in the orbit of Earth. For a smaller  $\Delta t$  value, the orbit is still circular, and unlike the Euler method, Earth will not fall towards the Sun. However, the fluctuations now are a little bigger, which is why the circular line in figure 2 is a little bit thicker.

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

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

One can also use the Euler Cromer method to solve these systems. A plot of the Earth-Sun system using the Euler Cromer method can be found in the appendix. 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.

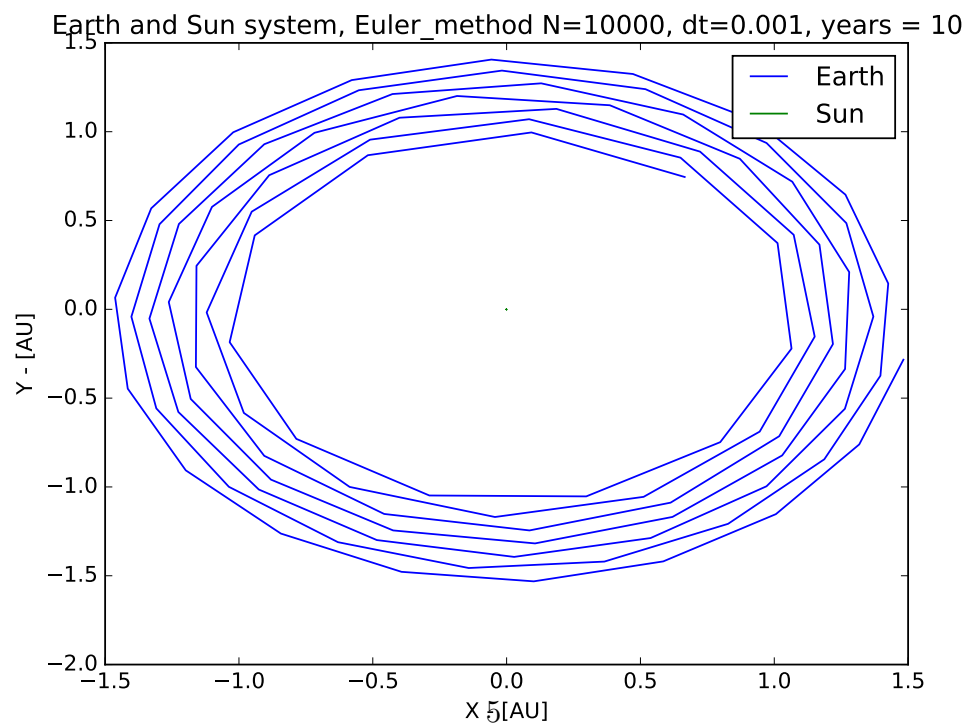
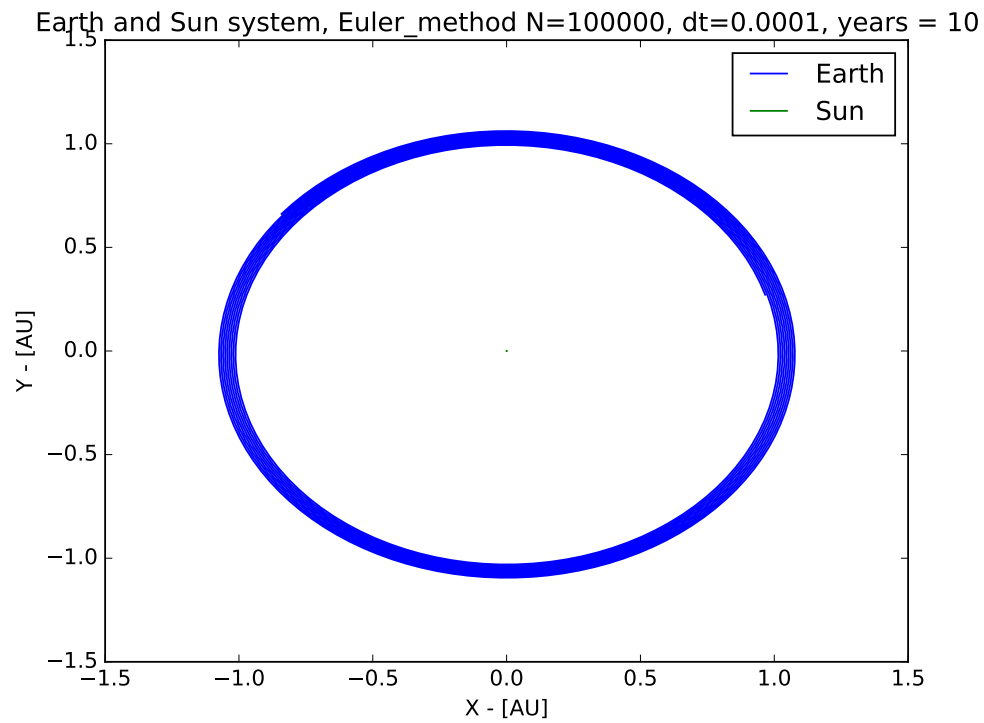


Figure 1: Euler method comparison for different  $\Delta t$

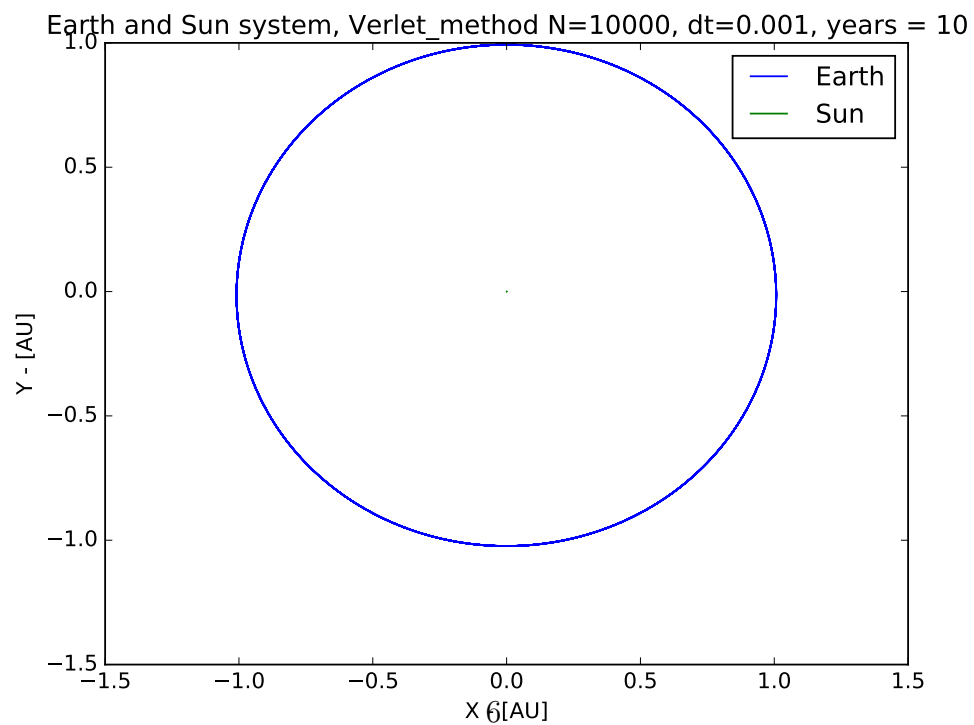
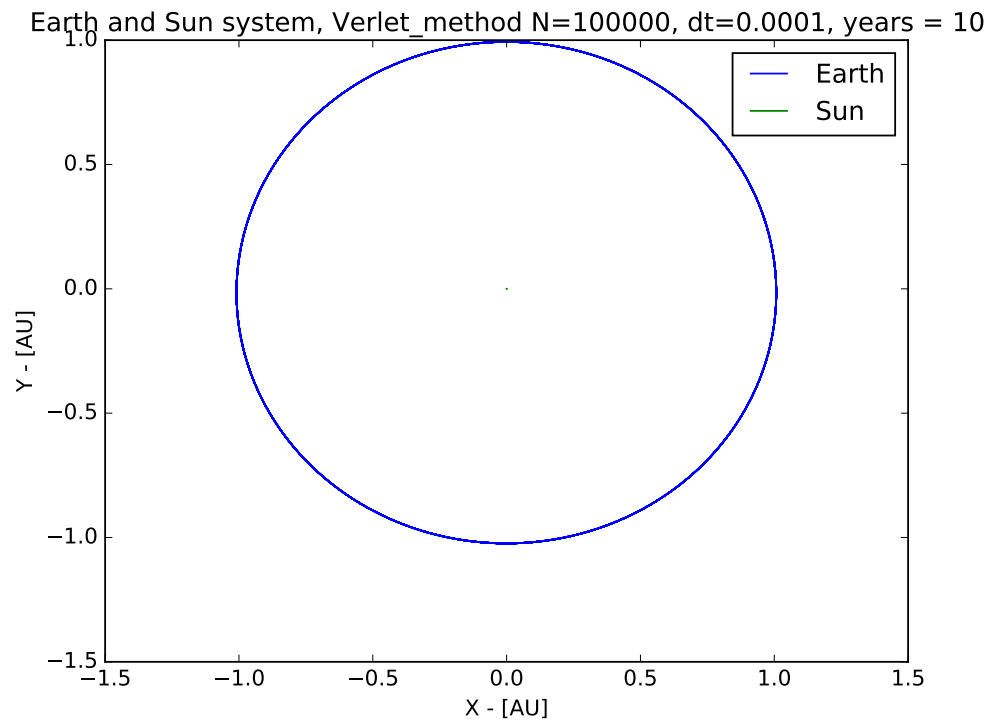


Figure 2: Verlet method comparison for different  $\Delta t$

### 3d) - 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

$$\frac{1}{2}M_1v = \frac{GM_1M_2}{r}$$

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 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 3 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, which was used previously, is also plotted as a comparison.

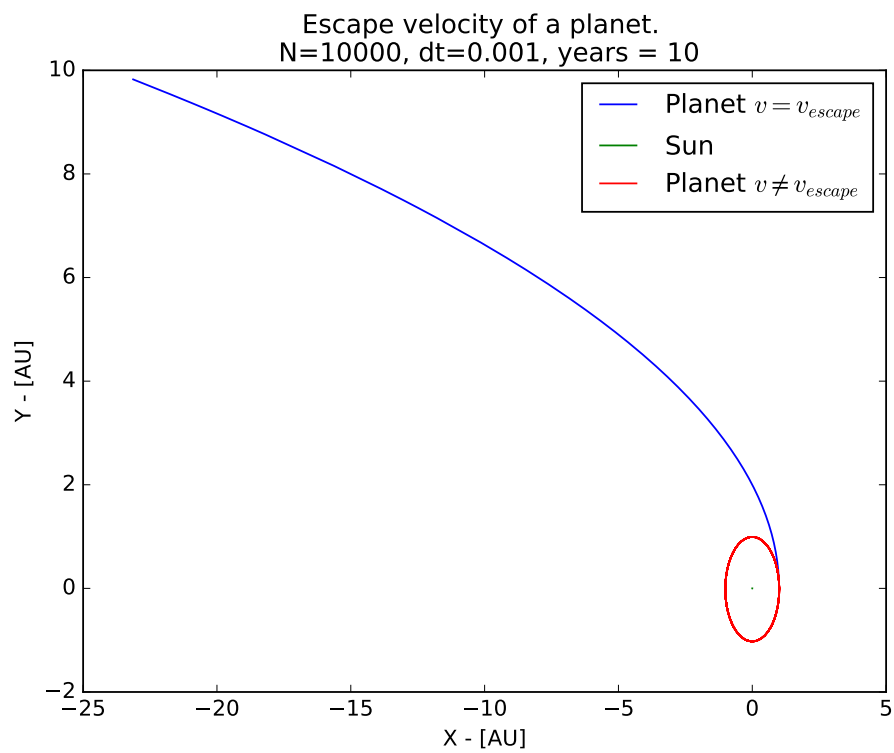


Figure 3: Plot shows a planet with the mass of Earth with and without escape velocity.



### 3e) - Three body problem

We already know that Earth's orbit will change, from the theory we previously introduced, when we add Jupiter to the system. Figure 4 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 5.

As we can see from figure 5, the orbit of the Earth has changed quite a bit compared to the system without Jupiter. Before we added Jupiter into the system, Earth's orbit was circular. Now that we have added Jupiter into the system, Earth's orbit is now elliptic, which is more realistic if we compare it to the orbit we observe. In addition to that, the position of the Sun is also affected a little bit. The Sun is no longer at rest, which means that the Sun is not the center of mass in the system.

Increasing the mass of Jupiter will change the orbits of Earth, the Sun, as well as Jupiter itself even more. In figure 6, the top image shows the system when 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.

The bottom image of figure 6 shows the system when the mass of Jupiter is increased by a factor of 1000. At this point, Jupiter has practically the same mass the Sun, so the gravitational effects from that should be quite dramatic. This is clearly seen from the plot, where the Sun, along with Earth, now moves away and will move away from it's initial position.

The Verlet method for both these cases are still quite stable, given that  $\Delta t = 10^{-4}$ .

3D Plots of this systems with unchanged Jupiter mass and 1000 times the Jupiter mass can be found in the appendix.

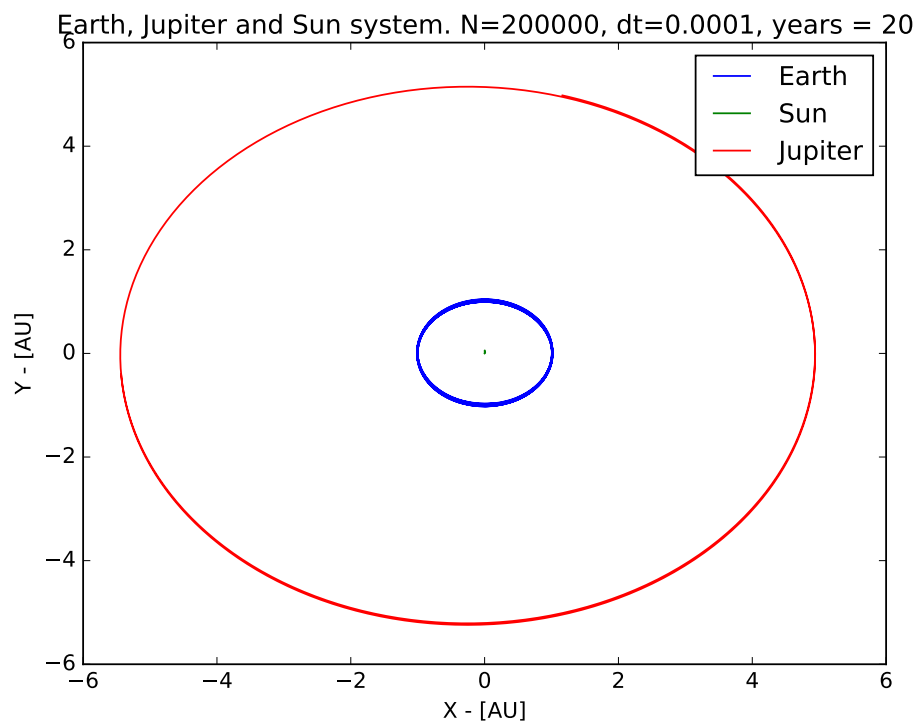


Figure 4: Orbits of the Earth, Jupiter and the Sun.

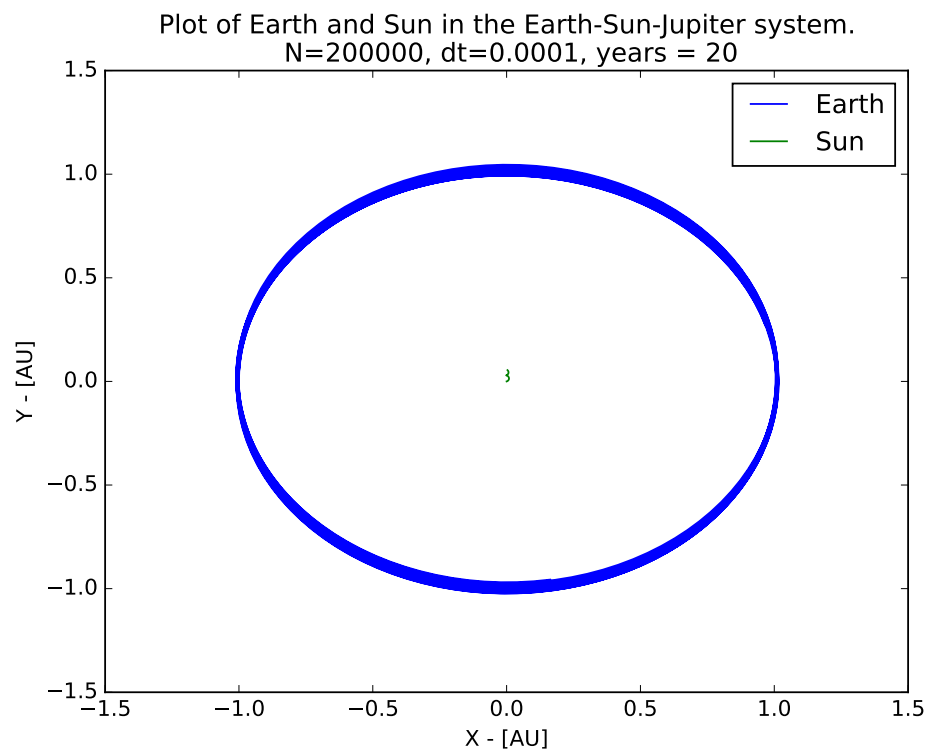


Figure 5: Plot of the Earth and the Sun in the Earth-Sun-Jupiter system.

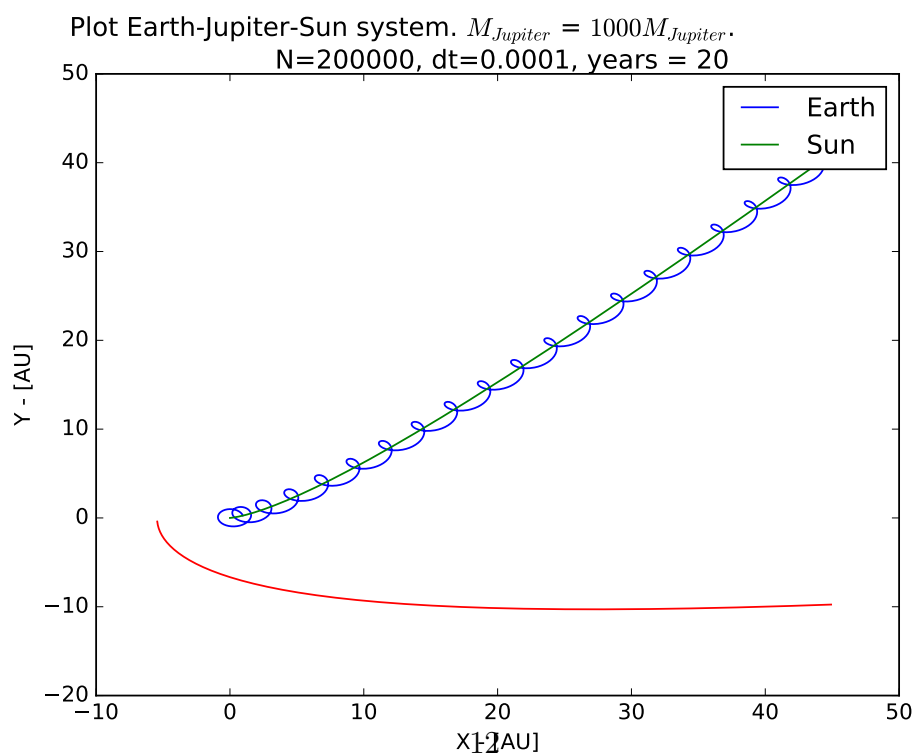
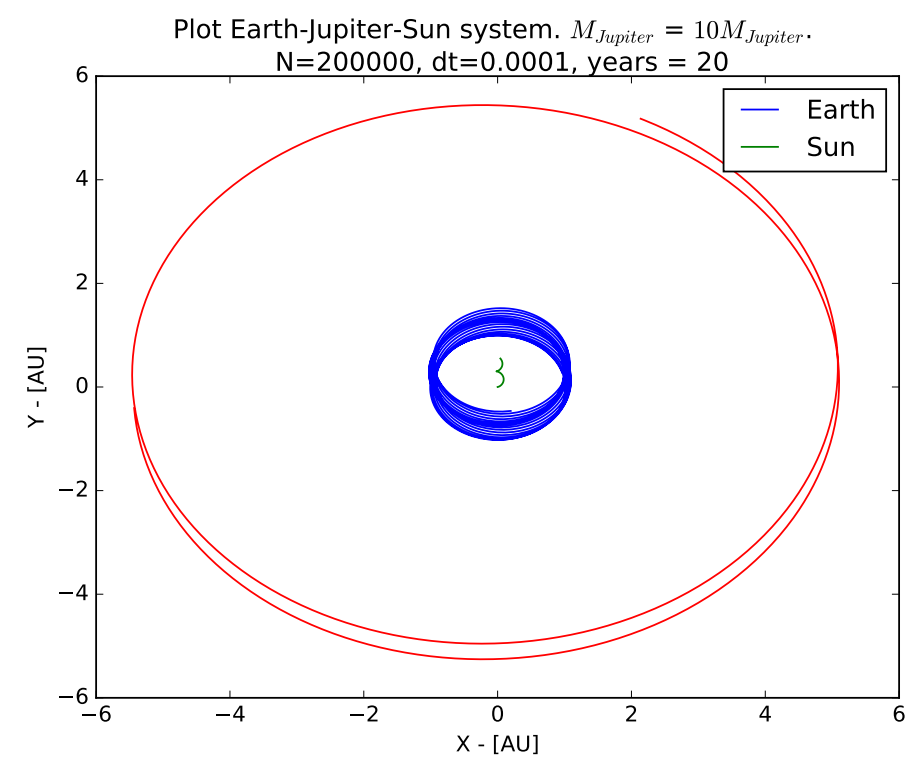


Figure 6: Earth-Sun-Jupiter system for  $M_J = 10M_J$  (top image) and  $M_J = 1000M_J$  (bottom image).

### 3f) - 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 7 shows the orbits of all planets in 3 dimensions. Because of the orbit sizes of the outer planets (i.e Jupiter, Saturn, Uranus, 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 reduce the value of  $\Delta t$  to simulate the first four planets, while still taking the effect of the gravitational forces of the other planets into account. This will of course reduce the number of years the simulation shows, which means that the outer planets will not have full orbits. The orbits of the first four planets under these settings can be found in figure 8.

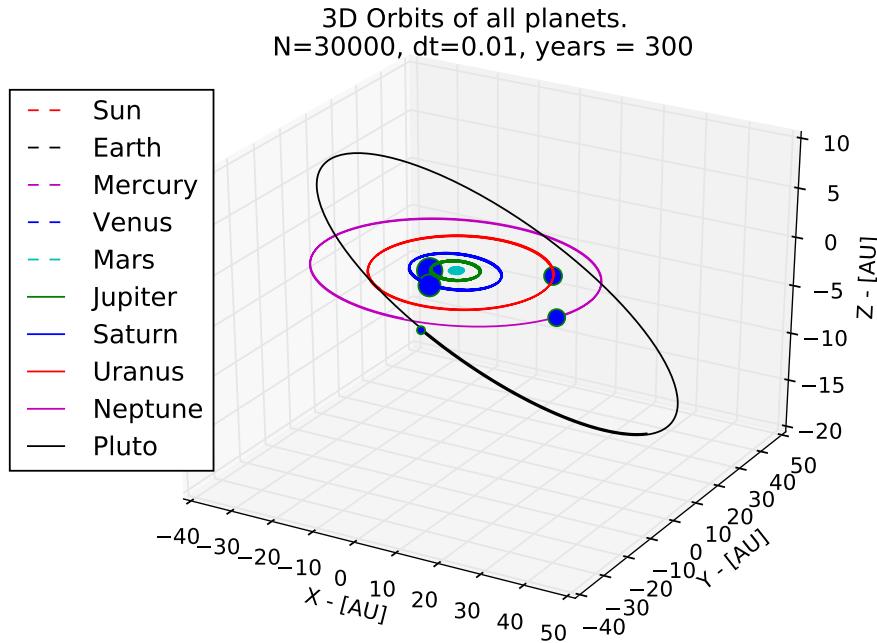


Figure 7: Plot of all planets in 3D. Note: the scatter plot (spheres) of the planet does not scale with it's real size.

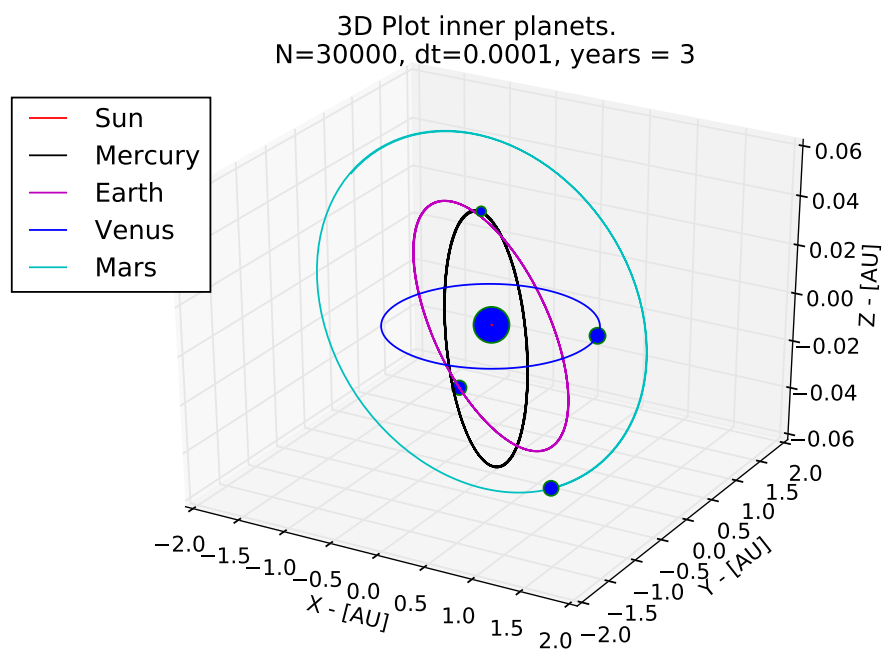


Figure 8: 3D plot of the first four planets orbit. Note: the scatter plot (spheres) of the planet does not scale with it's real size.

## Appendix

Earth and Sun system, Euler\_Cromer\_method N=100000, dt=0.0001, years = 1

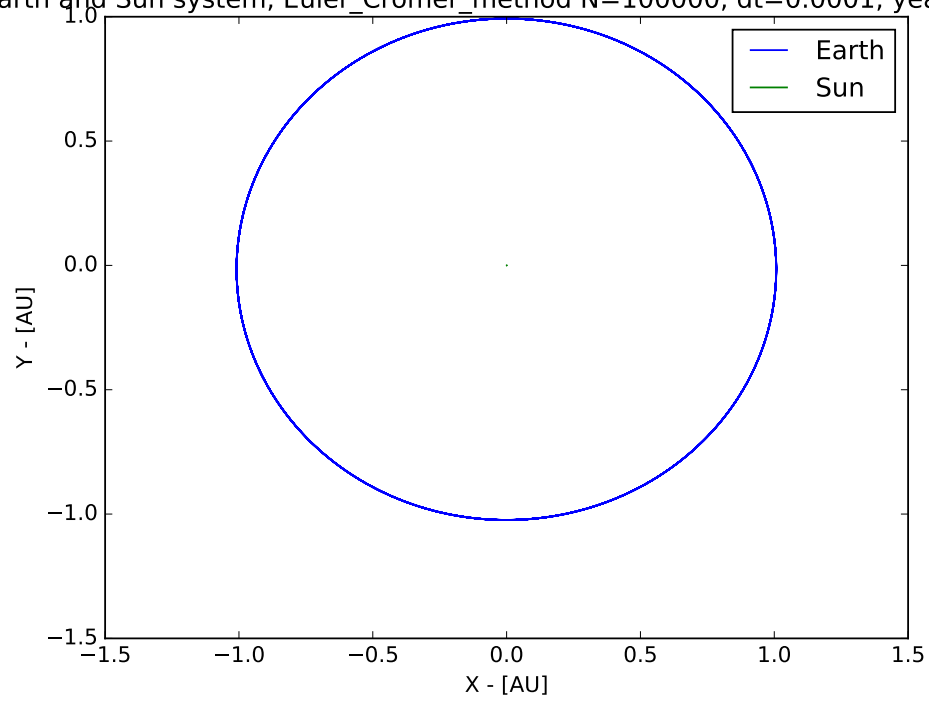


Figure 9: Earth-Sun system using Euler Cromer method

3D Orbits of the Earth-Jupiter-Sun system.  $N=200000$ ,  $dt=0.0001$ , years = 20

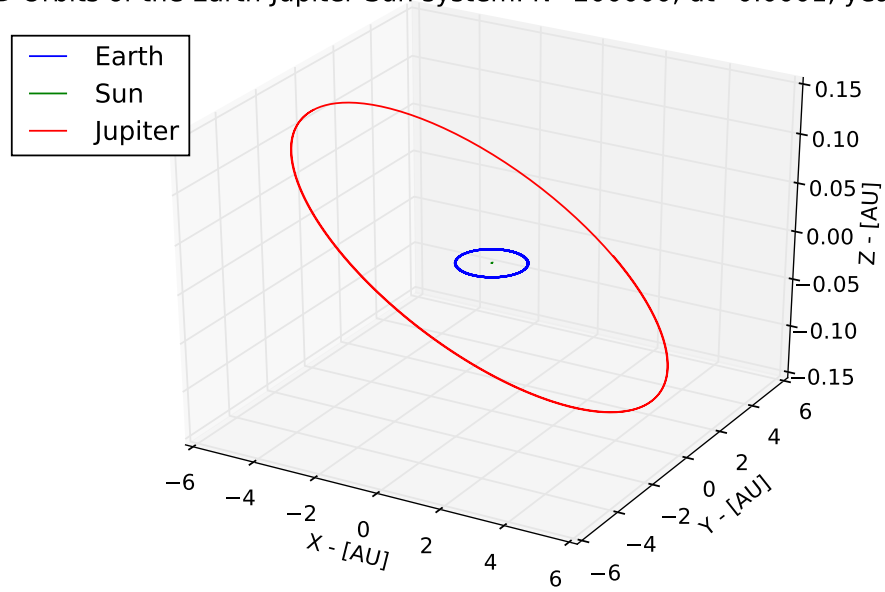


Figure 10: 3D Plot of the Earth-Sun-Jupiter system.



3D Orbits of the Earth-Jupiter-Sun system,  $M_{Jupiter} = 1000M_{Jupiter}$ .  
 $N=200000$ ,  $dt=0.0001$ , years = 20

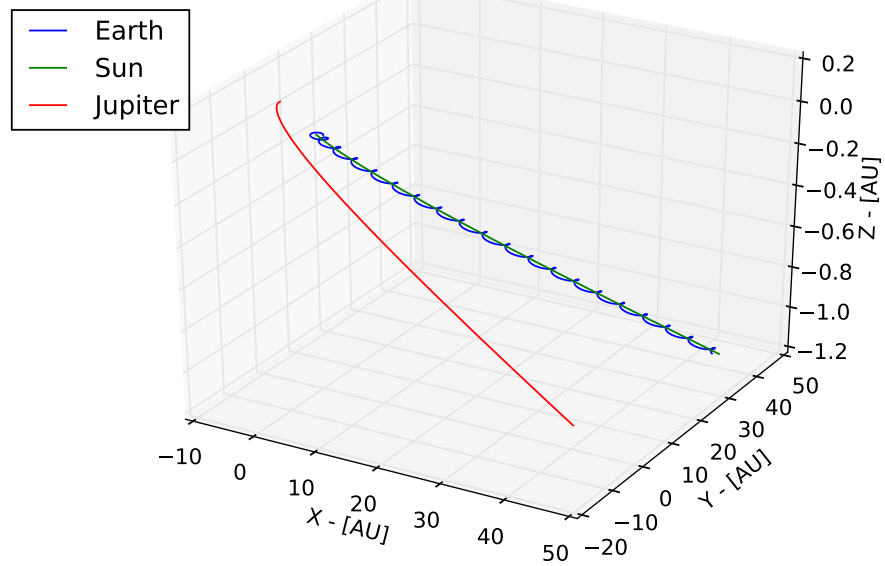


Figure 11: 3D plot for Earth-Sun-Jupiter system with the mass of Jupiter as  $1000M_{Jupiter}$