

FYS4150 - Computational Physics

Project 3

Alex Ho

October 20, 2016

Theory

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

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

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} .

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.
- **odesolver**s: 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 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 **odesolver**s. In addition to that, this class will save the positions of all objects to a file.

Unit test for energy and angular momentum 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. That is

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

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 $\epsilon = 3 \cdot 10^{-4}$.

Unit tests

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.

Results

3c) - Testing the algorithm

All initial conditions, i.e positions and velocities of both the Sun and Earth are obtained from NASA. The date of the data is 2016.10.05.

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.

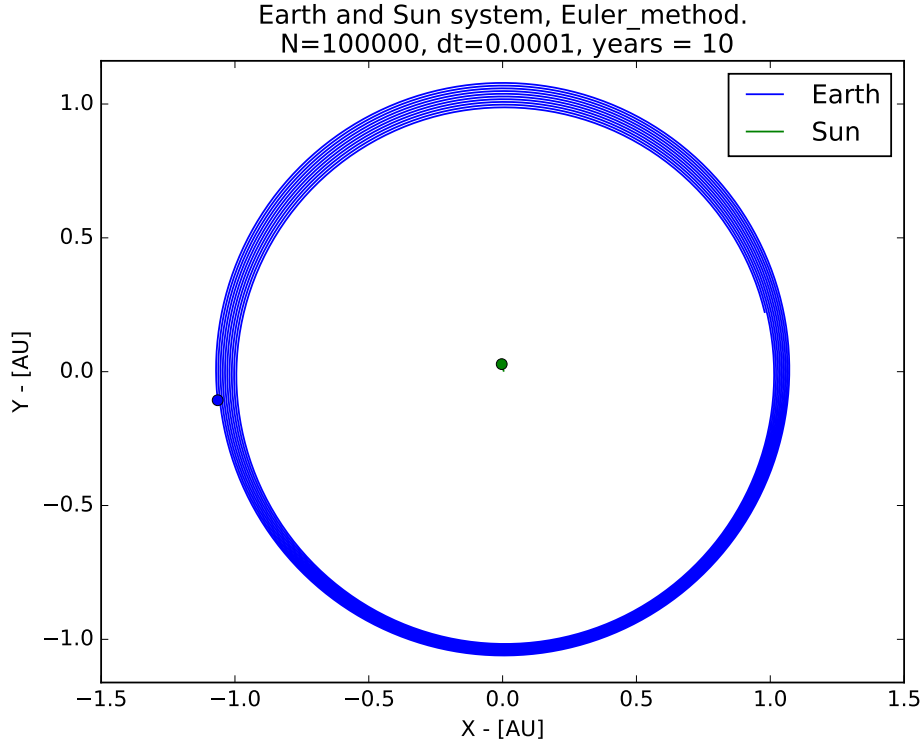


Figure 1: Euler method using $\Delta t = 10^{-4}$

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. In figure 4, we have increased the step length to $\Delta t = 10^{-3}$. However, the results appear to be identical, so for the Earth-Sun system, Verlet method is very stable up to $\Delta t = 10^{-3}$.

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

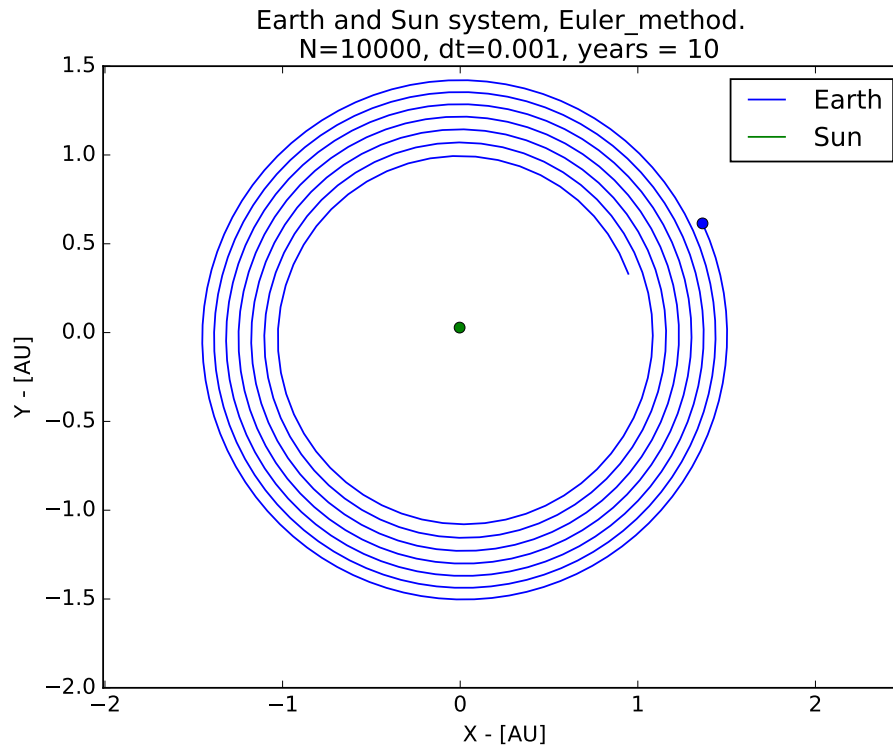


Figure 2: Euler method using $\Delta t = 10^{-3}$.

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
```

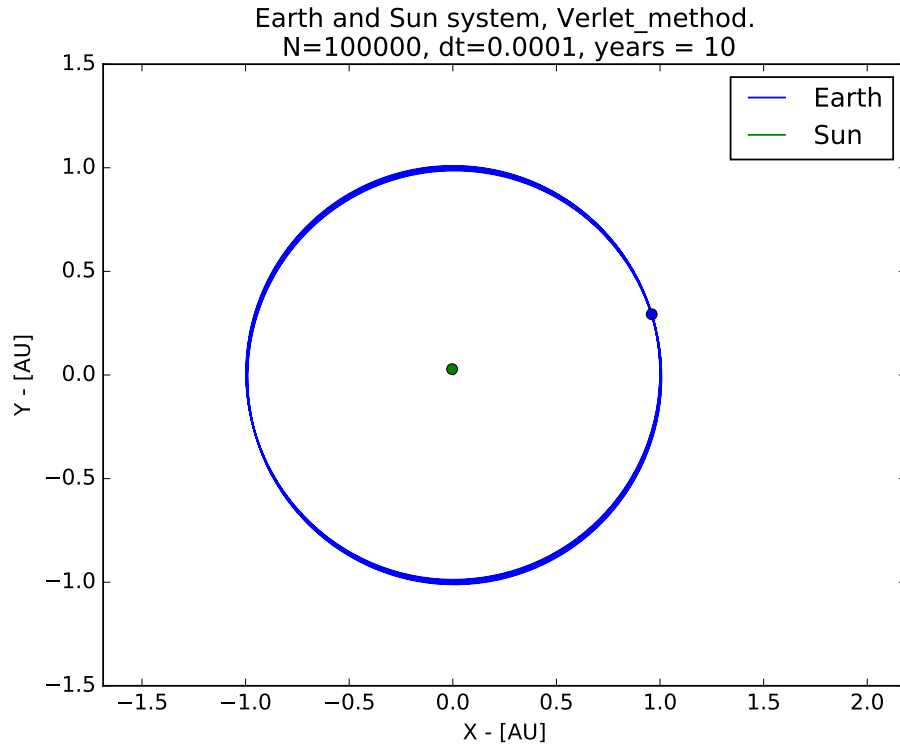


Figure 3: Verlet method using $\Delta t = 10^{-4}$

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

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 13 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.

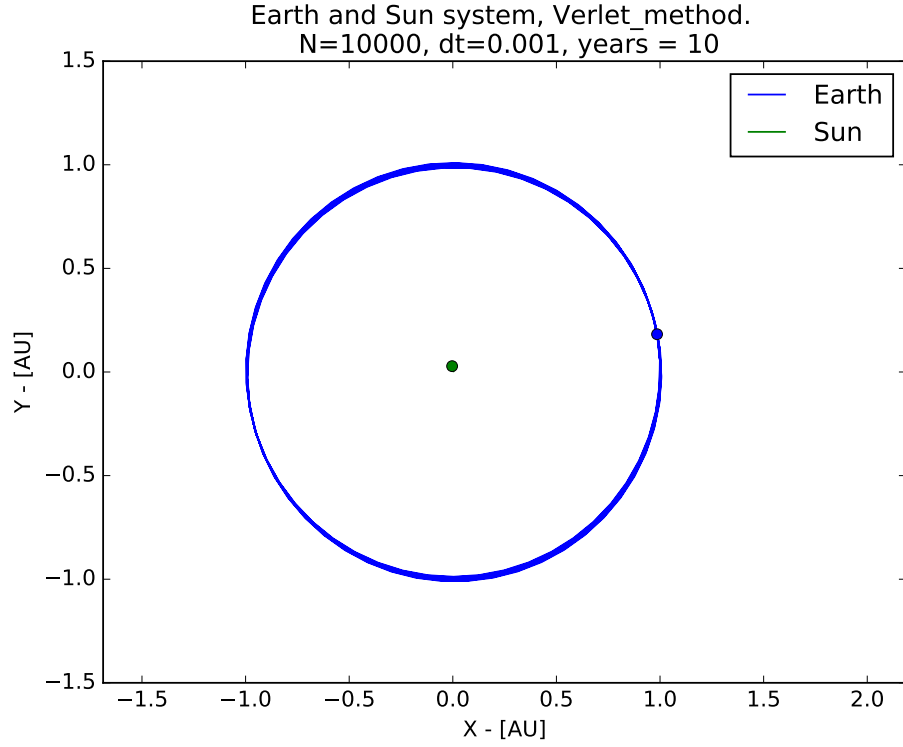


Figure 4: Verlet method using $\Delta t = 10^{-3}$

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

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

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 5 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.

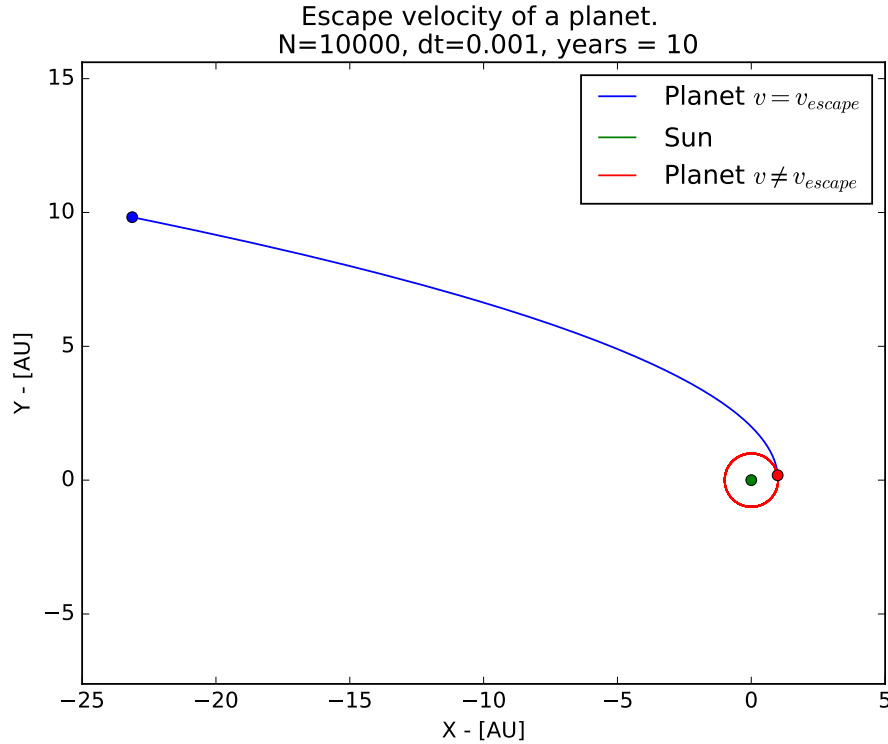


Figure 5: Plot shows a planet with the mass of Earth with and without escape velocity.

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 6 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 7.

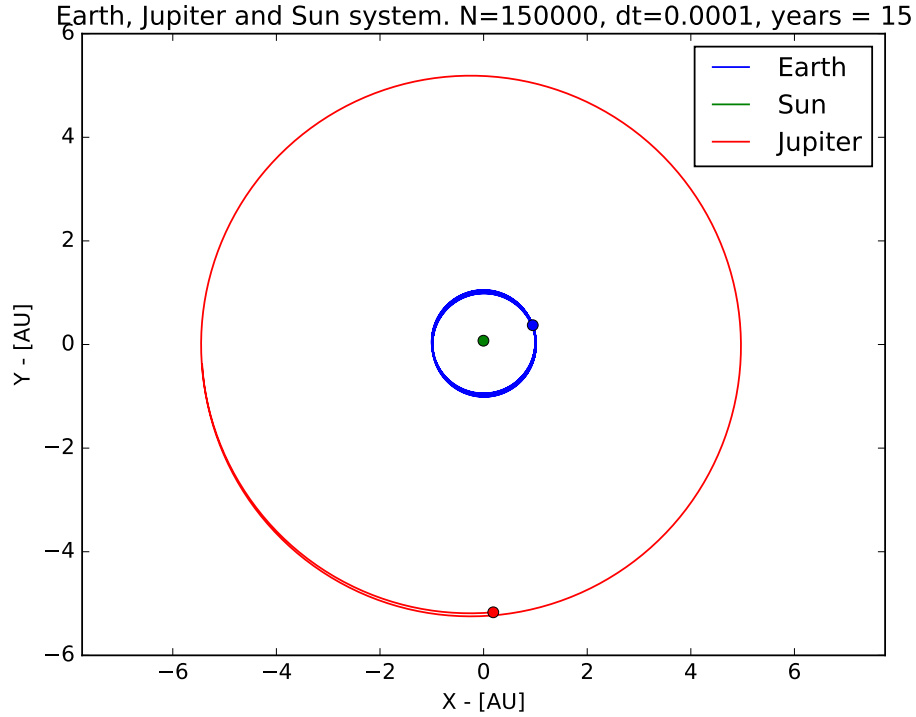


Figure 6: Orbits of the Earth, Jupiter and the Sun.

As we can see from figure 7, 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 slightly more elliptic, which is more realistic if we compare it to the orbit we observe. In addition to that, the position

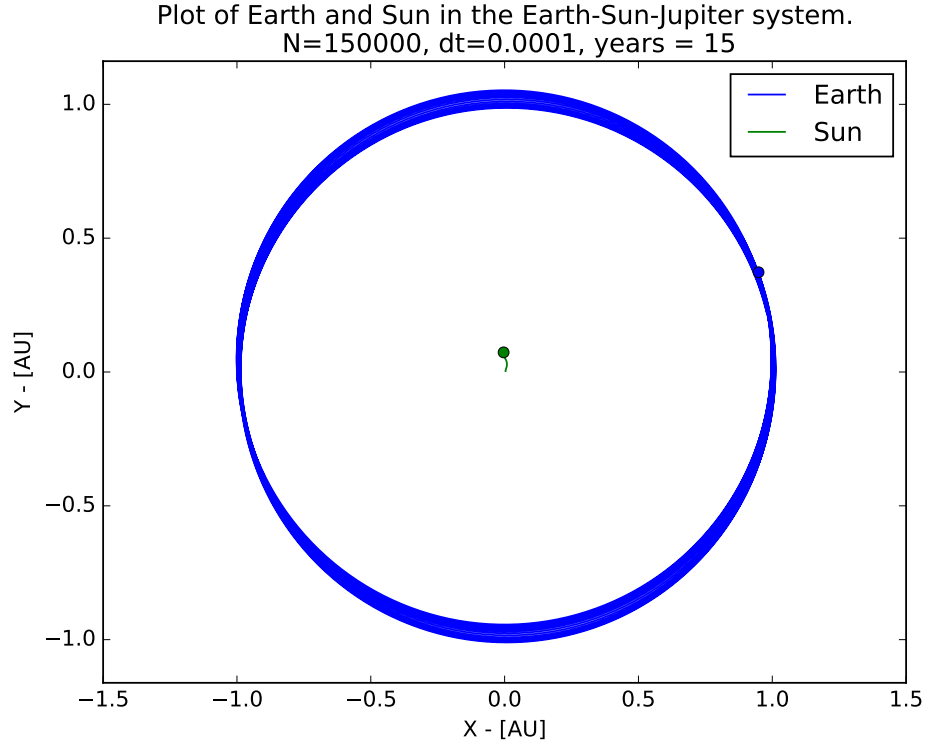


Figure 7: Plot of the Earth and the Sun in the Earth-Sun-Jupiter system.

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.

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 8, 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 9 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 10 shows the system when the mass of Jupiter is increased by

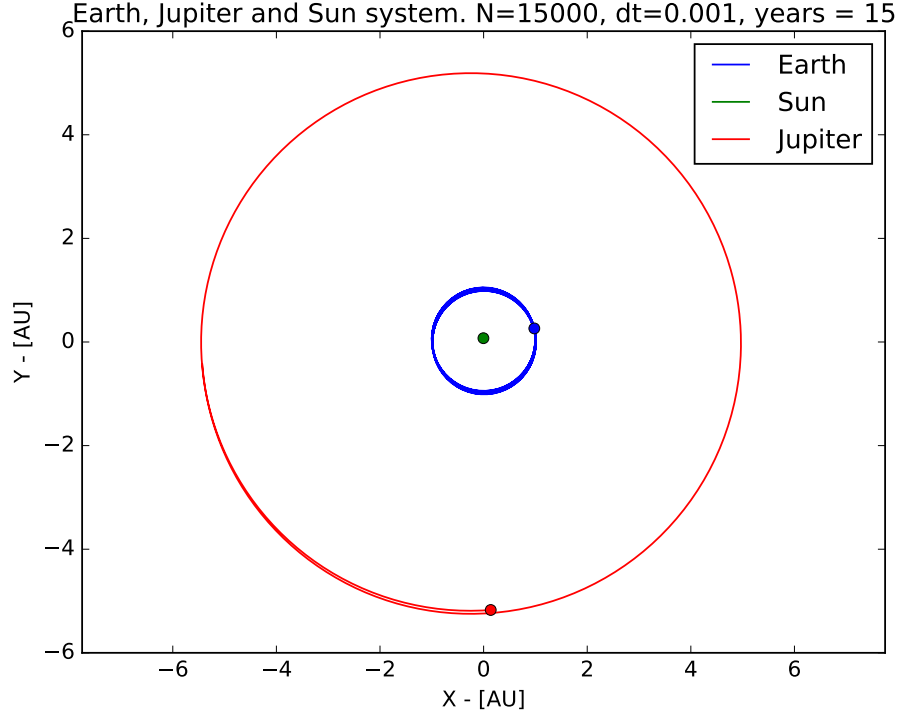


Figure 8: Earth-Sun-Jupiter system for $\Delta t = 10^{-3}$. The result is identical to the system with $\Delta t = 10^{-4}$.

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 its initial position. (To be adjusted)

3D Plots of this system with unchanged Jupiter mass and 1000 times the Jupiter mass can be found in the appendix in figure 14 and 15 respectively.

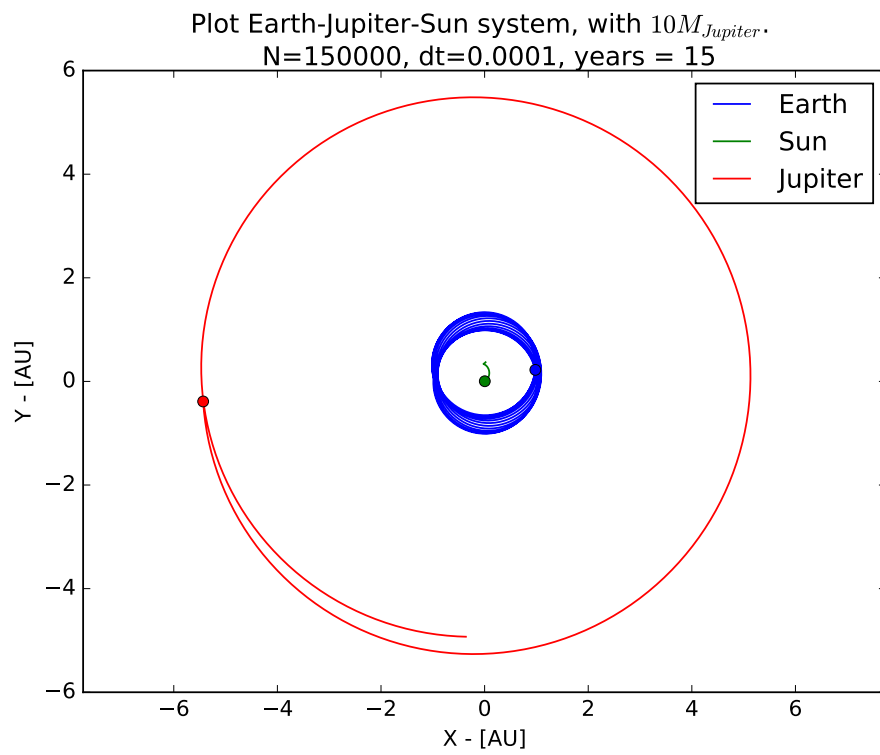


Figure 9: Earth-Sun-Jupiter system for $10M_J$.

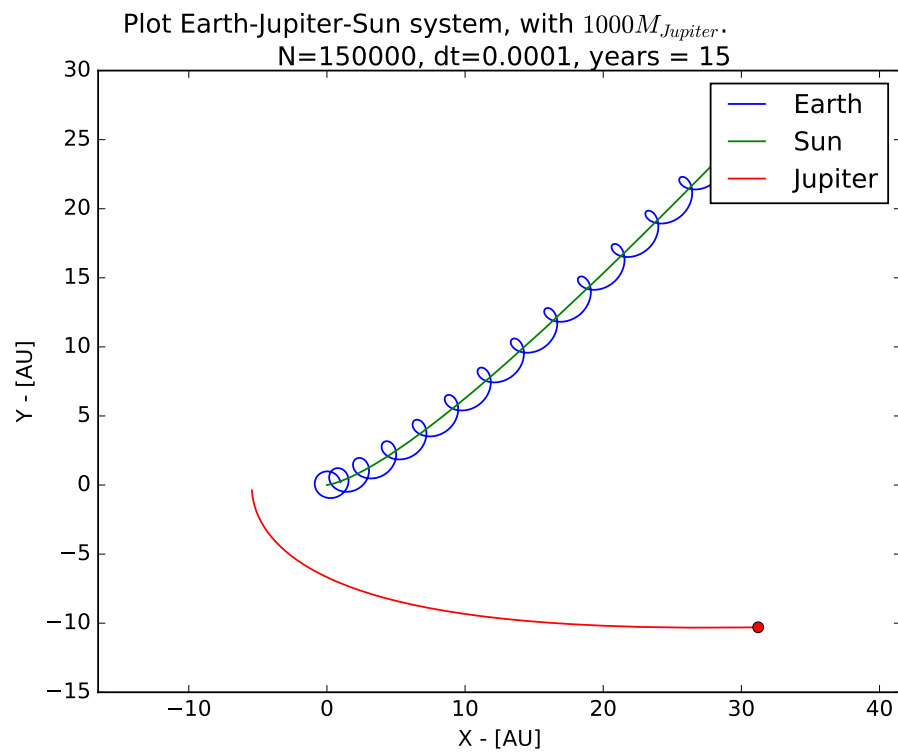


Figure 10: Earth-Sun-Jupiter system for $1000M_J$.

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 11 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

$$M_{\odot}v_{\odot} + \sum_i M_i v_i = 0 \quad (1)$$

Unlike the Earth-Sun-Jupiter system, the Verlet method is quite stable for $\Delta t = 10^{-3}$ in the system with all planets.

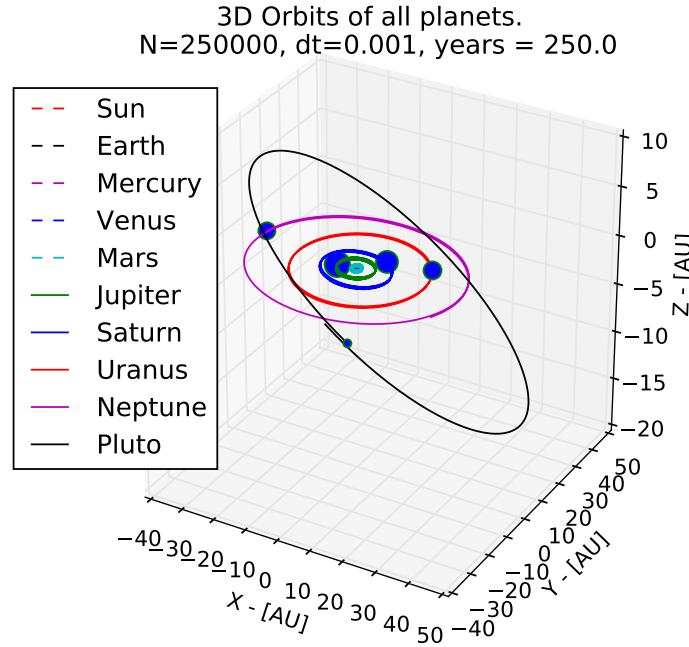


Figure 11: Plot of all planets in 3D. Note: the scatter plot (spheres) of the planet does not scale with it's real size.

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 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 12. One thing to note is that the outer planets will no longer have full orbits, like in figure 11. However, the effect from the gravitational pull of these planets are still present, which is the most important part.

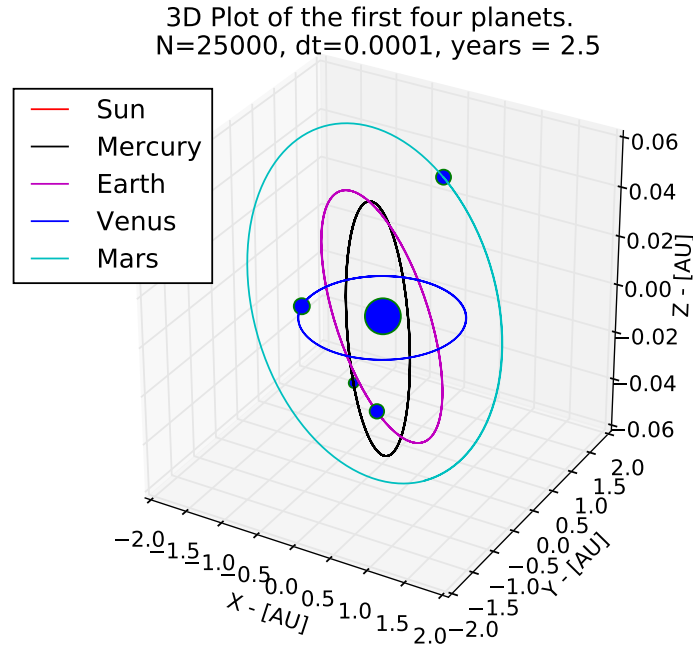


Figure 12: 3D plot of the first four planets orbit. Note: the scatter plot (spheres) of the planet does not scale with it's real size.

The results are promising. The orbits of all planets resembles to the orbits we observe. When looking at the plot for the first four planets, one

may quickly be confused on 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 $\pm 0.04AU$, which is a very very small change. By looking at figure 11, the vertical movement of the first four planets are practically zero, especially when we include Pluto to the system.

Appendix

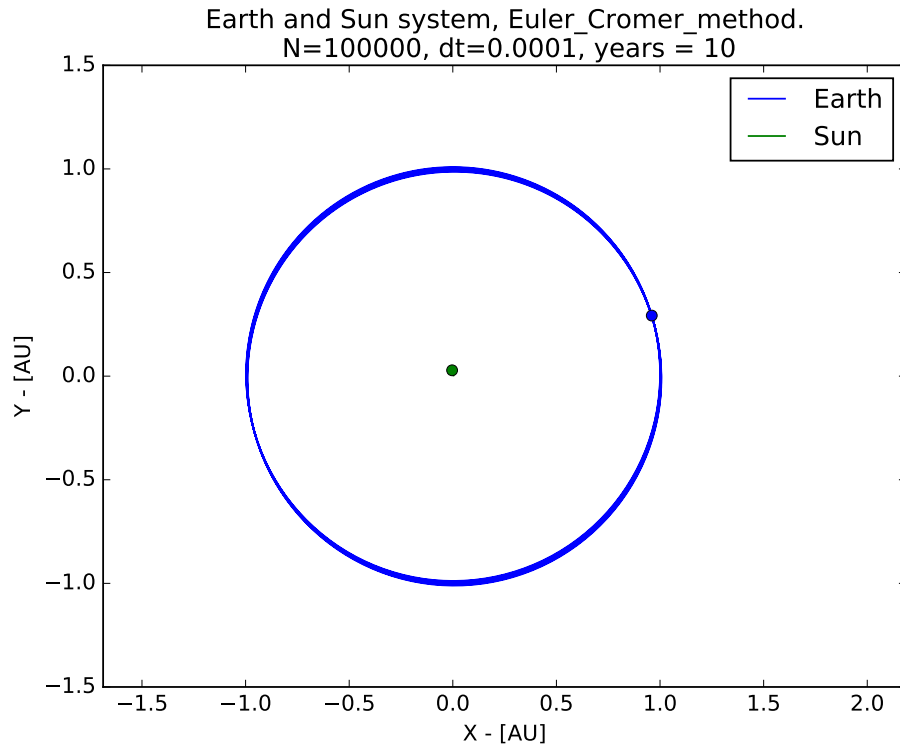


Figure 13: Earth-Sun system using Euler Cromer method

3D Orbits of the Earth-Jupiter-Sun system. $N=150000$, $dt=0.0001$, years = 15

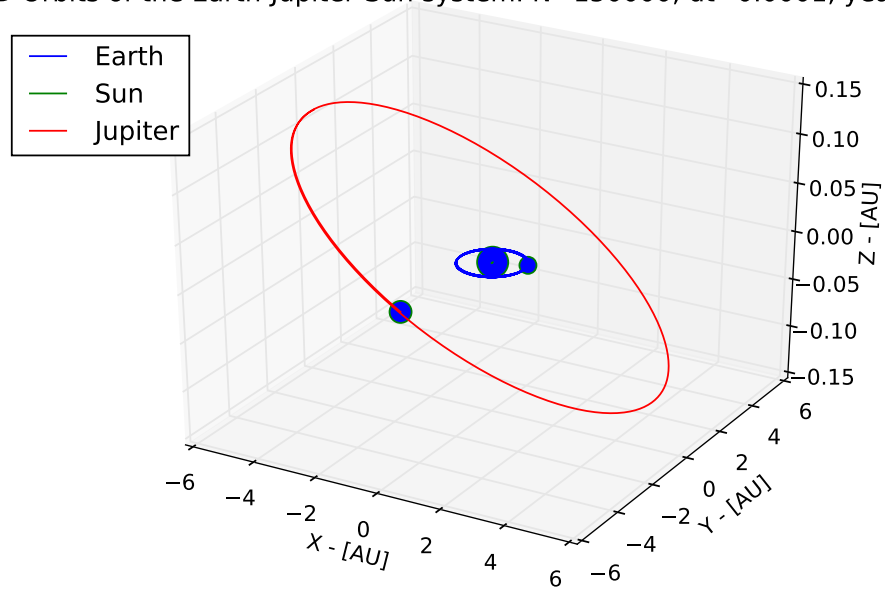


Figure 14: 3D Plot of the Earth-Sun-Jupiter system.

3D Orbits of the Earth-Jupiter-Sun system, with $1000M_{Jupiter}$.
 $N=150000$, $dt=0.0001$, years = 15

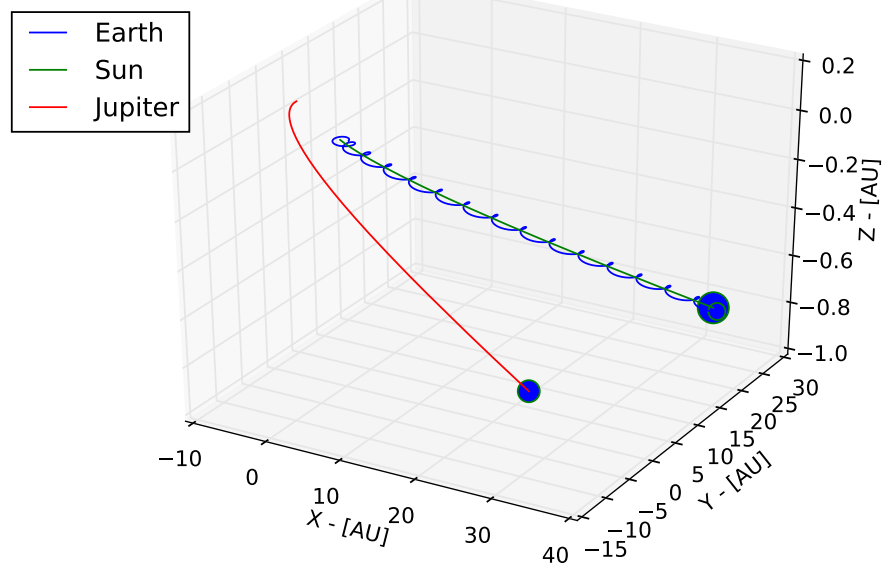


Figure 15: 3D plot for Earth-Sun-Jupiter system with the mass of Jupiter as $1000M_{Jupiter}$