Planetary Motion Simulator

Daniel Holmberg
014738452
daniel.holmberg@helsinki.fi
December 17, 2017

Contents

1 Introduction

This program, the Planetary motion simulator, calculates the motion of an arbitrary number of objects due to their gravitational forces towards each other. This has been achieved with the help of the Velocity Verlet algorithm. There was a few simulation tasks to be completed with the program, which are in the results section.

2 Methods

The simulation is based on the Velocity Verlet algorithm which iterates motion step by step. By first calculating accelerations, in this case due to the gravitational forces between all of the objects, and from there the new velocities which with a given timestep gives the new positions.

The steps looks like this:
(All equations holds for x, y and z directions by just swapping x for y or z)

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

$$a_{i+1} = F(x_{i+1})/m \tag{2}$$

$$v_{i+1} = v_i + \frac{1}{2}(a_i + a_(i+1)\Delta t \tag{3}$$

where the force F is given by

$$\mathbf{F} = G\frac{m_1 m_2}{r^3}\mathbf{r} \tag{4}$$

$$F(x_{i+1}) = G\frac{m_1 m_2}{(\sqrt{(x_1-x_2)^2+(y_1-y_2)^2+(z_1-z_2)^2})^3}(x_2 - x_1) \tag{5}$$

$$\Rightarrow a_{i+1} = G\frac{m_2}{(\sqrt{(x_1-x_2)^2+(y_1-y_2)^2+(z_1-z_2)^2})^3}(x_2 - x_1) \tag{6}$$

3 Implementation of the methods

The main part of the program is called main.f90, it uses a module called verlet.f90. I chose to make a main allocatable array p with all the objects' information excluding their acceleration wich has two allocatable arrays of its own. All the other attributes have their own allocatable arrays defined in the beginning of main, but I used them only for reading purposes. After that I iterate new values for the objects in p using the Velocity Verlet algorithm.

Apart from masses, positions and velocities in all three dimensions the program also needs the number of objects so that the arrays can be allocated correctly, a timestep for the algorithm, how many iterations and at what intervall the program should write to the output file and print progress to screen. All of this information must be in the input file, called "planets.txt".

This is how the input-file should be formatted:

```
[number of objects] [timestep in seconds] [iterations] [write/print interval]
[mass] [x pos.] [y pos.] [z pos.] [x velocity] [y velocity] [z velocity]
...
```

An example of what a file could look like:

```
2 1000000 3750 1
1.99e30 2.92024E+08 8.97213E+08 -1.85227E+07 -9.76602E+00 8.99910E+00 2.34305E-01
1898e24 -6.54642E+11 -4.80872E+11 1.66371E+10 7.58005E+03 -9.90845E+03 -1.28440E+02
```

To run the program the commands used are as follows:

```
gfortran -c verlet.f90 main.f90 && gfortran verlet.o main.o
./a.out
```

I have attached the input files I used for the upcoming tasks. Only one is named planets.txt but the naming is logical for the other input files, they are included in case one wishes to reproduce the results I got myself. I took my values from `https://ssd.jpl.nasa.gov/horizons.cgi` and the date for them are December 6th. The output file plot.txt and plot.xyz are also attached and they contain simulation values for the whole solar system. If they are to be used again and are in the same folder as the program file, before running, empty the plot files.

I visualized the data using matlab, planetmotion.m has produced all the plots and the only thing that needs to be changed depending on input is integer n which equals the number of objects. I did an animation with matlab, animation.m, which looks pretty neat (I only made it for the whole solarsystem). I recommend trying it as the attached plot.txt has suiting values.

Lastly I tried out Ovito, which uses the output file in xyz format, and made another animations of the solarsystem, animation.avi.

4 Results

In this section all the problems that were part of this project are answered.

4.1 Jupiter and Sun

I tried a timestep of 10 million seconds which can be seen in figure 1. This was obviosly too big as edges can be seen on the orbit. A ten times smaller step works, as can be seen in figure 2.

Figure 1: Jupiter orbit where timestep is too big

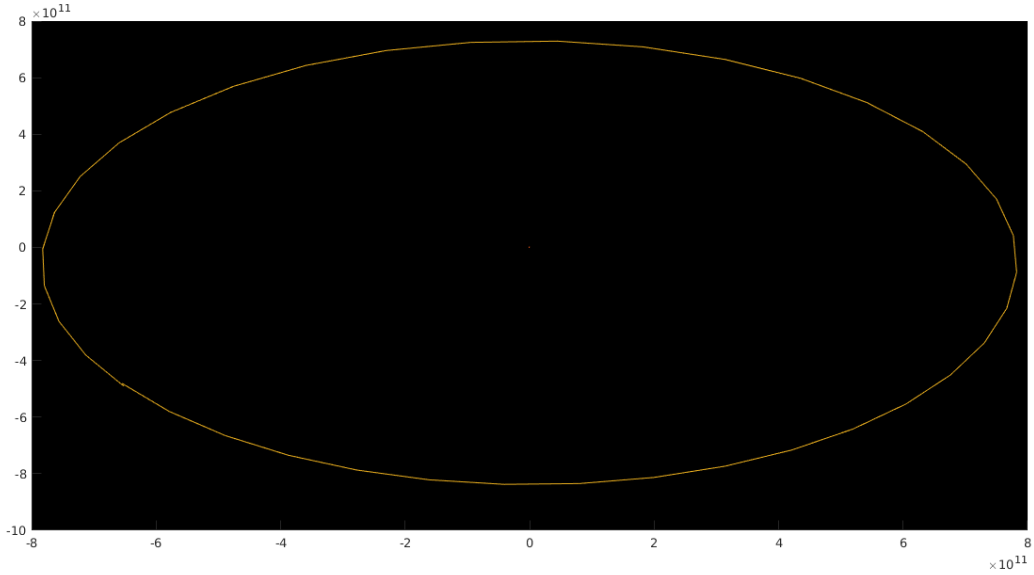
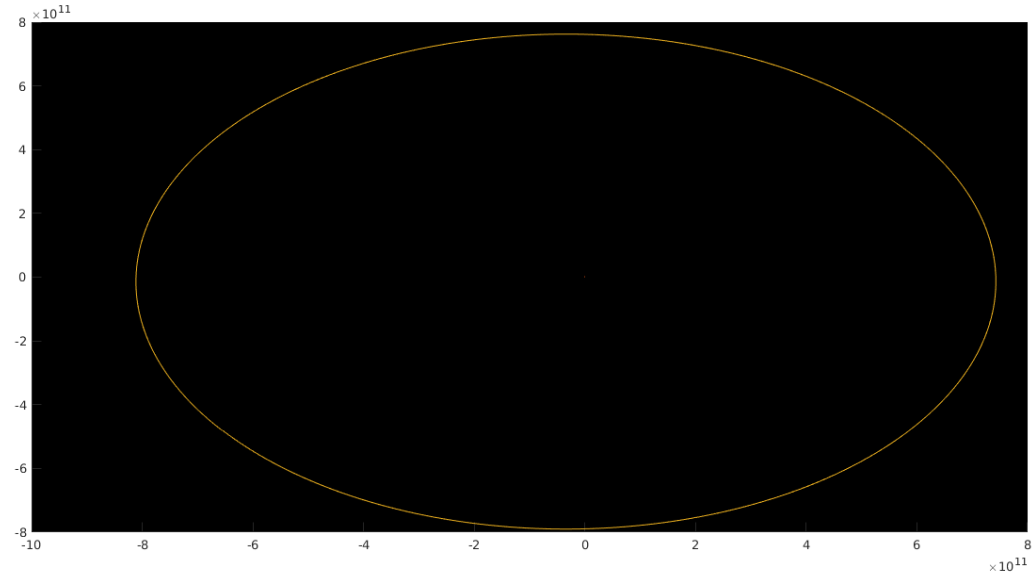


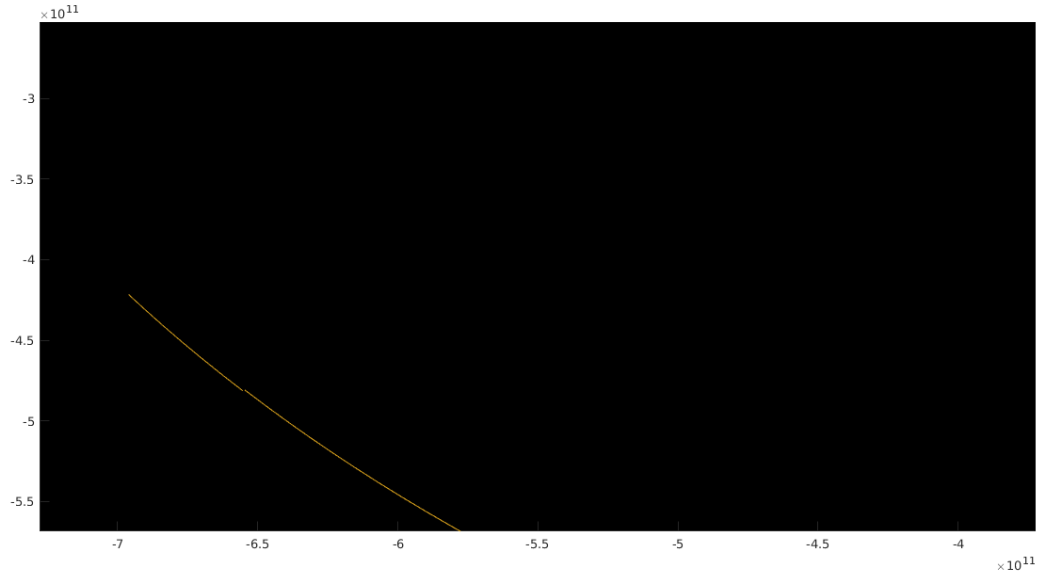
Figure 2: Jupiter orbit with appropriate timestep



I then wanted to see if it moves over or under one full period of motion. According to Nasa's fact sheet it takes Jupiter 4332.589 days which is approximately equal to 374.3 million seconds to orbit the Sun once. In the image 2 above I used a timestep of 1 million seconds and 375 iterations, which should be slightly more than one orbit.

The result can be seen in figure 3 below. So with such a big timestep it would be slightly under one orbit with 374 iterations. But all in all good results.

Figure 3: Very close to one orbit



I took a closer look at the Sun's motion after one orbit and this can be seen in figure 4, but the goal was to get its radius and period of motion, so I simulated ten orbits and got the result in figure 5. The diameter is roughly 10^9m so the radius becomes 0.5e9m. The period is the same as one orbit for Jupiter, so 375 million seconds or 4340 days.

Figure 4: The Suns motion in 3d

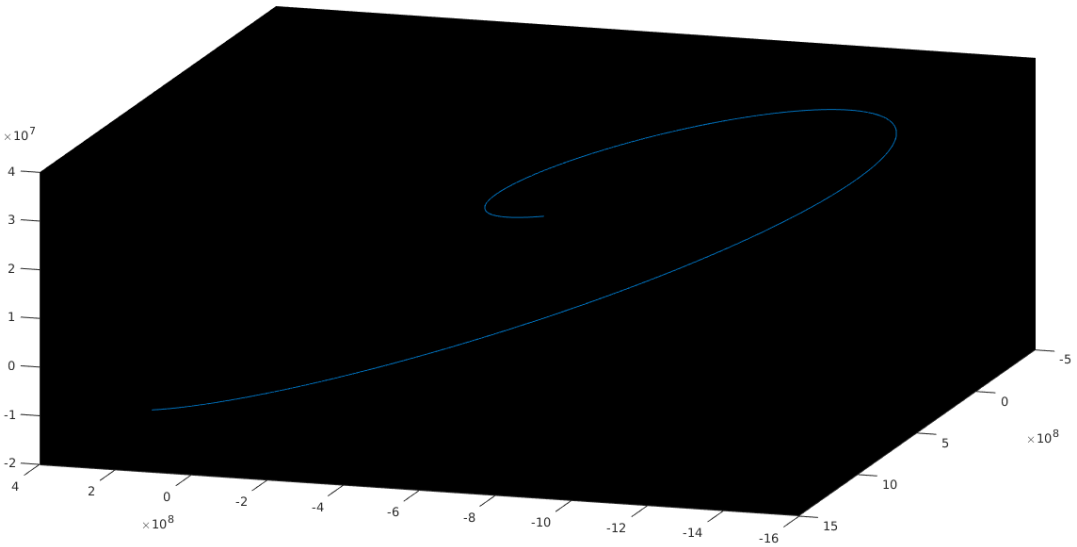
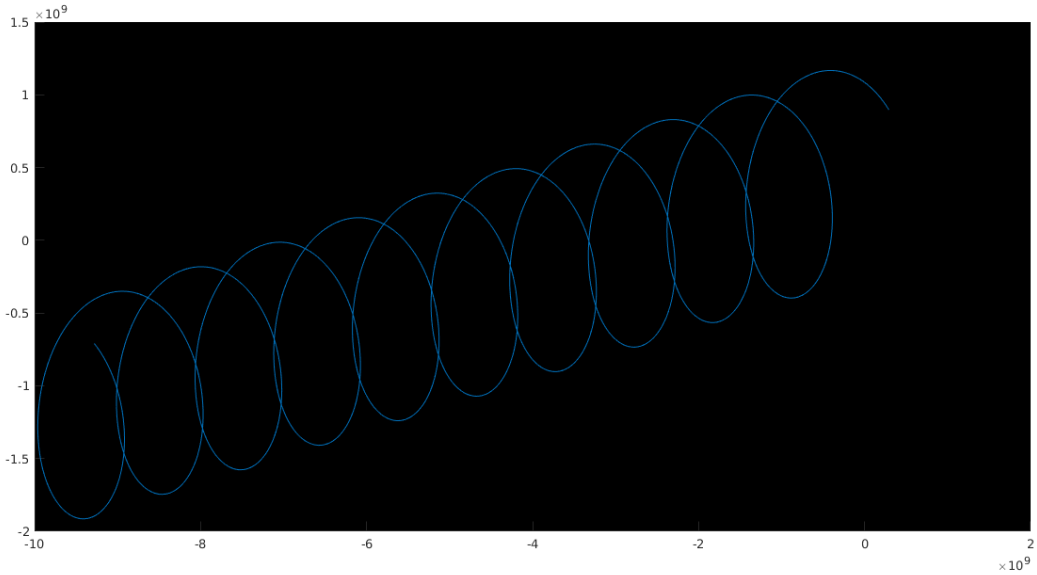


Figure 5: Ten cycles of the Suns motion



4.2 Simulated length of year and month

The following task was to simulate the length of one year and one month. For that I let the Earth orbit once around the Sun and the Moon once around the Earth.

The year can be seen in figure 6 and the timestep used was 10e5s and it took 315 iterations which totals 364.6 days, and according to the image it is slightly less than one year which is completely correct since the year is approximately 365.25 days.

The month simulation is displayed in figure 7 where i had the timestep set to 10e3s and the number of iterations were 232 which in turn makes the length of one orbit 26.85 days. The sidereal month (one orbit) is really 27.32 days.

Figure 6: One earth orbit ie one year

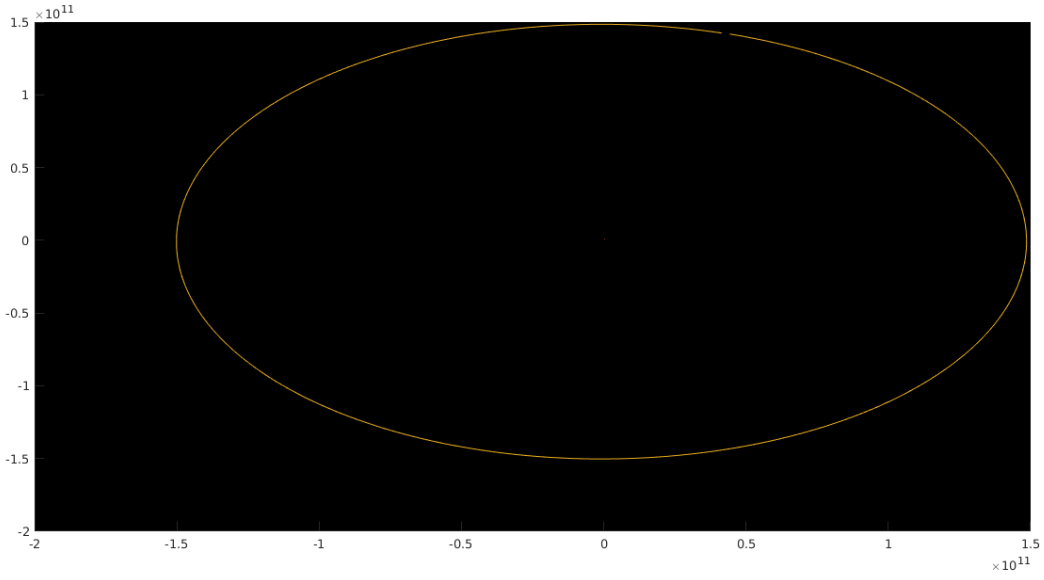
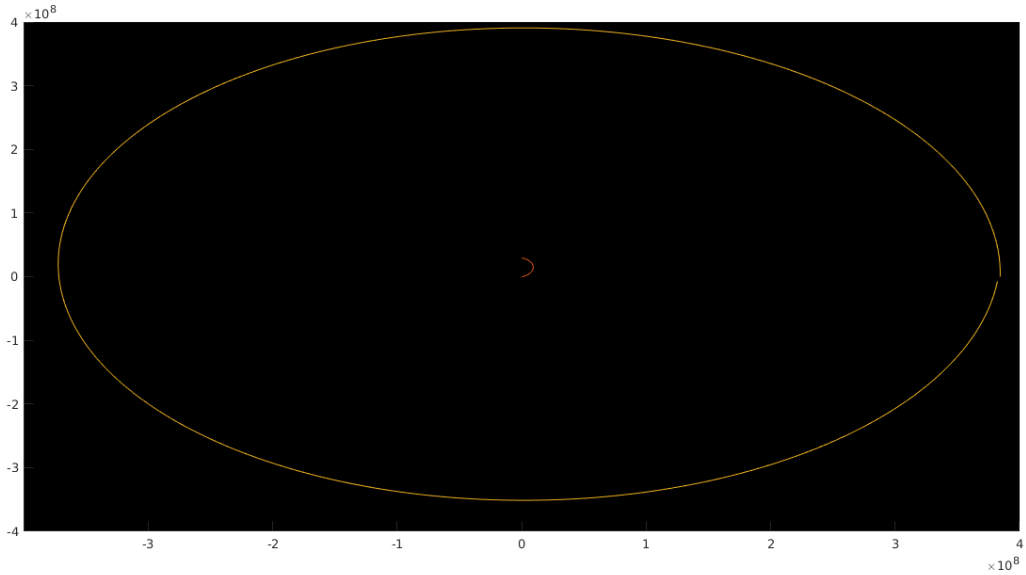


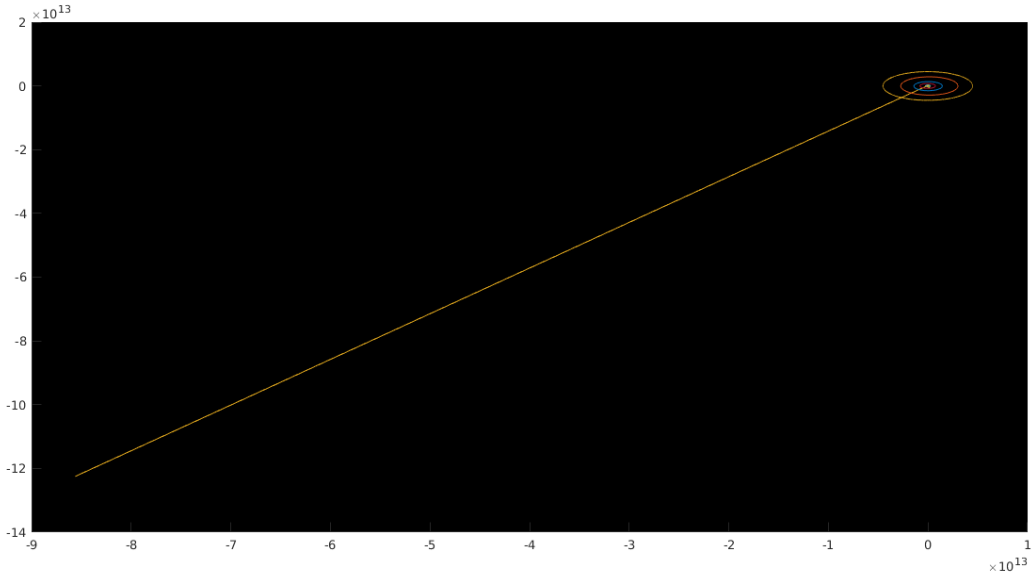
Figure 7: One moon orbit ie one month



4.3 Solar system simulation

Next up was to give the whole solar system a go, I first used the same timestep as I did with only Jupiter and the Sun. This gave an odd plot as can be seen in figure 8 below.

Figure 8: Solar System, timestep 10e6s



I zoomed in some more to see what was going on, and it wasn't apparaently a good idea to simulate the orbit of Mercury with such a big timestep (plot 9). So I made the timestep ten times smaller ie 10e5 seconds, figure 10, which worked just fine for plotting the whole system. It took 51980 iterations since it takes Neptune approximately 165 years to orbit once. And to end it, just for visualisation purposes I included picture 11 where all three dimensions of the orbits can be seen.

Figure 9: Timestep too big according to Mercury

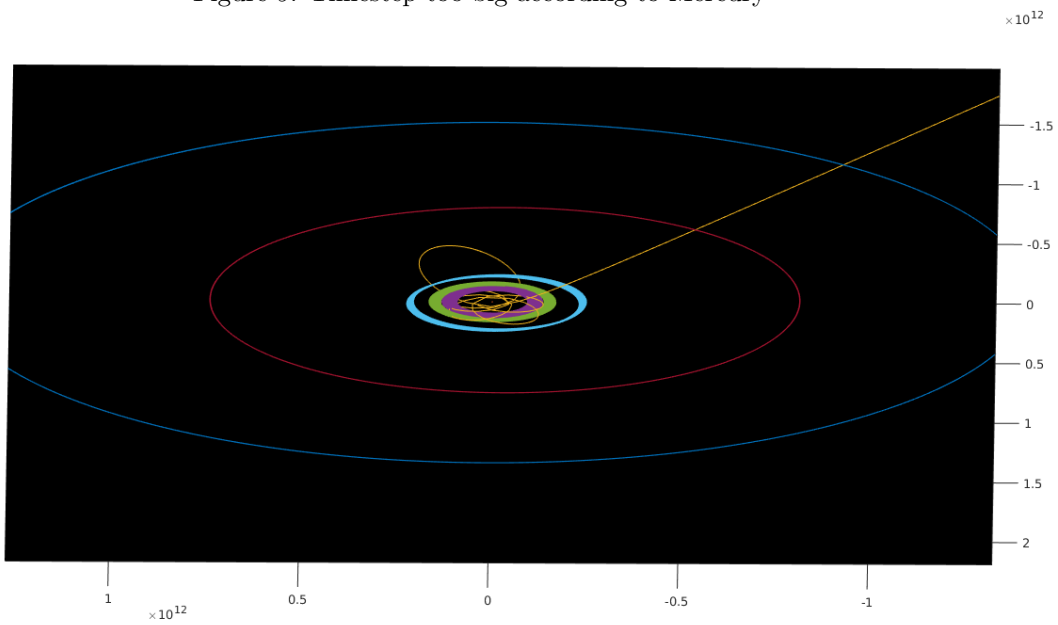


Figure 10: Solar System, timestep 10e5s

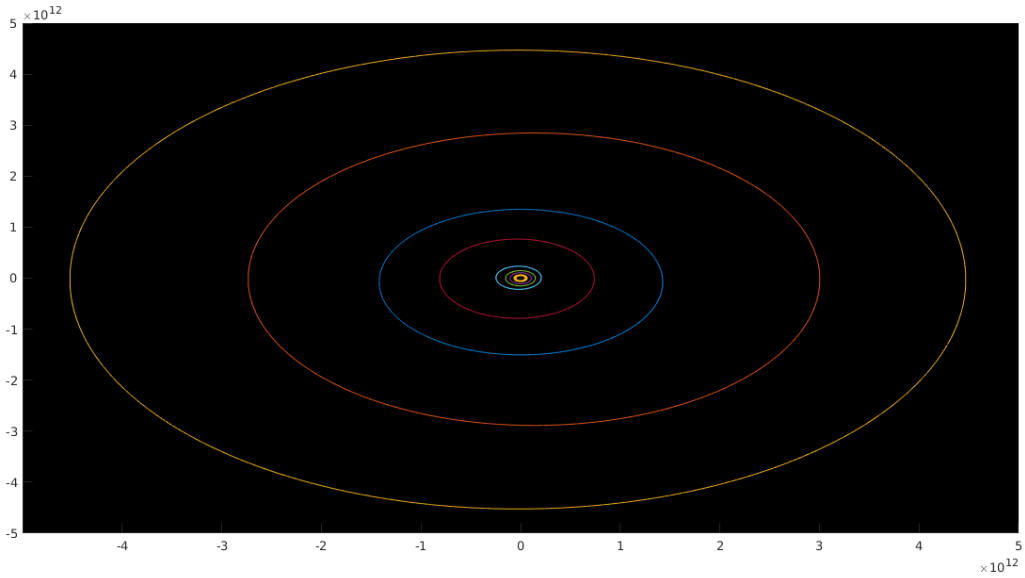
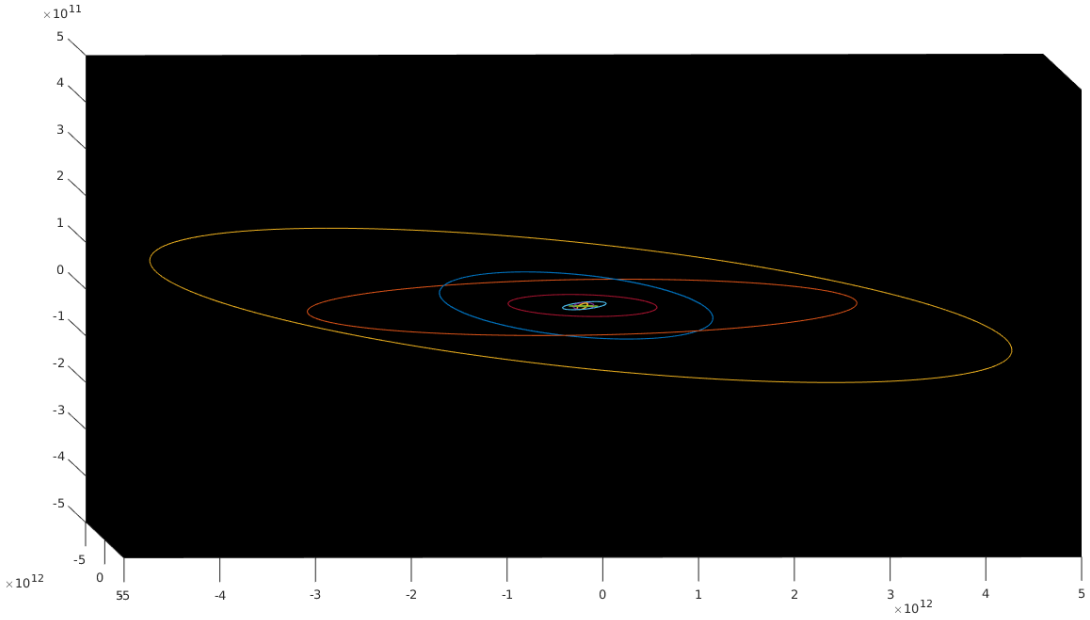


Figure 11: 3d Solar System



5 Conclusions

Overall the program handles all the tasks and works properly. The amount of data points needed to make a decent plot were surprisingly few and the timestep I could use and still get good results were surprisingly big. The program didn't take long to run either considering that there are many forces to be calculated between nine planets.