N-Body Simulation

Introduction

The N-Body problem simulates the evolution of a dynamic system of particles, under the force of usually, but not necessarily, gravity. These simulations are widely used in astrophysics, to mimic stellar dynamics in a globular cluster, the formation of the structure in the universe or the orbits of solar system bodies.

The aim of this project is to simulate an N-Body astrophysical system using Newtonian gravity:

$$F_g = G\frac{M_1 M_2}{r_{12}^2}$$

Where F is the force experienced of a particle, of mass M_1, by another particle, of mass M_2, where the particles are separated by distance R_{12}. Using this simple law, the dynamic evolution of an arbitrary number of particles can be simulated. The Visual Molecular Dynamics (VMD) program is implemented to visualise the trajectories of all particles. This project is written in java.

Object-Oriented Design

In this simulation, there are three java classes and four IO files. The following flow chart visualises the simplified interaction of all java classes and files.

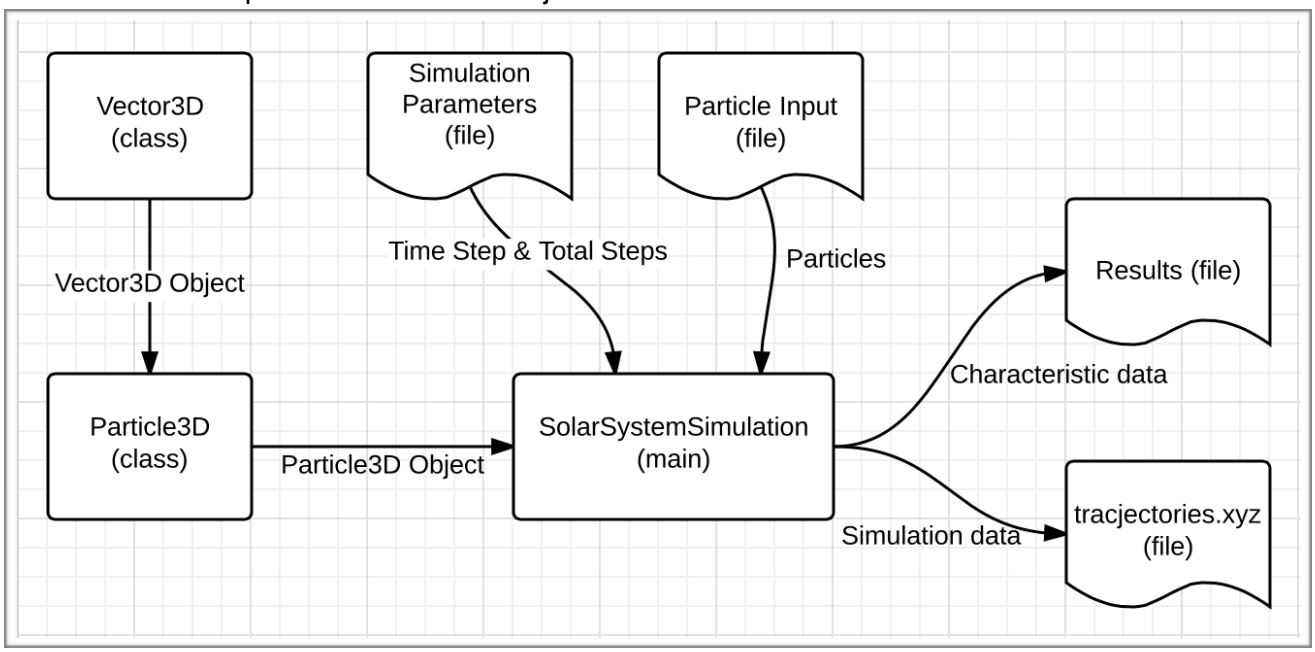


Figure 1 - Interaction chart of all classes and files

As shown in Figure 1, the Particle3D class makes use of the Vector3D object, supplied to by the Vector3D class. Then the SolarSystemSimulation class creates an array List of Particle3D objects. Next, the SolarSystemSimulation class simulates the interaction of particles with user defined inputs from the ‘Simulation Parameters’ file and the ‘Particle Input’ file. Finally, the characteristic data, such as the period, is written to the results file and the simulation data, all positions of all particles at all times, is outputted in a VMD friendly format.

Changes to design and algorithm:

1) No changes were made to the Vector3D class.

2) Particle3D new and altered methods:

Method	Parameters	Return Type	Description
updateTheta (New)	rrayList<Particle3D> particles, double[] xposition,double[] yposition, double[] changeTheta, int[] orbits	void	The updateTheta makes use of the previous and current x and y positions, to update the change in angle of a given particle in the x-y plane. See methodology section for a more detailed explanation.
updateClosestPoints (New)	ArrayList<Particle3D> particles, double[] closestPoints, int sunIndex	void	The updateClosestPoints method compares the stored and current distance of the given particle to the sun. It keeps the smaller of the two. See methodology section for a more detailed explanation.
updateFarthestPoints (New)	ArrayList<Particle3D> particles, double[] farthestPoints, int sunIndex	void	The updateFarthestPoints method compares the stored and current distance of the given particle to the sun. It keeps the larger of the two. See Methodology for a more detailed explanation.
resultantForce (Altered)	ArrayList<Particle3D> particles, Particle3D focusParticle	Vector3D	The resultantForce method returns the vector sum of all forces on a given ‘focus’ particle. Unlike the design document, this method does not use overloading. Note the additional parameter of a ‘focus’ particle.

3) Main class method

The main class very closely follows the guidelines of the design document. Besides the uneventful name change of the class, an extra output file was added to the parameters. This output file included characteristic data, such as the aphelion, perihelion, year lengths, number of full orbits, year ratios and energy ratios.

Methodology

Leap Frog Algorithm

In this simulation an arbitrary number of particles will be created and evolved in time using a leapfrog algorithm. This algorithm is more accurate over time when compared to the Euler algorithm. The Leap Frog algorithm works as shown below:

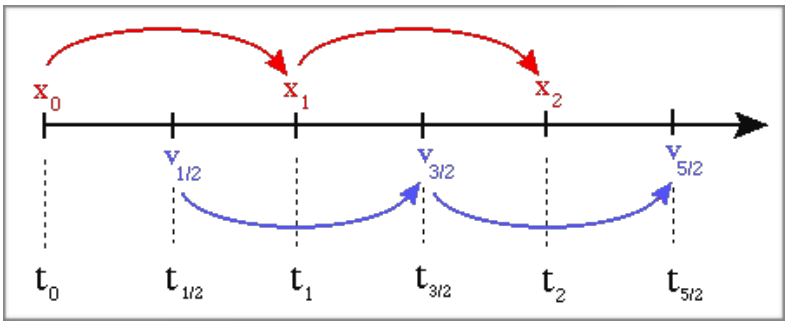


Figure 2: Leap Frog algorithm Schematic [1]

Figure 2 visualises how the leap frog algorithm works. First, the velocity is leapt forward by half a time step. Now, the loop of the program starts. The position is updated using:

$$\vec{x}_{i+1} = \vec{x}_i + \vec{v}_i dt$$

Next, the velocity is updated, using the following:

$$\vec{v}_{i+1} = \vec{v}_i + \frac{\vec{F}_i}{m_i} dt$$

Where F_i is updated using the most recent position. This loop is then repeated for a given amount of steps.

Circular and Elliptical Orbits

This simulation is conducted for circular and elliptical orbits in a x,y,z coordinate system. When simulating simple circular orbits, initial tangential velocities are calculated as the following:

$$v_c = \sqrt{\frac{G(m_1 + m_2)}{r_{12}}}$$

Where v_c is the tangential velocity, G is the gravitational constant, m_1 is the mass of the orbiting particle, m_2 is the mass of the stationary particle, and r_{12} is the separation between the particles. For simulating elliptical orbits, initial positions and velocities are extracted from NASA's JPL's HORIZONS system, which gives the current position and velocity of a given planet with respect to any major body in the x,y,z coordinate system [2]. The website also provided the masses of all planets, which is then converted to earth masses. Note, that the position and velocities in the z-axis are small for all planets, and significant for pluto and Halley's comet. This in itself sets the angle in the ecliptic for all particles.

Gravitational constant G:

Furthermore, it is appropriate to work in astronomical units (AU), Earth masses and Earth days. Obviously, making a change in units, means the gravitational constant G must be converted to this new set of units. The new value of G is:

$$G = 8.89 \cdot 10^{-10} \left[\mathrm{AU}^3 M_{Earth}^{-1} T_{Days}^{-2} \right]$$

Calculating the Initial Velocity of the Sun

The total system linear momentum needs to be zero. So, we can calculate the total linear momentum of all particles, except for the sun, and then set the initial velocity of the sun accordingly.

Calculating the Aphelion and Perihelion

First, store the initial distance of all planets to the sun in an array of doubles. Then for the next step, compare if the current distance is larger/smaller than the stored distance. In the event that this is the case, replace the stored distance with the current distance. Repeat for each time step. For the aphelion case, the current distance must be *larger* than the stored distance and for the perihelion case, the current distance must be *smaller* than the stored distance.

Calculating the Period

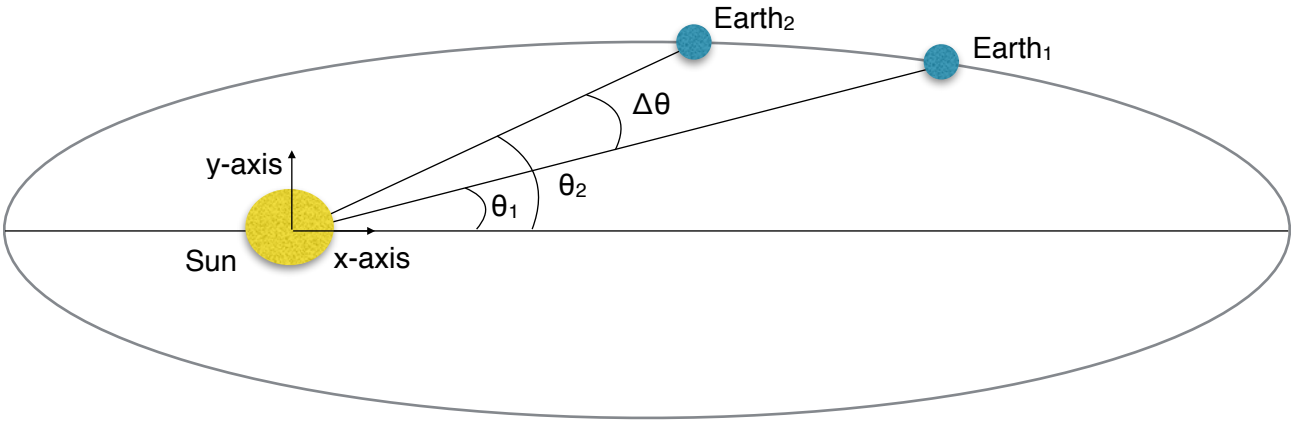


Figure 3 - Period calculation schematic

Calculating the period relies on the finding the change in angle in the x-y plane. For each step, the updateTheta method calculates the *current* angle of a particle to the sun in the x-y plane. It also calculates the *previous* angle of that particle to the sun in the x-y plane. Then the absolute difference between the *current* angle and the *previous* angle is stored as the change in theta. In every step, the change in theta increases by a small amount until it exceeds two pie. At this point, the orbit is increased for the given planet and the change in theta is reduced by two pie. The major drawback of this method is that the period can only be found for bodies which orbit in the x-y plane. However, since all bodies, including Halley's Comet and Pluto orbit in this plane, this is not an issue for this report.

Calculating the Year Length and Ratios

Once one period of a particle is completed, the current time is recorded and assigned as the year length in days. By simply dividing the year length of a given particle by the year length of the earth, we obtain the year ratios. It is appropriate to find the ratios in terms of earth years.

Calculating the year length, year ratios, aphelion and perihelion are used to evaluate the accuracy of the simulation.

Results

I. Circular Motion

A three body system consists of the Sun, Mercury and Venus, where realistic mass ratios and orbital radiuses were chosen, and converted into Earth masses and Astronomical units (AU), respectively [2]. The velocities of Mercury and Venus were calculated so that they perform a circular orbit, as explained in the methodology section. The parameters of this simulation are a time step of 0.1 (days) and 10.000 total steps.

Planet	Simulated Number of Orbits (days)	Real Number of Orbits (days)	Simulated Year Length (days)	Real Year Length (days)
Mercury	11	11.36	88.09	87.96
Venus	4	4.45	224.79	224.70

The real number of orbits and real year length are extracted from here [3]. Here is a screenshot:

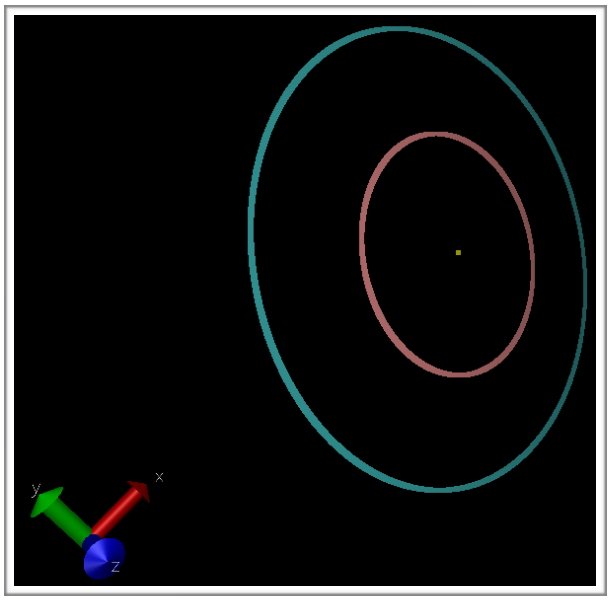


Figure 3 - 3 Body Circular Orbit

The ratio of the final to initial energy is 1.000653.

II. Elliptical Motion

The circular orbits are now changed to elliptical orbits by simple extracting the initial conditions from NASA's JPL's HORIZONS system. Once again, the time step is 0.1 (days) for 10.000 total steps.

Planet	Simulated Number of Orbits (days)	Real Number of Orbits (days)	Simulated Year Length (days)	Real Year Length (days)
Mercury	11	11.36	87.99	87.96
Venus	4	4.45	224.79	224.70

Note, the real values are the exact same as for the circular orbit. Here is a screenshot:

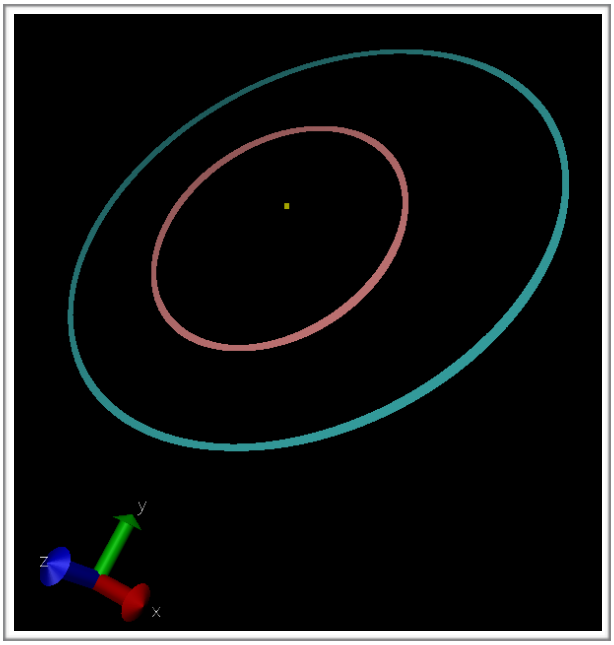


Figure 4 - 3 Body Elliptical Orbit

The ratio of the final to initial energy is 1.00725.

III. Solar System

All other planets and the moon are now added to the simulation. Once again, the initial conditions are extracted from NASA's JPL's HORIZONS system. The parameters of this simulation are a time step of 0.1 (days) and 1.000.000 total steps.

Planet	Simulated Year Ratio (no unit)	Real Year Ratio (no unit)	Simulated Year Length (days)	Real Year Length (days)	Simulated Perihelion (AU)	Real Perihelion (AU)	Simulated Aphelion (AU)	Real Aphelion (AU)
Mercury	0.2412	0.2408	88.10	87.97	0.3080	0.3074	0.4655	0.4667
Venus	0.6148	0.6152	224.6	224.7	0.7183	0.7184	0.7275	0.7282
Earth	1.000	1.000	365.3	365.23	0.9823	0.983	1.1017	1.017
Moon	0.07555	0.07480	27.60	27.32	N/A	N/A	N/A	N/A
Mars	1.876	1.881	685.4	687.0	1.381	1.381	1.663	1.666
Jupiter	11.89	11.86	4342	4333	4.958	4.950	5.463	5.458
Saturn	29.42	29.46	10746	10759	8.998	9.048	10.08	10.12
Uranus	83.73	84.01	30587	30686	18.19	18.28	20.13	20.10
Neptune	163.8	164.8	59832	60190	29.68	29.81	30.22	30.33

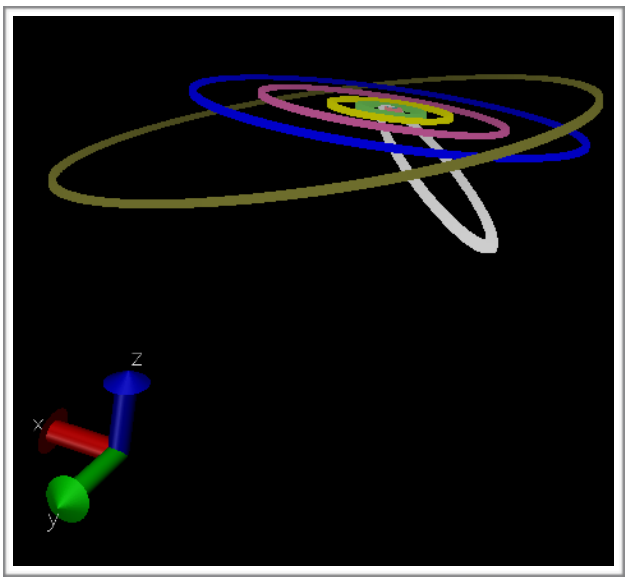
The ratio of the final to initial energy is 0.9973.

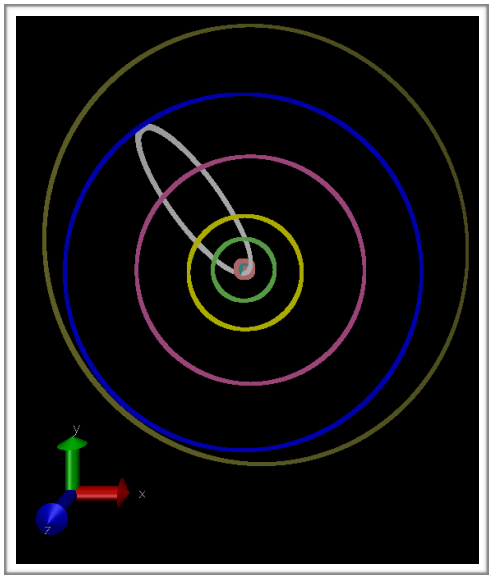
IV. Pluto and Halley's Comet

Finally, Pluto and Halley's Comet are added to the simulation.

Body	Simulated Year Length (days)	Real Year Length (days)	Simulated Perihelion (AU)	Real Perihelion (AU)	Simulated Aphelion (AU)	Real Aphelion (AU)
Pluto	88660	90465	29.44	29.66	48.23	48.87
Halley's Comet	27512	27503	0.575	0.586	35.2	35.1

Here are some screenshots of all bodies:





Figures 5 & 6 - Elliptical Trajectories of All Bodies

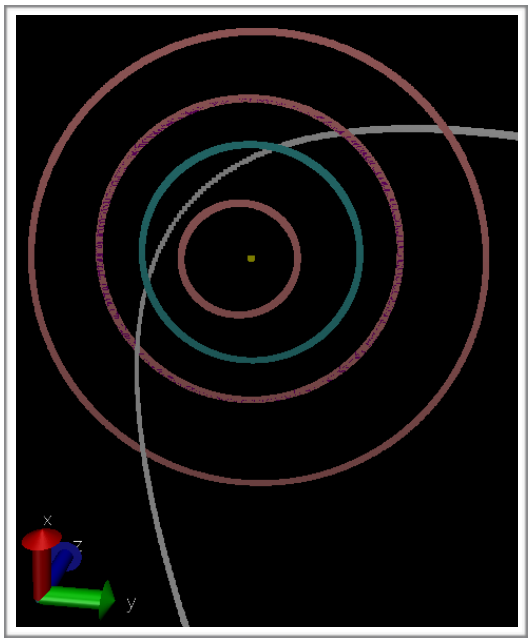


Figure 7 - Zoom in of Elliptical Trajectories

The ratio of the final to initial energy is 0.9973.

V. More Simulations

To answer the questions, such as "Is the Moon really orbiting the Earth?" or "Does the comet's orbit show any perturbations?", additional simulations were run for a variety of parameters.

Halley's Comet Perturbation:

The simulation is run for 1.000.000 steps with a time step of 5 days.

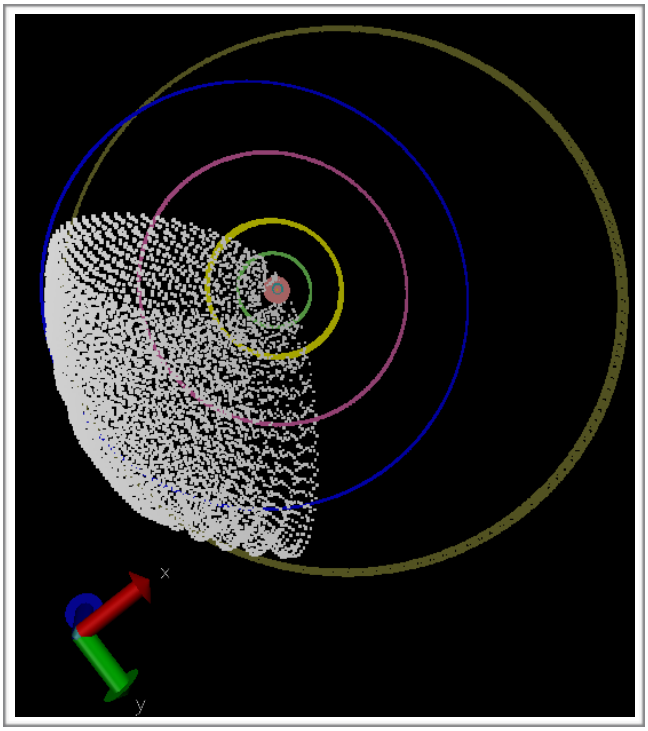


Figure 7 - Halley's Comet Perturbation

Moon's Orbit:

The simulation is run for 1.000.000 steps with a time step of 0.1 days.

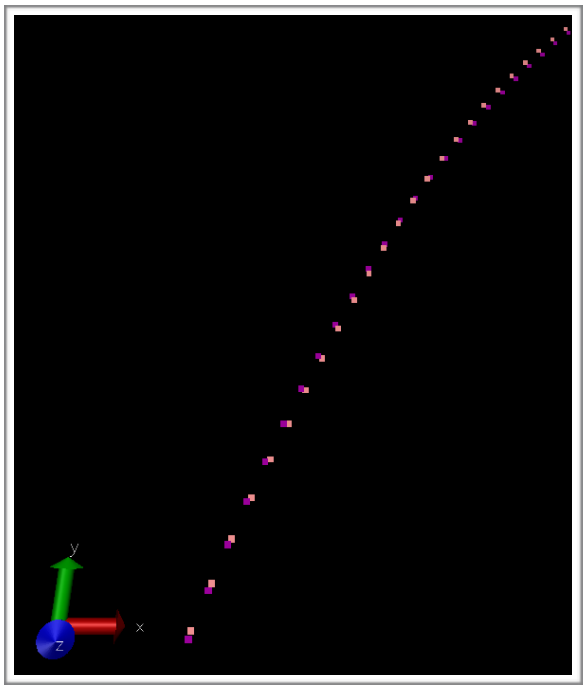


Figure 8 - Moon's Orbit

Note, the moon and earth is depicted by the dark purple and pink points, respectively.

Discussion

The three body system consisting of the Sun, Mercury and Venus produced very accurate results that were within 0.1% of the real year length. The number of simulated orbits were excellent, since they only recorded completed orbits, i.e. they returned integer values. This simulation was the most accurate in terms of the ratio of the initial energy to the final energy, giving a 0.065% error.

Compared to circular motion,the elliptical motion of the three body system is very similar, if not slightly more accurate. For instance, the year length of Mercury was simulated to be 87.99, which is just 0.003 days lower than the real year length. This would correspond to a 0.034% error. Interestingly, the energy ratio of the elliptical orbit was about 10-fold that of the circular orbit. This is likely due to the nature of the elliptical orbit, since the planets distance to the sun changes much more substantially.

Our solar system simulation produced very good results. The largest error in the simulated year ratio and year length is just 0.6%. Similarly, the largest error in the aphelion and perihelion calculations is 0.5%. Intriguingly, the largest error occurred for the planets that were furthest from the sun. It is uncertain as to why this is the case. The moon orbits the earth the correct amount of times, as indicated by the simulated year length of 27.60 days. Figure 8 clearly shows that the moon is in fact orbiting the earth. It should be noted that calculating the aphelion and perihelion for the moon is rather pointless, instead the apogee and perigee should have been calculated.

While the results for Pluto and Halley's Comet were not as accurate as for the other planets, they were still more than satisfactory. Pluto has the largest error in year length of 2.0%, and the error in the simulated perihelion is 1.9% for Halley's Comet. Again, this is consistent with the previously mentioned observation: the planets planets furthest from the sun produced the largest error. Then again, Halley's Comet error in year length is just 0.03%! On a side note, Figure 5 visualises the ecliptic angle of Pluto and Halley's Comet very nicely.

When increasing the time step to 5 days, Halley's Comet started to show unwanted perturbations as shown in Figure 7. However, when Halley's Comet approaches the sun, its velocity increases by a considerable amount. If the time step is too large, this will result in Halley's Comet overshooting and thus perturbations will begin to show.

This leads to the next discussion point. What is the largest time step that accurately simulates the solar system and does the inclusion of Halley's Comet change this this time step? Well, 'accurately' simulating the solar system comes down to personal preference. In this report, it was deemed appropriate to choose the time step so that the maximum error of all results does not exceed 2.5%. There are many indicators of how to find the largest time step. For example, the energy conservation of the system should be constant, i.e. the ratio of initial energy to the final energy should be 1. Moreover, the time step should be reduced until apsidal precession is visually no longer present. In the simulation, Halley's Comet and the Moon have a tremendous effect on how high the time step can be dialled. This is because the moon has the smallest orbit, so it changes its direction very quickly when compared to planets and Halley's Comet will overshoot, causing perturbation. When taking all of these circumstances into account, the largest time step that accurately simulates the solar system is 0.1 days. However it must be restated that this is not the only acceptable time step. When closely observing Halley's Comet's trajectory in Figure 5, it is evident that the trajectory is slightly thicker than those of the other planets, due to ever so small perturbations. One could have reduced the time step even further, but this was not done, since this would have meant an increase in our total amount of steps (so that pluto completes an orbit), and thus even longer computation time.

Lastly, I would like to talk about our group. Overall, I am very satisfied with our group as a whole, there were no major issues. To our benefit one member studies astrophysics and the other two are proficient in java.

Conclusion and Post-Mortem

In conclusion, the outcome of the N-body astrophysical simulation using Newton's gravity produced excellent results. All simulated values agree with the real values within less than 2.5% and the appropriate time step, while maintaining accuracy, is 0.1 days. Even so, it is worth mentioning that pluto had a considerably larger error than most planets. There were no major issues within the group.

Improvements:

The simulation is limited to one moon, that is the earth's moon. However, other planets have moons as well, for instance, Jupiter has 63 moons. The scan method could be changed so that it recognises a keyword such as 'Jupiter_Moon1' and then associates a planet/body with it. Moreover, the perihelion and aphelion calculation should be stopped after one planet completes an orbit to save computing time. Another possibility would be to average the perihelion and aphelion results of all orbits, producing a more accurate value. Furthermore, the updateTheta method calculation should make use of the dot product method in the Vector3D. This way, the period can be found for a particle orbiting only in x-z plane or y-z plane. Currently, the code can only calculate the period in the x-y plane. Most importantly, the force calculations should be changed. Instead of calculating the force between all permutations of bodies, it would be better to find the force of body1 on body2, then use Newton's third law and simply reverse the direction of the previously calculated force. This would save a lot of computer time. Unfortunately, time constraints did not allow for these modifications.

References

[1] "Home." Drexel University -. Web. 25 Nov. 2014. <http://einstein.drexel.edu/students/courses/Comp_Phys/Integrators/leapfrog/>.

[2] "HORIZONS Web-Interface." HORIZONS Web-Interface. Web. 27 Nov. 2014. <http://ssd.jpl.nasa.gov/horizons.cgi>.

[3] "Tables." Tables. Web. 27 Nov. 2014. <http://www.astronomynotes.com/tables/tablesb.htm>.