Computer Simulation Project Report:
Simulating the Solar System

Nicholas Piano - S0903260

April 5, 2012

Abstract

The goal of this project was to simulate the inner Solar System, send satellites to Mars and Jupiter using the program structure, and verify its accuracy using the periods and energy of the system as criteria. The results were satisfactory. The closest approach to Mars was $1.515 \times 10^7 m$, the Jupiter slingshot was found to be extremely sensitive to initial conditions, the closest approach actually hitting Jupiter at $4 \times 10^6 m$, and the periods of the planets were found to be 87.9802, 223.301, 365.311, 685.179 and 4330.772 days respectively. The ratios of these to Earth's period are correct within a small margin of error. The simulation was found to be very effective.

0.1 Introduction

The goal of this project was to simulate the inner Solar System and investigate its properties by manipulating the program. The simulation was written primarily in Java using the extremely effective OOP or Object Oriented Programming approach. The simulation was split, using this method, into a group of classes with associated objects that could interact with each other. The full details of this approach are described in the third section: Method and Object Oriented Approach. Figure 1 shows some of the results of the simulation in graphic form. The graphics are not strictly speaking necessary, but are useful for visualising the effects of simulated gravity and the interactions between the heavenly bodies.

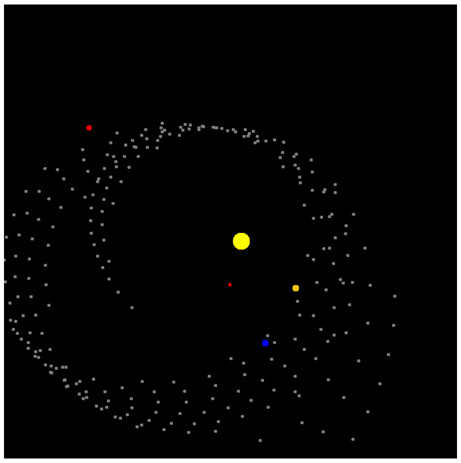

Figure 1: The power of simulation. In more detail, this image shows a group of approximately three hundred and sixty satellites, each with a mass of a thousand kilograms, orbiting the sun after having been given a unique impulse from the earth approximately three years prior to the image.

The main method of the program was then converted into several forms in order to carry out experiments. The experiments performed were the Jupiter Satellite, the Mars Satellite, and Energy Conservation. In the Jupiter mission, the goal was to fire a satellite towards Jupiter in order to slingshot around it. This slingshot event was then investigated by measuring the effects of minute changes in launch angle and velocity on the resultant trajectory. For the Mars mission, launch velocities were found that allowed a satellite to perform a fly-by of Mars. In Energy Conservation, the conditions for energy conservation were investigated, and reasons for the lack of energy conservation discussed with regard to the simulation itself.

0.2 Theory

Before any simulation began, the coding of this project was founded on a solid theoretical basis. The main component of the theory is Newton's Law of Universal Gravitation, given by the following equation:

$$F(\vec{r}) = \frac{-GMm}{r^3}\vec{r} \tag{1}$$

Unfortunately, this function is not suited for the purposes of simulation because it is continuous and non-algorithmic. Computer Simulation works best around functions that are based on a time-step or some similar iteration. For this reason, an algorithm is used. There were several choices including Einstein's and the Runga-Kutta Method, but the one chosen for this simulation was the Beeman Algorithm.

0.2.1 The Beeman Algorithm

The Beeman Algorithm is a three-step process that allows the velocity and position of a body to be calculated quasi-simultaneously given an initial position, velocity and acceleration. In other words, each time-step is based on the previous one, and as the time-step interval approaches zero, the process becomes more and more precise. It is effectively a method of numerical integration. The two equations used in this algorithm are given below:

$$x(t+\Delta) = x(t) + \dot{x}(t)\Delta + \frac{1}{6}[4\ddot{x}(t) - \ddot{x}(t-\Delta)]\Delta^2 \tag{2}$$

and

$$\dot{x}(t+\Delta) = \dot{x}(t) + \frac{1}{6}[2\ddot{x}(t+\Delta) + 5\ddot{x}(t) - \ddot{x}(t-\Delta)]\Delta \tag{3}$$

These are used to calculate the next position and velocity respectively.

0.3 Method and Object Oriented Approach

In order to simulate the inner Solar System, the program was written to be optimised for a certain number of planetary bodies, although it can be easily expanded simply by adding more data and tweaking a few values. The structure of the program is heavily dependent on an approach known as Object Oriented Programming.

0.3.1 OOP

0.3.1.1 Concept

The main focus is to modularise the code so that it can be separated easily and individual parts can be used and reused without accessing the source code of the entire program. This allows for a very efficient design. In this approach, the code is divided into Classes, each of which contains a constructor to instantiate objects, and various methods, functions, or subroutines to manipulate these objects.

0.3.1.2 Class Structure

A diagram of the class structure is shown in Figure 2:

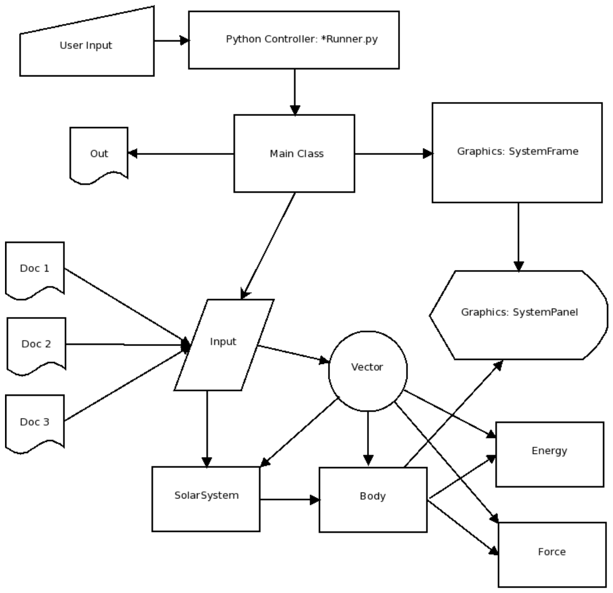

Figure 2: A flow chart showing the workings of the Program. The flows represent documents and classes

0.3.1.3 Main Class

The main class instantiates a SolarSystem Object, which in turn instantiates Body Objects to populate it. It has no methods to manipulate it as its purpose is only to consolidate the simulation as a whole. It serves to input initial values, run the simulation by iterating through a number of values such as time, velocity and angle, and print the results of various measurements to a file or the terminal.

0.3.1.4 SolarSystem

The SolarSystem classes instantiates a SolarSystem object, which in turn contains Bodies. It was meant to summarise all the unified properties of the collection of bodies such as centre of mass, total energy and total mass. It instantiated an array of Bodies and served as a relay between the Main class and the Body class. The input data was gathered from the JPL website and gives the positions and velocities of the planets on $1^{st}$of January, 2000.

0.3.1.5 Body

This class could be said to be the most important, perhaps surpassed by the Algorithm Class. Each Body object has a number of properties as private variables including mass, initial velocity, and initial angle with respect to the coordinate system of the sun.

0.3.1.6 Algorithm

This class is at the heart of the program. It instantiates an Algorithm Object which stores a characteristic time-step to run the simulation. This time-step is set from the main class. It also contains several static methods used to manipulate Body objects. These methods were not included in the Body class in order to separate the idea of a general Body from the specific algorithm in this simulation. If the forces and accelerations were fully generalised, it could be moved instead to the body class.

0.3.1.7 Vector

The Vector Class instantiates Vector objects that can be added, subtracted, dot- and cross-multiplied and manipulated in other ways such as finding the unit vector and magnitude by specialised methods. These vectors can be created, set and get for anything from position to force.

0.3.1.8 Graphics

This class, SystemFrame, contains a frame to display an instantiated panel, SystemPanel. The panel displays an array of coordinates gathered at each time-step from the array of Bodies in the SolarSystem class. This class is limited by its method of painting each of the bodies with their different colours and sizes to approximate those of the planets in Sol. In order to simulate another solar system, these values would have to be changed, although this is not hard.

0.3.1.9 Energy

This class gathers values for the kinetic and potential energy from the velocities and position vector magnitudes of all the bodies and contains methods to return the total energy of the system in order to plot it.

0.3.1.10 Force

This class contains a formula for Newtonian Gravity with its associated constants in order to calculate the acceleration of each planet to be used in the Beeman Algorithm.

0.3.2 Experiments

Once the simulation of the inner solar system was complete and functioning, several experiments were performed to judge the accuracy of the system.

0.3.2.1 Energy Conservation

In the first experiment, the conditions for the conservation of total energy were investigated. This was done by regularly printing the values of the kinetic, potential and total energies to a file and examining plots of the data.

0.3.2.2 Mars Mission

There were several choices to be made when running this experiment. The first option was to run one satellite at a time until one or several of them got sufficiently close to Mars to be a fly-by, which is said to be approximately ten million kilometres, depending on the planet. The second option was to run a large number of satellites at the same time in the hope that one or several would hit mars. The second method was chosen as the time taken was significantly less even with the extra calculations needed for multiple satellites. In the end, a fleet of 361 satellites were launched from Earth in clusters of ever increasing precision. The precision was increased by using a method in the SolarSystem class to determine the winner each time. The satellites were launched in 2002 simulation time. Figure 3 shows some of the results of multiple satellites. It was also useful to visualise all the satellites at once in order to understand the dynamics of the system.

0.3.2.3 Jupiter Mission

A similar approach was taken in the Jupiter Mission, whereby approximately 360 satellites were launched from Earth towards Jupiter. This time, when a

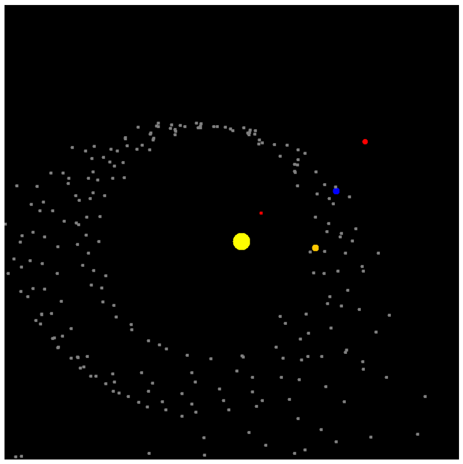

Figure 3: 361 Satellites orbiting the sun several simulated years after being launched from Earth.

satellite passed within $1 \times 10^9 m$ of Jupiter's centre, the coordinates of its position were recorded and printed to a file so the trajectory of the slingshot could be plotted and visualised. The launch conditions of the satellite that performed the slingshot were then varied slightly to produce another plot to be compared with the first.

0.4 Results and Analysis

0.4.1 Periods

The periods of five the planets, Mercury, Venus, Earth, Mars and Jupiter were calculated over a period of 1000 simulated Earth-years. This was done by finding the time-step at which the planet repeated its initial angle within a certain margin of error (half its angular velocity) while also incorporating this error into the period. Table 1 shows the calculated values, the true values, and their true and calculated ratios to Earth's period.

Body	Period (days)	True Period (days)	Calculated Earth Years	True Ratio
Mercury	86.9802	87.969	0.2381	0.241
Venus	223.301	224.7	0.6113	0.6152
Earth	365.311	365.241	1.00019 (to true value)	1
Mars	685.179	686.971	1.8756	1.8808
Jupiter	4330.772	4332.59	11.8550	11.8618

Table 1: The Orbital Periods of the Inner Planets

0.4.2 Energy

The simulation was run to see if the total energy of the system was conserved. Figure 4 shows a plot of the potential, kinetic and total energies of the system at each timestep. This was done with a time-step of a day.

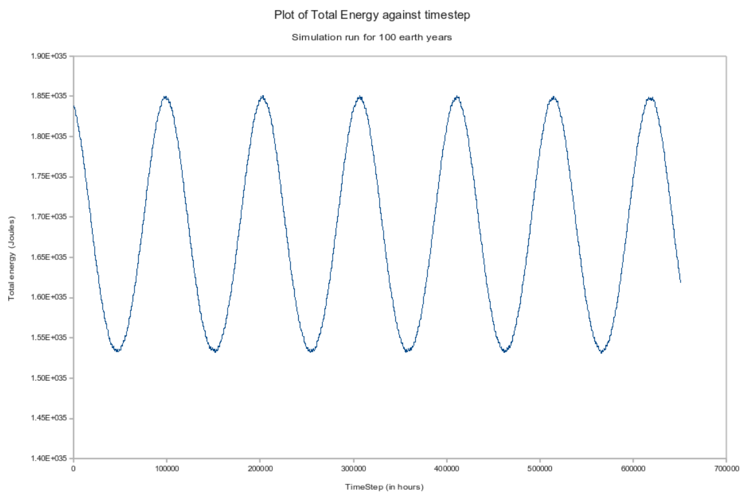

Figure 4: Energy of the Solar System over time

A close up of one of the peaks is shown in Figure 5:

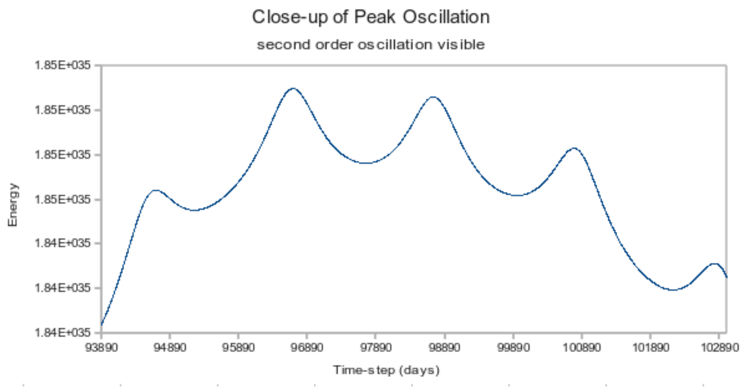

Figure 5: Energy Oscillation close-up

One of the main reasons for the energy not being conserved is the fact that this simulation only considers two dimensions, so a large section of energy data is ignored. The other planets such as Uranus and Neptune are also ignored along with the rest of the mass of the solar system. Energy is stored by this mass, both kinetically and potentially, so including this mass would greatly reduce the oscillations. It is no surprise to find that the period of oscillation in the above graphs is almost that of Jupiter. This is because Jupiter and the Sun orbit a approximately common centre of mass being the most dominant masses in Sol.

0.4.3 Mars Mission

Figure 6 shows a plot of launch angle against distance of closest approach. It gives exactly the result expected with one peak at the record holders angle.

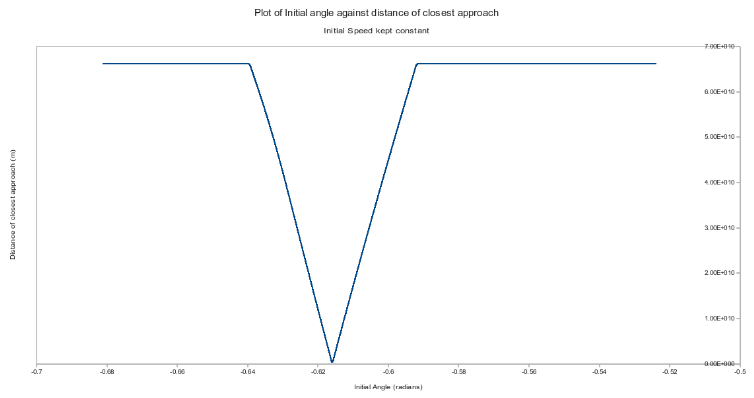

Figure 6: Distance of closest approach

The details of the winning satellite are given below in Table 2.

Launch angle	Launch Velocity	Distance	Time taken
127.45 degrees	9448.205ms^{-1}	$1.515 \times 10^7 m$	1.59 years

Table 2: Winning Mars Satellite

The winner did not compare well to the Voyager 1 probe in 1976, which did the journey in 9 months and 30 days or 0.839 years. This was most likely to the difference in configuration of the Solar System. It is likely that the configuration in 2002 was not optimal. Given the right conditions however, the satellites did in fact return to earth due to the intersections of the orbits. In order to do this, the satellites could not pass too close to Mars in case they were overly affected by its gravitational pull.

0.4.4 Jupiter Mission

Figure 7 shows a plot of three slingshot trajectories around Jupiter at a distance of approximately$1 \times 10^9 m$. This was chosen as the slingshot distance in the style of the Cassini-Huygens probe in 2004, which performed a slingshot at about the orbit of Io, or $4.2 \times 10^9 m$.

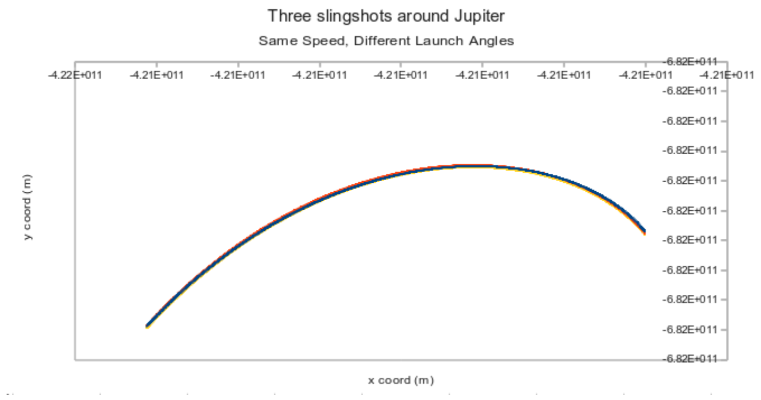

Figure 7: Jupiter Slingshot: These three lines differ in launch angle by $1 \times 10^{-3} radians$

The winning satellite actually hit Jupiter, coming within $4 \times 10^6 m$ of Jupiter's centre. The radius of Jupiter is approximately $7 \times 10^7 m$.

0.5 Discussion

There were several problems that merited discussion and several improvements that would be useful yet time-consuming to carry out.

0.5.1 Problems

one of the major problems is the fact that the third dimension is not considered. However, the program was written with this in mind and 3D could be enabled relatively easily, but would complicate calculations and increase processing time. Another problem is number of bodies in the Solar System. A small simulation such as this one cannot fully simulates the likes of the Solar System due to a lack of processing power. It would useful to find out at what point the simulation becomes overloaded and find ways to compensate.

0.5.2 Improvements

Ideally, such a simulation would be written in a more efficient, memory-savvy code such as C or FORTRAN. Although the Object Oriented Approach in Java is useful and easy to visualise, it is not efficient and requires a lot of processing power to perform relatively small tasks. It is also not optimal in its memory and storage management. If I were to rewrite this code, it would be in FORTRAN.

To further improve it, the code could be made even more modular and general by making Body an abstract class and separating it into sub-classes like Planet, Asteroid, Star and Satellite. The Force class could also be split into Gravity, Electrostatic, Strong and Weak, for use in further simulations.

Of course, the simulation could also be improve by small thing such as reassessing the algorithm and number of calculations done on an "absolutely-need-to-calculate" basis in order to weed out the unnecessary processes.

0.6 Conclusion and Evaluation

In conclusion, the simulation of the inner planets in the Solar System was successful and surprisingly accurate.

0.6.1 Results

The total energy of the simulation was found to oscillate within 1% of . It was found to be conservative, but it oscillated with the period of Jupiter, implying that Jupiter is the most dominant mass in the Solar system besides the Sun, a result confirmed by its true mass. The best result for the mars fly-by was , obtained by launching it with velocity and angle relative to Earth. The slingshot event around Jupiter was found to be extremely sensitive to initial launch conditions, responding significantly to changes in the angle as small as $10^{-3} radians$. The record satellite hit Jupiter at , with an initial launch velocity of and launch angle of .

0.6.2 Effectiveness of Simulation

The terminals provided were adequate, although more detailed simulations could have been done with faster and more capable processors. The computers provided had only two cores, and the system did not support a simple way of choosing which core to use for a particular set of calculations. Core management could have greatly increased the speed of the calculations.

0.6.3 Power of Simulation

Finally, even in this simple project, the power of computer simulation is apparent and is worth investigating further. It is valuable to question what our ability to simulate a system accurately implies about the nature of the system itself. in other words, does gravity calculate itself somehow? If not, how are we able to simulate something that approaches reality, even if we cannot ever truly represent it. Difficult as it is to picture, we are essentially creating an image of the real universe in the simulation. The true reality must be dependent in some way, if a more complicated way, on the things the simulation depends on. 'Tis an interesting question.

0.7 Bibliography

1. Edinburgh University Computer Simulation Handbook 2012

    (a) (https://www.vle.ed.ac.uk/webct/urw/lc102122001.tp0/cobaltMainFrame.dowebct)

2. JPL HORIZON Ephemeris generator

    (a) http://ssd.jpl.nasa.gov/horizons.cgi