**A Solar System Simulation and Analysis Using Newtonian Physics with the Beeman Numerical Integration Algorithm for the First Four Planetary Bodies with an Artificial Satellite**

Osian ap Sion (s1811930)

**Introduction**

In this project a simulation of the solar system was constructed using the Beeman Numerical Integration Algorithm, given by:

Where is the position vector, is the velocity vector, t is the current time, is the time-step and is given by:

The object details, including the satellite and simulation parameters were read from file, then the Beeman integration scheme was used to update the position and velocity of each object for a given number of time-steps until the end of the simulation. The details of these objects were logged at each time-step for later analysis.

Using the simulation log, the total energy, potential energy and kinetic energy of the system for each time-step were calculated and graphed, these were then outputted to file. The orbital periods of the simulation objects were calculated in Earth years and printed to the console and written to file. These experiments were conducted to test if the energy was conserved in the simulation, as well as to compare the simulated orbital periods with their real values.

Over multiple simulations, a range of velocities were added to the satellite in an attempt to get the satellite close to mars. This was to see if the simulation was versatile when applied to real world applications.

**Methods**

Three classes were chosen for this simulation. The first, Solar\_system\_object(), defines an object inside the simulation including its position, velocity and acceleration.

Simulation(), essentially dictates how the space (position) and time of the objects are related to each other. It reads the list of the simulation objects from file, tracks the current time of the simulation and updates their positions, velocities and accelerations for each time-step. It then stores these variables into the variable, log, each time the function log\_variables() is called. This is for further analysis.

A notable feature in this class was the careful updating of the object variables in the function time\_step\_forward(), where the acceleration and velocity were calculated and updated for all objects, and only then the position. A simpler method would be to iterate through all objects, updating the acceleration, velocity and position, but in doing this, the position of the first object in the list would be updated before calculating the variables for the second object, therefore causing the simulation to have a bias for the objects at the beginning of the object list. This could cause large errors with objects that are close together, such as Earth and the satellite, where Earth would move a time-step forward (2000 seconds) travelling an additional distance of 59,560km before the satellite would calculate its acceleration due to Earth.

A more sophisticated method of interpreting the simulation data from file was chosen (the function get\_values()). This was to allow instruction at the beginning of the file, as well as to allow the planet variables and simulation variables be defined in only one text file. There was also an added functionality, where data in the text file could be defined with an exponent of ten, making writing large values into the text file much clearer.

The final class, Solar\_system\_analysis is for the data analysis and visualisation. This class makes extensive use of the log variable in Simulation() to process the data and then either export graphs, print to console, export to text, or export animations of the data. All the analysed data was exported to file, this was to make viewing the data more robust, so that closing the visualisation would not delete it forever. This is especially true for the animation - where it was exported to an mp4 file. This was to make replaying and sharing the animation considerably easier. This did however cause it to be challenging to simply show any visualisations with plt.show(), due to matplotlib using agg, a non-GUI backend.

The orbital periods were calculated by using the fact that in an orbit, the object would cross the y-axis twice, each time changing the sign of the x-coordinate. Therefore, it was possible to calculate the period by measuring the time in between each occurrence of the objects x-coordinate changing sign, and the subsequent sign being positive. This only happened once each orbital period, allowing a precise measurement of the orbital period. This method did require a rather lengthy simulation period for all the objects to complete a full orbit after crossing the y-axis.

The starting position and speed for the planets were chosen to be the average orbital speed and average orbital radius of their real-life counterparts. This was to simplify the calculation when placing a planet somewhere other than on one of the axes, where calculating the velocity, direction and position on an ellipse would be considerably more difficult than on a circle. This was done for the starting position of mars to assist in the satellite experiment.

When defining the simulation parameters, a precision and duration variable is required. These were chosen, rather than simpler number of time-steps and time-step size(seconds) parameters to simplify the input values. This would avoid forcing the user to calculate the duration of the experiment by multiplying the number of time-steps with the length of a time-step.

The starting values of the satellite experiment was chosen to be that of a Hohmann transfer orbit, with mars at approximately 44 relative to Earth. The initial satellite velocity was calculated as the sum of Earths velocity, orbital velocity at 35786km away from Earth, and the delta v required to reach mars. This was calculated to be 29780.0 + 3070 +v = 32850 + v. The satellite was placed at geostationary orbit because this was far enough from Earth to not cause any issues in the simulation (due to relatively large time-steps) but close enough to still be affected by Earth’s gravity. A range of initial test values were chosen ranging from 35400 to 35800m/s.

**Results and Discussion**

**Orbital Periods**

The orbital periods measured in the simulation were

|  |  |  |
| --- | --- | --- |
| Object | Simulated Orbital Period (years) | Actual Orbital Period (years) |
| Sun | 1.0781 | na |
| Mercury | 0.2333 | 0.240846 |
| Venus | 0.6149 | 0.615198 |
| Earth | 0.9995 | 1 |
| Mars | 1.8539 | 1.88085 |
| satellite | 1.4237 | na |

We can see that the simulation matched the orbital period of mercury, Venus, Earth and Mars within 4%, 0.005%, 0.0006% and 1.5% respectively. This shows that the simulation is able to accurately predict the orbital periods of the inner planets. The relatively large disparity in Mercuries orbit could be due to the more relativistic conditions close to the sun.

**Energy Conservation**

The total energy of the system is given by:

Chart, bar chart

Description automatically generated

We can see that the total energy value oscillates around -6.21092e33 J but does not deviate outside of this value much more than +- 3e28 J. This shows that over the time period of this simulation, energy is conserved.

**Satellite to Mars**

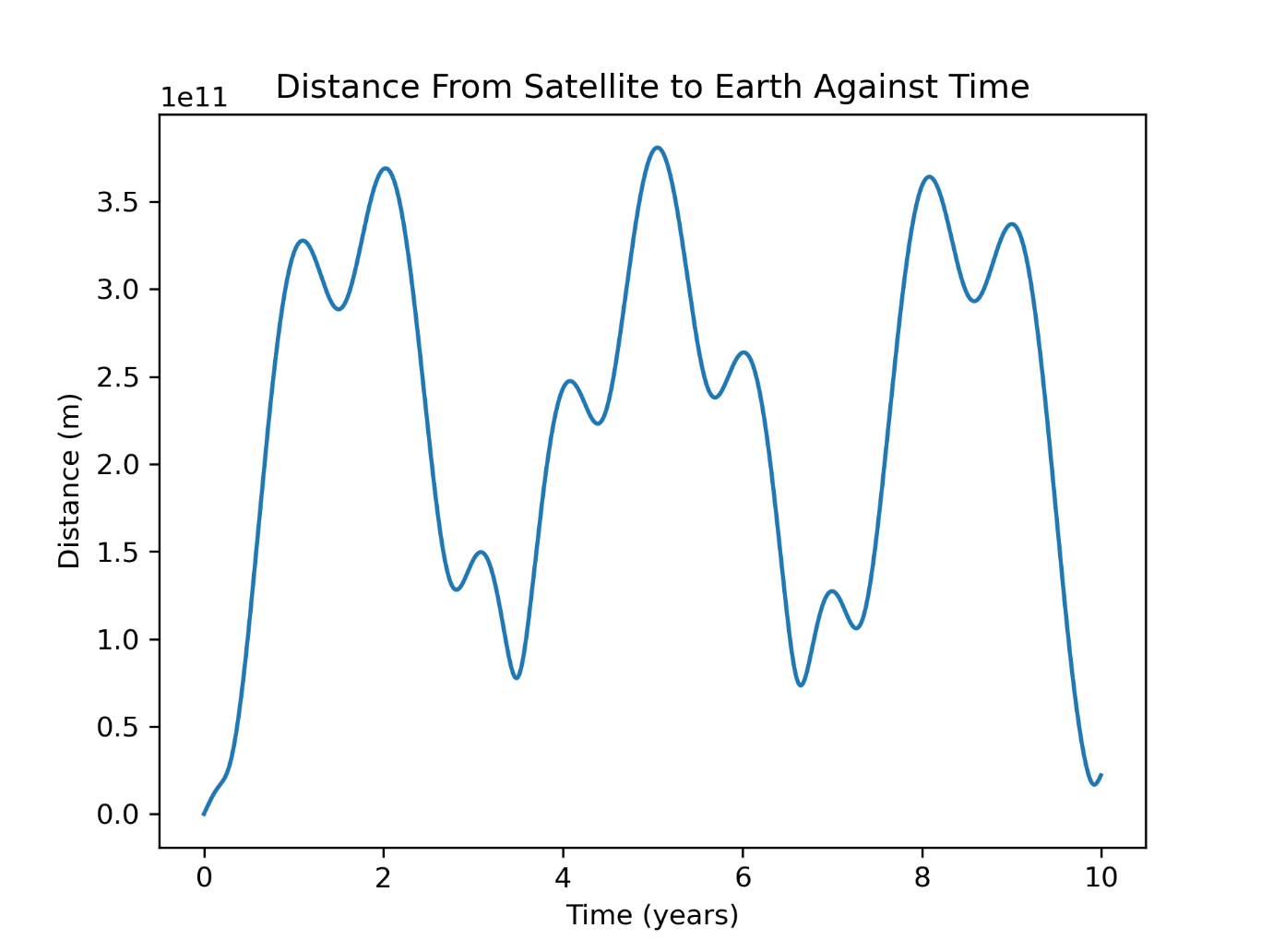
The Satellite reaches approximately 300000km from Mars, this is well within the effective range of its gravity. It took about 343 days for the satellite to reach Mars, compared to the 203 days of the Perseverance mission. The range of initial velocities tested were the following:

|  |  |
| --- | --- |
| Starting y velocity (m/s) | Distance from Mars (km) |
| 35400 | 52231841.17 |
| 35500 | 28351667.58 |
| 35550 | 15968580.97 |
| 35575 | 9796618.157 |
| 35600 | 3765476.618 |
| 35615 | 298387.0669 |
| 35625 | 1900596.961 |
| 35650 | 6841568.098 |
| 35700 | 13660535.77 |
| 35800 | 19304612.06 |

From the table we can see this distance was achieved using an initial velocity of 35615 m/s. Here is a graph to visualise this data:

Note, the first two values were omitted to make the visualisation clearer.

For this value of initial velocity, the satellite came relatively close to earth as demonstrated in the following visualisation:



If there was some fuel left in the craft, it likely would not require much energy to alter the crafts trajectory back towards Earth.

**Discussion**

The orbital periods were comparable to within 5% to real values. This demonstrates that the simulation is an effective predictor of real orbital periods of the inner planets. The orbital periods of Venus and Earth were especially close to the real-life values, being accurate to 0.005% and 0.0006% respectively. The disparity in Mercuries value could be because of the more relativistic circumstances known to affect Mercuries orbit.

The total energy was conserved within the simulation, demonstrating the symplectic nature of the Beeman algorithm as well as the validity of the simulation over extended periods of time.

The satellite successfully reached mars with a distance of approximately 300000km using realistic initial velocities and an arrival time about 140 days slower than that of the Perseverance mission. This demonstrates that this simulation has potential for real world applications, although further experiments must be made to determine the accuracy of the simulation for close objects.

**Conclusion**

This simulation was successful in simulating the path of the inner planets, demonstrating similar orbital periods, conservation of energy and its ability to simulate interplanetary satellite missions.

Finally, FFmpeg is used by the Perseverance rover on Mars for image and video compression before being sent back to Earth.[[66]](https://en.wikipedia.org/wiki/FFmpeg#cite_note-77) This is the same software I used to output the solar system animation to mp4.