Solar Simulation Experiment
Computer Simulation Project 2019

Benjamin Carpenter (S1731178)

7th March to 5th April 2019

Contents

1 Introduction & Aims

This experiment would aim to simulate the motion of bodies within the inner solar system and to use this simulation to determine if the Beeman algorithm would conserve total energy over time (as would be expected for a simplified model of the solar system). The simulation would also be used to plot a course for a probe from the surface of Earth to some distance close to Mars and determine if this path could also allow for a return journey to Earth; this probe path would be compared to a probe path used by NASA's Viking mission[9].

2 Theory used to simulate orbital motion

2.1 The Gravitational force

Objects in space move due to gravity from other objects. This motion is dependent upon the distances between objects as well as the masses of these objects, the force acting between two objects of mass m_1 and m_2, with displacement $\vec{r}$ is given by:

$$\vec{F} = -\frac{Gm_1m_2}{r^2}\hat{\mathbf{r}} \tag{1}$$

From [4, s. 3.12]

Where G is a constant $G = 6.674\,08 \times 10^{-11}\mathrm{m}^3\mathrm{kg}^{-1}\mathrm{s}^{-2}$ [2] and $\hat{\mathbf{r}}$ is toward the other body as shown in fig.1.

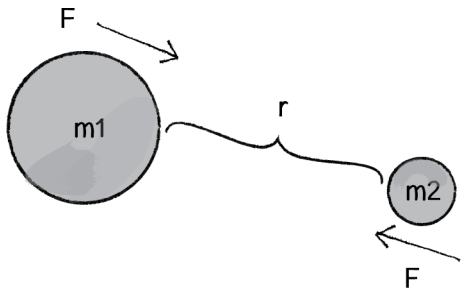


Figure 1: Gravitational Force (1) between two masses
[3]

2.2 Potential and Kinetic Energies of objects moving in space

The kinetic energy of an object moving in space can be described simply from its mass m and speed v in the relationship:

$$E_K = \frac{1}{2}mv^2 \tag{2}$$

From [4, s. 4.4]

Energy that a body stores due to its position relative to another body is given by the integral of the magnitude of force (described in eqn 1) with respect to position hence the relationship:

$$E_P = \frac{Gm_1m_2}{r} \tag{3}$$

From [4, s. 4.6]

It should be noted that as this is a pair potential, if summing these energies for a single object then the produced value should be halved to prevent 'double counting' energies.

3 Implemented code and Experimental Method

3.1 Overall interaction between objects

The main simulation was made up of three classes: A space object would contain within it many body objects which would each represent a planet or other type of body (such as a probe). An AnimateSpace class could then be used to either animate this space before or after data for it had been processed.

Two other minor classes were included in the program however these provide minimal functionality:

Probe develops on the body class adding features that would only be expected of a controlled object (e.g. launching after two years as specified in experiment.3 [5, s. Solar 1.4.3]).

Path acts as a storage structure to simplify interaction with data about a bodies path as it travels through space being used by `Animatespace` to show a bodies path.

These interactions are summarised in fig.2.

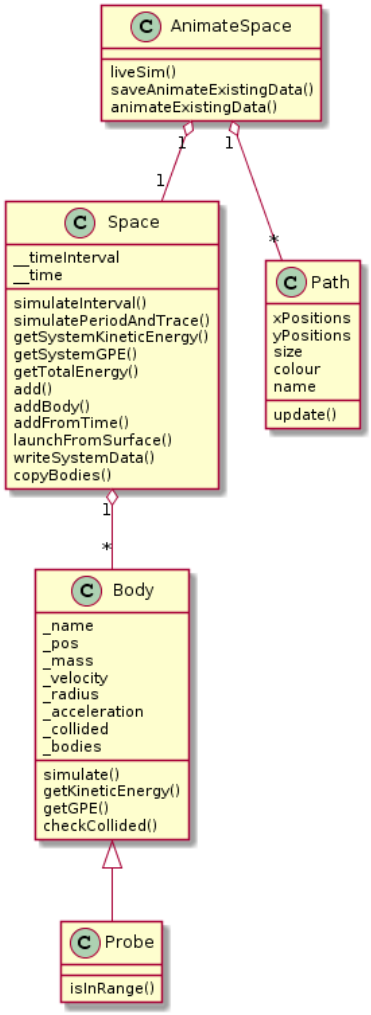


Figure 2: Class diagram of the main simulation files

3.2 Bodies within space - The Body and Probe Class

This class is used to provide the properties of each body as well as being responsible for updating these values when called to do so with the `simulate` method. Its properties also providing details to produce a representation in an animation as shown in fig.2.

3.2.1 Simulating changes in position - The Beeman Integration method

Each body would have a property which referenced the list of bodies stored within the space object it was a part of; this was important as the motion of a body in space is dependent upon other masses that it is in contact with (see eqn 1).

Being able to calculate this force allowed for an acceleration to be calculated, simply by Newton's second law, this could hence be used with the Beeman integration method to produce a new value for position and velocity which could then be used to produce the next 'steps' force.

Applying this iteratively could simulate a bodies movement in space accurately. This method required storing the acceleration of the body and required both a step back and forward acceleration at the same at the same time. Usage of the various acceleration variables followed the algorithm show shown in fig.3, noting that `__acceleration` is the only property that is stored outside of the scope of this algorithm.

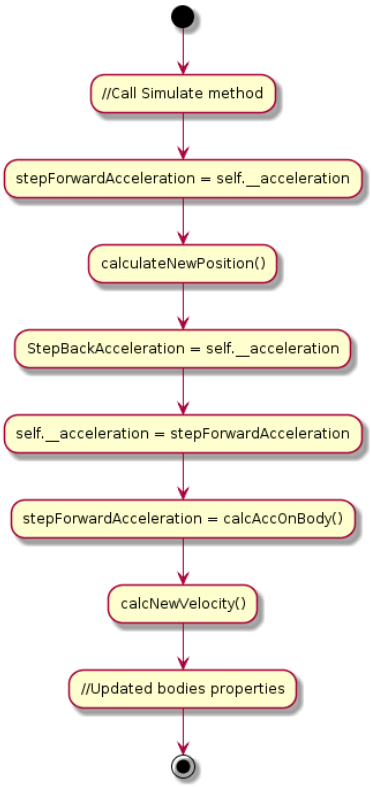
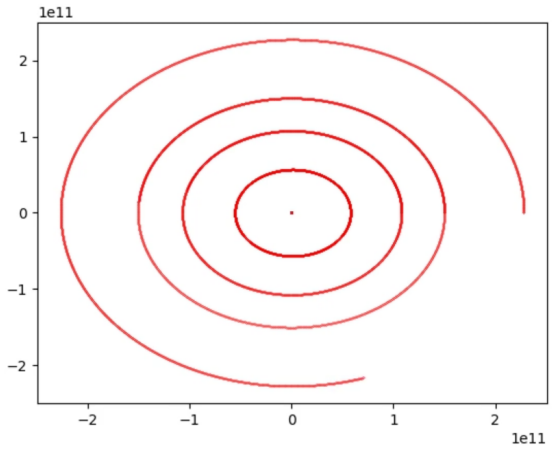


Figure 3: Basic algorithm for updating the position and velocity.

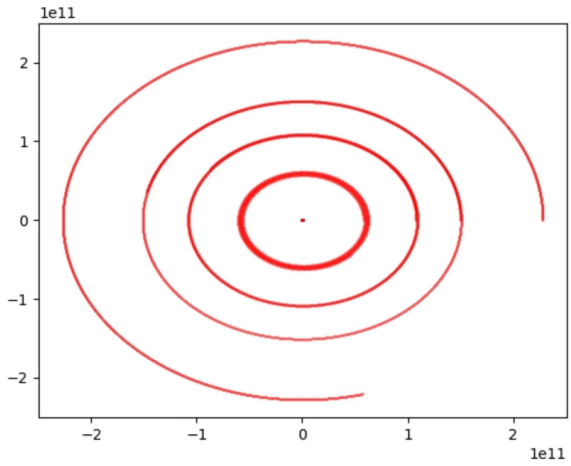
It was expected that when calculating the 'step ahead' positions, the 'current position' was not to be updated, this would be expected to have the effect of adding a systematic error to each position where they were off-put due to not using the correct position for a given interval ,therefore, a copy of the space objects `__bodies` would be updated with the new positions which would have the effect of keeping the original property constant and removing the error. This copying would be done before an interval simulation i.e. where many bodies were being updated and hence was implemented in `Space`.

Unexpectedly, this had the effect of causing an apparent increase in energy, for example, the orbits of planets would appear to increase as shown in fig.4

It was therefore decided that the bodies would not be copied at each interval and instead would simply use the list of bodies as it was as this appeared to produce the correct behaviour. This is further justified in appendix A.



(a) Produced Graphic when using updated positions to calculate acceleration of a body in an iteration



(b) Produced Graphic when using non updated positions to calculate the acceleration of a body in an iteration

Figure 4: Apparent proper behaviour shown by algorithm that did not seem initially valid (i.e. deviation from a constant path is visible in 4b due to wider path lines)

3.2.2 Dealing with collisions

A property to note was the boolean `_collided` which record if a body had collided with another (determined from if one body was at a distance that was within the distance of another plus its radius) afterwhich the simulation could not reliably simulate its behaviour due to loss of energies in the collision.

By marking these objects as collided experiments which were run using this program would be able to choose what should happen when a collision occurs rather than the source program providing some action that may be inappropriate e.g. a tiny probe colliding with the sun will have little effect to the system. Recording allows a specific behaviour to be implemented by the experimenter.

3.3 The Programs simulation space - The Space Class

Acting as a container and controller for the bodies to be simulated in meant that the `Space` class would mostly contain functionality for making each body perform actions as well as collecting data about the entire system.

A representative method that showed this behaviour was the `simulatePeriodAndTrace` method which was used to generate simulation data of the currently setup system (usually read in from file by the `addFromFile` method) over a given time and record the properties at each 'universal' time step. This functionality would involve going through each body created in the `Space` object and simulating the time step for each separately (using their `simulate` methods). Then the new body positions and properties could be recorded by copying each body into a new list which would contain the changing bodies over time.

3.3.1 Generating data on bodies through simulation

Another core functionality of the `Space` class was collecting data about the system such as the simulation time and values of gravitational potential and Kinetic energies again this would work in general by iterating over each body and determining these properties for each and then combining these together appropriately.

The object was also able to generate a prediction for the stable orbital period of a body by calculating the time difference between the body in question being in the same position to within a fraction of the position change between intervals. This accounted for both variation in a bodies path and non-simulated positions.

3.3.2 Launching from a body

The `Space` object allowed a body to be launched at the surface of another given a latitude and initial velocity. The body to be launched would be placed a distance slightly larger than the sum of radii of the launch object and body from their centers of mass to prevent an immediate collision.

3.4 Animating the produced data - The AnimateSpace Class

To produce an animated output of generated data the `AnimateSpace` class was used, this was able to parse the data produced by a `Space` object and output it in an animated format (through use of the `Matplotlib` library [6]). The class is able to run a simulation and immediately output it to the screen using the `liveSim` method or alternatively can be used to animate an existing data set as well as saving to file.

This came about due to it being easier to debug basic behaviour using the 'on the fly' method however this became slow for large datasets and would not allow for easily reviewing what had occurred in an animation hence the further methods were implemented.

3.5 Experimental Scripts to produce data

3.5.1 Constant Total Energy

A script was created that would simply run a selection of bodies from file over a given time period and output the energies as a graph as well as (as already required) outputting the data to a file for further analysis using spreadsheet software. This script provided a quick method for checking if energies were conserved after making changes to the implementation.

3.5.2 Probe Path From Earth to Mars

To determine a path between Earth and Mars a simple script was created to iterate over the various possible launch parameters (latitude, initial Y-axis velocity and initial X-axis velocity) and determine the closest position to Mars that was reached by the probe in a certain time frame as well as writing each result to file.

To add some efficiencies to an otherwise very basic data collection script as well as when meeting the decided stop time, the script would also stop simulating a certain launch parameter when the `_collision` property (discussed in section 3.2.2) of the probe was `True`.This generally meant that the probe had crashed into Earth or Mars (unfortunately more often and less usefully Earth) which were conditions where no more useful data could be collected.

This script could have been automated to rerun itself until optimal parameters were found however it was more simple to run it multiple times focussing on 'close' (i.e. smallest closest distance) values through interpretation of the data by hand than add this functionality.

By using this script many times, the range and intervals used could be reduced to a point where a few 'good' (valid time and within a close distance i.e. a few kilometres of Mars) paths generated. These could then be animated to see how they compared to known paths/each other.

Through running an 'on the fly' animation, the time to intercept could then be determined for each of the produced 'promising' results by stopping at intercept and getting the time from the output file.

4 Results & Analysis

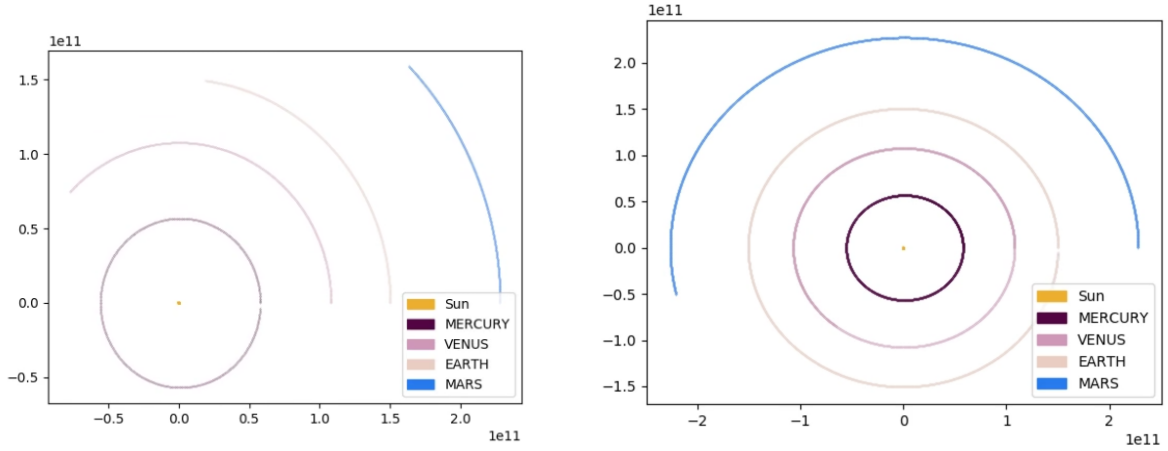
4.1 Basic Program Functionality

The classes were usable to generate data and output this in an animated form, examples of these are shown in fig.5.

These appeared to show the correct ratios between the different planets, fig.5a shows Mercury has reached its orbital period of 88days[10] whilst Venus is at around a third of its total orbital period (a Venus year is 243 Days[10]), hence the ratio of these two orbital periods is $\frac{88}{243}0.36$ or just over a third of the total journey which is the visible distance covered in fig.5a.

We can also see that the ratio of Earth (where a year is 365.25 days) to Mercury appears to be around a quarter, this matches the ratio of orbital periods of ~ 0.24.

We can see in fig.5b, Mars is about halfway through a Mars year. This is the expected result from its year 687 Days vs Earth's 365.25 Days (a ratio of ~ 0.53) [8]



(a) Animation just before completion of a Mercury year

(b) Animation just before an Earth year has passed

Figure 5: Graphic showing stills from an animation of the inner planets of the solar system

These visual ratios are also backed by the fact that the orbital periods of the planets matched their known values to within 5% for all values and to within 1% for the majority as seen in fig.6.

Planet	Known Orbital Period (Days)	Simulated Orbital Period (Days)	Percent error vs real values
MERCURY	88	85	3.41%
VENUS	225	223	0.89%
EARTH	365.25	368	0.75%
MARS	687	684	0.44%

Figure 6: Percentage error in known[10] and simulation produced values for orbital period

4.2 Conservation of total energy within the system

By creating a simulation that simulated an overall time of 2 years (with an interval of 1 hour) a collection of data points for the various energies in the system was generated. These were then plotted to determine if energy was conserved over time, this is shown in fig.7.

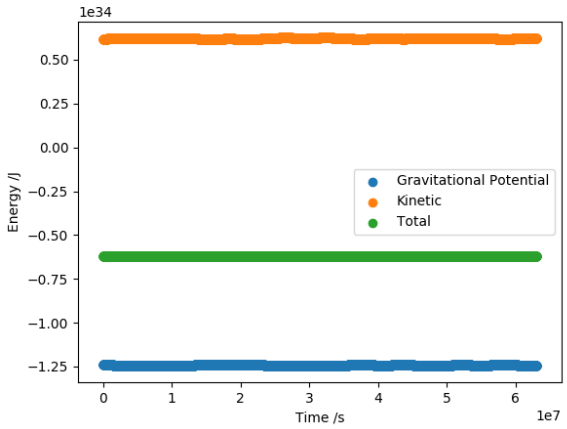
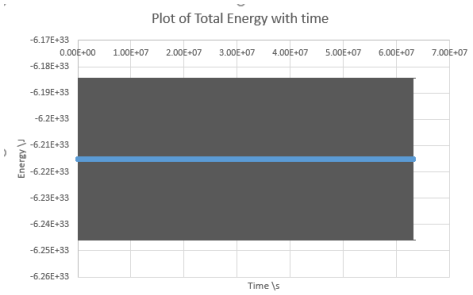
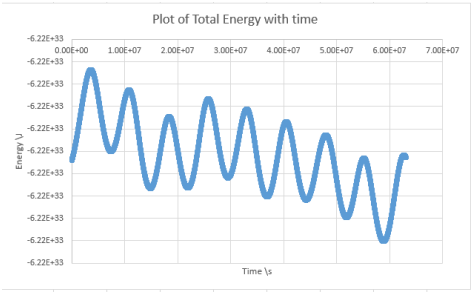


Figure 7: Comparison of energies over time produced with the inner planets

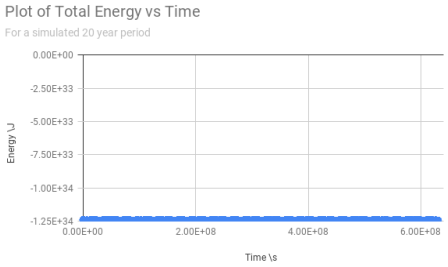
Using the outputted data with spreadsheet software the data could be better viewed as well as allowing determination of uncertainty for data as seen in fig.8a. An appropriate uncertainty for total energy was the propagation of uncertainties from Kinetic and Gravitational potential energies. The uncertainty in each of which was used as standard deviation of the data describing the dispersion in values and hence an appropriate uncertainty for the large data set. Looking closer at the data's trend (see fig.8b) without uncertainty suggested that it appeared that energy was decreasing with time, the uncertainty however made this trend less well founded and hence could be disregarded further verified by performing the test again over a larger time period (20 years) as seen in fig.8c.



(a) Graph showing total energy vs time with uncertainty



(b) Graph showing total energy vs time without uncertainty



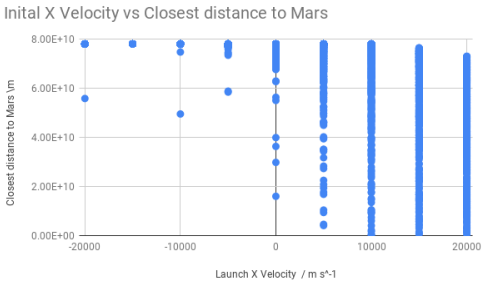
(c) Energy vs time over a 20 year period showing an overall constant value

Figure 8: Figures showing constant energy vs time, ran using the inner planets.

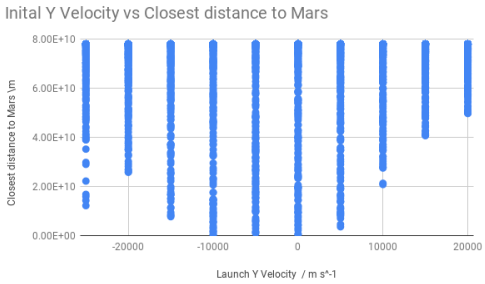
4.3 Plotting a path to Mars

The iterative approach of collecting a large collection of data for various latitudes and launch velocities as discussed in s.3.5.2 appeared to be somewhat successful, it was able to determine various 'promising' parameters that could then be animated to view how viable they were in producing a path to Mars.

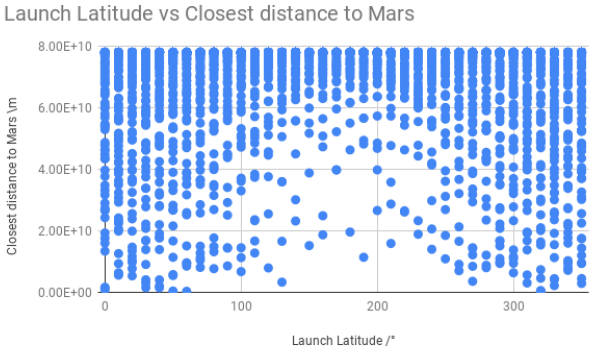
Initial results suggested that latitudes were key to determining the best path whilst various launch velocities effectively only added noise to results as seen in fig.9.



(a) Initial results on launch X-axis velocity vs closest distance to Mars



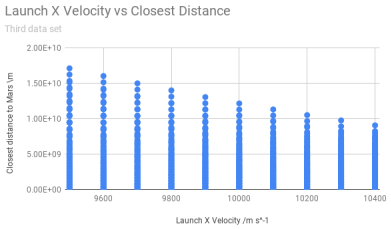
(b) Initial results on launch Y-axis velocity vs closest distance to Mars



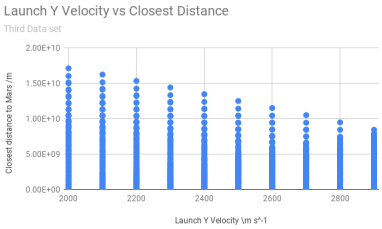
(c) Initial results on launch latitude vs closest distance to Mars

Figure 9: Effect of launch parameters on determining a path to Mars in the initial (and most general) data set collected

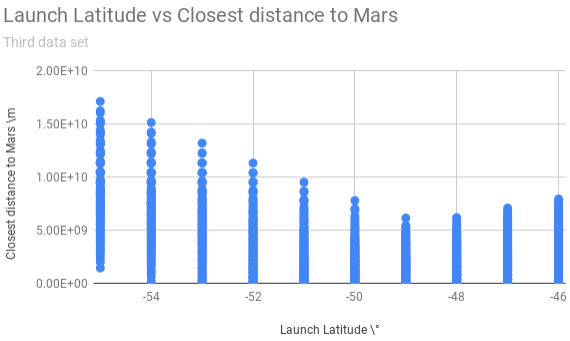
As the range of angles that were being tested was reduced launch velocities became more important in determining the proper path to Mars as seen in fig.10



(a) Trend in launch X-axis velocity vs closest distance. Visibly closest distances were where X was close to 2600ms^{-1}



(b) Trend in launch Y-axis velocity vs closest distance



(c) Trend in launch latitude vs closest distance to Mars

Figure 10: Trends in launch parameters after three reductions in the range and intervals used

The most realistic path produced was with the parameters:

Latitude	$-50°$
Initial X-axis velocity	10030ms^{-1}
Initial Y-axis velocity	2525ms^{-1}
Initial Speed	10343ms^{-1}

These related to intercept conditions of

Intercept distance	3.10km (i.e. a collision or landing)
Time to intercept	2.5 Months

This produced path is visualised in fig.11.

The results of this path are unrealistic when compared to velocities that would be reasonable for usage with a probe as the time taken to reach the target of Mars is much shorter than that of used paths such as that of the Viking mission which had times to intercept of around 9 months [9].

It is likely that the issue of an unrealistic orbit is due to usage of only the most obvious intercepting launch parameters, from the large collection of data produced by the initial run of the `MarsProbeExperiment` script, only 3 of $\sim$ 4000 data points were further investigated due to time constraints. It is likely therefore that investigation of other launch parameters would yield more valid results.

Because this parameters overall speed was less than that of the Earth's escape velocity $\sim 11\text{kms}^{-1}$[1] this would be supportive that this method would likely able to through further analysis produce a valid 'there and back again' path.

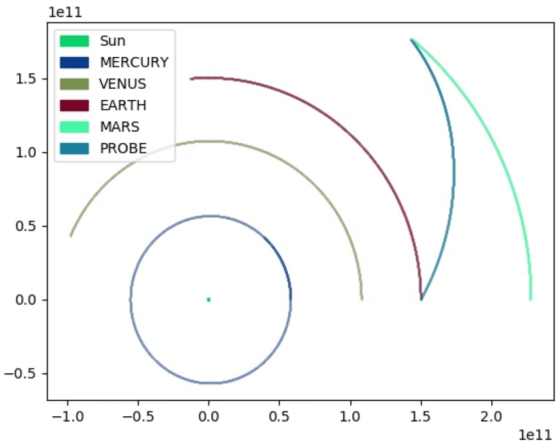


Figure 11: Visualisation of a probe launched with parameters shown in 4.3 .

5 Discussion on possible improvements

Overall this implementation was successful in simulating the movement of a 2D solar system. Improvements could have been added to the orbital period determination in ensuring it was more accurate across all values types of planets as higher velocity bodies were generally less realistic, as shown by Mercury in fig.6. This however, was an acceptable discrepancy for a planet which has a complex orbit [7].

The determination of probe paths could have been improved by considering a greater number of experimental values either through automation of the process or simply given greater time to allow for testing of more launch parameters from the produced data sets.

6 Conclusions of Experimentation

Overall this experimentation was able to produce results that would suggest a valid simulation of the inner planets of the solar system.

It has shown valid ratios between planet's orbital periods as well as energy conservation over time, as expected for a closed system.

It was also usable to produce paths between Earth and Mars for a probe and although the paths that were currently determined were not realistic for a probe, there was indication that a more realistic path could be generated given further time for analysis.

References

[1] Escape velocity. "http://hyperphysics.phy-astr.gsu.edu/hbase/vesc.html"[Accessed:2/4/2019].

[2] Codata internationally recommended 2014 values of the fundamental physical constants, 2014.

[3] Khan Academy. Gravitational attraction. "https://www.khanacademy.org/computing/computer-programming/programming-natural-simulations/programming-forces/a/gravitational-attraction"[Accessed:1/4/2019].

[4] Dr. Ross Galloway. Notes for physics1a. 2017/2018.

[5] Prof. Judy Hardy. Computer simulation course book. 2019.

[6] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

[7] Sam Jarman. "correcting einstein's calculation of the orbit of mercury". *Physics World*, 2018.

[8] NASA. Mars facts. "https://mars.nasa.gov/allaboutmars/facts/#?c=inspace&s=distance"[Accessed:1/4/2019].

[9] NASA. Viking mission overview, 2017. "https://www.nasa.gov/redplanet/viking.html"[Accessed:1/4/2019].

[10] Dr. David R. Williams. Planetary fact sheet - metric. "https://nssdc.gsfc.nasa.gov/planetary/factsheet/"[Accessed:2/4/2019].

A Further justification of implementation of simulating intervals method

As discussed in section 3.2.1 the current implementation does not copy the bodies before calculating acceleration meaning that some bodies use an updated position whilst others do not. This decision was taken as it provided the expected output in various forms as already discussed.

When testing the conservation of energy in the system it was found that when applying the copying of data before calculating the acceleration the variation was much larger and not covered by the uncertainty as shown in fig.12 further justifying the current implementation.

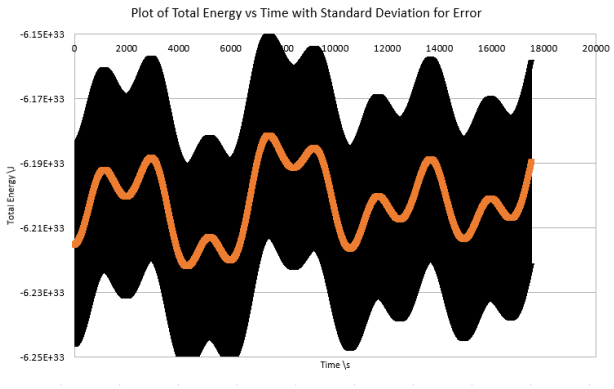


Figure 12: Total Energy output shows a much larger variation versus that of the non 'copy bodies' implementation that is shown in fig.8a over the same period