Computer Simulation Project

A simulation on terrestrial planets' motion, orbital period, energy conservation, and probe launching conditions

<s2084333>

April 1st 2021

Word Count: 2459

1. Introduction

This simulation investigated the gravitational planetary motion and their orbital periods, the conservation of the energy of the inner solar system, and explored possible initial conditions including initial velocity(both in direction and magnitude) and initial position, as well as the time taken for a hypothetical space probe mission to fly pass Mars.

The model used for the planetary motion is the Beeman integration scheme which determines the position, velocity, and acceleration of bodies at a given time with provided initial conditions of the system. This allows the entire system during a period of time after evolving from the initial conditions to be animated, as well as calculating the kinetic and potential energy of the system to check the energy conservation over time.

Lastly, the viable initial conditions for a probe launch was experimented in a prepared framework and a correlation was found between the initial speed and initial direction of the probe.

2. Methods

The following classes were implemented in the program.

class Model: this class holds information (in SI units) related to the initial condition of the system such as the initial position and velocity, as well as evolving the system using Beeman algorithm, while storing the calculated positions and energies to separate dictionaries.

- **Class Variables**

There are three public dictionaries to hold information. Using dictionaries allows for grouping information of a body by key words, and each body can be referred to using a key word which is its name. This made extracting specific information of a specific body readable and intuitive.

1. self.object_data

This is the dictionary to store the initial conditions which are being updated by the Beeman algorithm for each body of the system at each time step. This is where the most data modification happens.

2. self.position_dict

This is the dictionary to store the position of each body at every time step as lists, which holds the data that was used in animation, orbital period calculation and probe launching separate from the data that are being constantly modified.

3. self.energy_dict

This is the dictionary to store the kinetic, potential, and total energy of the whole system at each time step, it has the signature:

$$\{'\text{total\_energy(J)}': [], '\text{kinetic\_energy(J)}': [], '\text{potential\_energy(J)}': []\}$$

Similar to self.position_dict, it is used to separate the data needed for checking energy conservation and plotting from constantly modified data.

- **Data Reading**

self.__read_object_data

The initial conditions are read from file 'object_data.csv'. Using a csv file(comma separated values) making managing data easier. The data are clearly separated, easy to modify without much confusion, as well as able to directly read data into an OrderedDict while reading each line with csv.DictReader().

self.__read_object_data first read each line into an OrderedDict, then evaluate every value that's not a String using eval(). The OrderedDict is then saved into self.object_data, creating a signature like:

$$\{'\text{name1}': \{'\text{name1}': name, '\text{ position}': (x, y), '\text{ velocity}': (x, y),$$

$$'\text{current acceleration}': (x, y), '\text{ previous acceleration}': \dots, '\text{next acceleration}': \dots\}, '\text{name2}': \{\dots\}, \dots\}$$

Lastly, self.position_dict is saved with the following signature:

$$\{'\text{name1}': [initial\ position], '\text{name2}': [initial\ position], \dots\}$$

- **Beeman Algorithm**

self.__update_position, self.__calc_next_acc, self.__update_velocity & self.__update_accs

The Beeman Algorithm sequence is shown below:

1. Update the body's position at the next time step for each body

$$\vec{r}_i(t + \Delta t) = \vec{r}_i(t) + \vec{v}_i(t)\Delta t + \frac{1}{6}[4\vec{a}_i(t) - \vec{a}_i(t - \Delta t)]\Delta t^2 \quad [1]$$

This method takes Δt as an argument. The new position is updated in self.object_data and appended into self.position_dict as a tuple at this step of the sequence.

2. Calculate the body's acceleration at the next time step for each body

$$\vec{a}_i(t+\Delta t) = -G\sum_{j\neq i}\frac{m_j}{|\vec{r}_{ji}(t+\Delta t)|^3}\vec{r}_{ji}(t+\Delta t) \quad [1]$$

Where G is the gravitational constant. The 'next_acceleration' key in self.object_data is updated at this step.

3. Update the body's velocity at the next time step for each body

$$\vec{v}_i(t+\Delta t) = \vec{v}_i(t) + \frac{1}{6}[2\vec{a}_i(t+\Delta t) + 5\vec{a}_i(t) - \vec{a}_i(t-\Delta t)]\Delta t \quad [1]$$

This method takes Δt as an argument. The velocity of each body is updated at this step.

4. Update the previous and current acceleration

At this step, the previous acceleration is updated to the current acceleration, and current acceleration is updated to the next acceleration for each body.

After one iteration of sequence, the position for each body at the next time step is calculated and stored. Each step is wrote into separate methods rather than one, so other methods like energy calculation at each time step can be added into the whole iteration cycle instead of looping through the number of time step twice.

- **Energy Calculation**

self.__calculate_energies

The kinetic and potential energy for each body is calculated, summed to the total kinetic and potential energy of the system, and appended to self.energy_dict. The total energy is calculated by adding the total kinetic and potential, then appended.

The formula used are shown below:

$$K(t) = \sum_i \frac{1}{2}m_i v_i(t)^2 \quad [1]$$

$$U(t) = -\frac{1}{2}\sum_{i\neq j}\frac{Gm_i m_j}{r_{ij}(t)} \quad [1]$$

Where K is the total kinetic energy and U is the total potential energy.

- **Iteration Cycle**

self.update_iteration

The iteration cycle requires two arguments, the number of time steps and the length of each time step in seconds. It iterate through the energy calculation and the

Beeman algorithm at each time step. The reason for not supplying the number and length of time steps is because it allows flexibility in the time period of the simulation, and enable the possibility of methods that requires a variable time period.

class Simulation: This class is used for a graphic simulation of the evolving system, calculating the orbital period of bodies, the energy conservation of the system, and displaying the change of energies over time. The Model class is used here to generate results which are used for simulations.

- **Class Variables**

self.__display_data is used for storing information for simulation including the display radius of the Circle objects and the display colours. self.num_of_time_steps and self.time_step_length are used for producing the energy and position dictionary used for simulations.

- **Data Reading**

self.__read_time_step_settings & self.__read_ display_data

Similar to reading data in class Model, csv files are used to store values with the same benefits mentioned previously.

The signature of self.__display_data is similar to self.object_data in class Model:

$\{'\mathrm{name1}': \{'\mathrm{name1}': name, '\ \mathrm{display\ radius}': n, '\ \mathrm{display\ color}': \mathrm{r}\}, '\mathrm{name2}': \{\ldots\}, \ldots\}$

The number and length of time steps are simply read and stored as class variables.

- **Orbital Period**

self.__find_orbital_period

Instead of checking the sign change of the body's position vector which relies on the initial position of bodies to be effective, the orbital period is approximated using the angle between the initial position vector and the current position vector of each object. The angle between two vectors can be calculated as follows:

$$angle = \arccos\left(\frac{\vec{a} \cdot \vec{b}}{|\vec{a}| \cdot |\vec{b}|}\right)$$

By checking when has the angle become sufficiently close to 0, the orbital period can be determined regardless of the initial position vector of each body. In the code, this angle has been set to 0.25 degrees or π/360 radians. However, after the body has just left its initial position, the current angle would be also close to 0.

Thus, a counter is implemented in the code, once the angle has got close enough to π radians, the counter changes to indicate a half orbit has been committed. The next time the angle between position vectors is almost 0, an orbit would have been completed. In this way, the problem with initial angle can be mitigated.

- **System Animation**

self.__ix, self.__animate_func & self.display_simulation

The Circle objects used for animating is first initialized and appended into a class variable list. The animate function will update the centre of each Circle objects at once and return the list.

But, to access the position list of a body(self.model.position_dict[key]), the corresponding key of that body is required. Since each Circle object's index in self.__patch are corresponded to the key positioned at the same index in self.model.position_dict, a method that takes the index of a given list and returns the key place at the same index(self.__ix) resolved this issue.

The animation is done with FuncAnimation with an interval of 0.2, which produce a faster and smoother animation.

- **Energy Writing & Plotting**

self.write_te_to_file & self.display_energy_graph

Once every one tenth of number of time steps have passed, the total energy at that time step will be written to file 'TotalEnergy.txt'.

A more complete plot of the energies over time is produced by plotting each type of energy at a given time step as a point over the total number of time steps. The result after multiple different configurations is shown in the Result section.

class Satellite: This class specifically explores possible initial conditions of a probe launch in a fixed framework.

- **State Checking**

self.probe_state

This method determines whether the probe has reached Mars and would it return to Earth by checking the distance between the probe and Mars/Earth at each time step and a counter is set to determine the result of the launch.

If there is one time step where the distance between the probe and Mars is within a given range, it would calculate the time taken for the probe to travel to Mars, the time difference between this probe launch and NASA's Viking 2 probe's travel time, and change the counter from 0 to 1 which indicates this simulated launch reached Mars.

Similarly, after the counter has changed to 1, if the distance between the probe and the Earth is within a given range, the counter will be changed to 2, meaning the probe will travel back to Earth during the given period of time.

- **Probe Launching**

self.__probe_launch

This method checks over a given range of initial speeds and launching the probe with each initial speed at a given angle to find every viable initial speed that would reach Mars within said speed range.

At each iteration, the initial speed is rotated with the given angle, then a simulated launch happens with the probe's initial velocity modified to this rotated vector. With the calculated positions of the probe, use self.probe_state to check whether the probe has successfully completed the objectives in this launch.

3. Results

1.3 Project task

The default initial conditions used here are set according to NASA's Planetary Fact Sheet[2], the csv file containing the default values is in the project folder. The length of each time step is 3600 seconds and the total number of time steps is set to 17520 in this simulation. After multiple runs, the planets exhibited a stable orbit consistently:

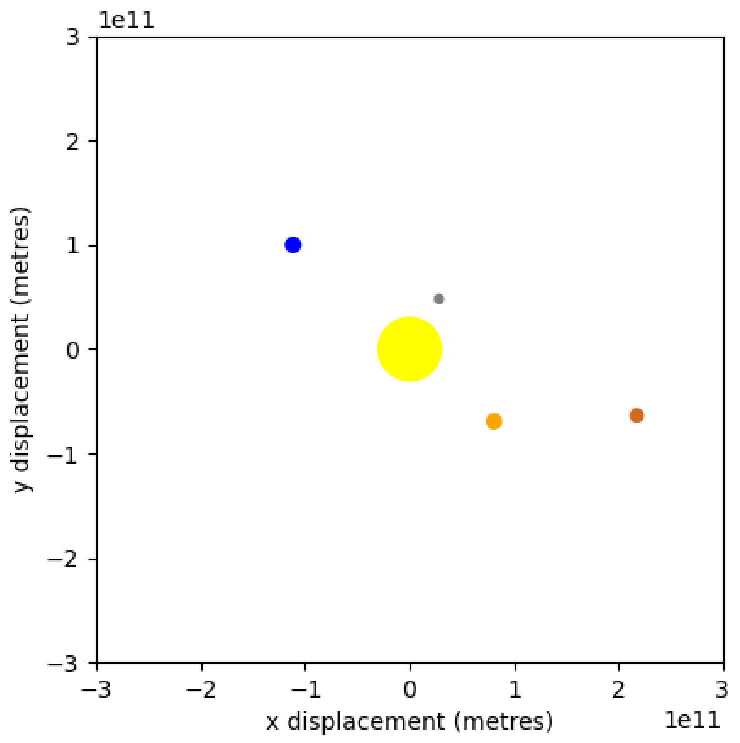

Figure 1 The simulated planets orbiting

The orbital period of each planet has a slight deviation from the actual period, the maximum percentage error is 4.15%. The determined Earth year length is 365.2 days in this simulation, and the actual orbital periods are calculated using data from Planetary Fact Sheet[2], as shown in the table below:

Planet	Actual orbital period (Earth year)[1]	Simulated orbital period (Earth year)	Percentage Error
Mercury	0.241	0.231	4.15%
Venus	0.615	0.604	1.79%
Earth	1.00	0.999	0.1%
Mars	1.881	1.855	1.38%

Table 1 A comparison of simulated and actual orbital periods

The total energy of the system has negligible fluctuations, as shown below:

Number of time step	Total energy at time step (1e33J)	ΔE_t (1e33J)
1752	-6.2697876	0
5256	-6.2697874	0.0000002
8760	-6.2697874	0
12264	-6.2697876	-0.0000002
15768	-6.2697876	0

Table 2 The fluctuations of the total energy of the system

The system has shown a lower initial total energy at time step 0 as much as 0.05% compared to the total energy at other time step. This is likely because of the initial acceleration of the system is set to (0, 0) for each body, the sudden increase in acceleration due to strong gravitation could have been the reason behind this phenomenon.

Overall, the simulation is fairly accurate considering the low percentage error in orbital period as well as stable total energy of the system.

1.4.1 Energy Conservation

After multiple runs of the program with the same initial condition mentioned above, the following graph has also been produced:

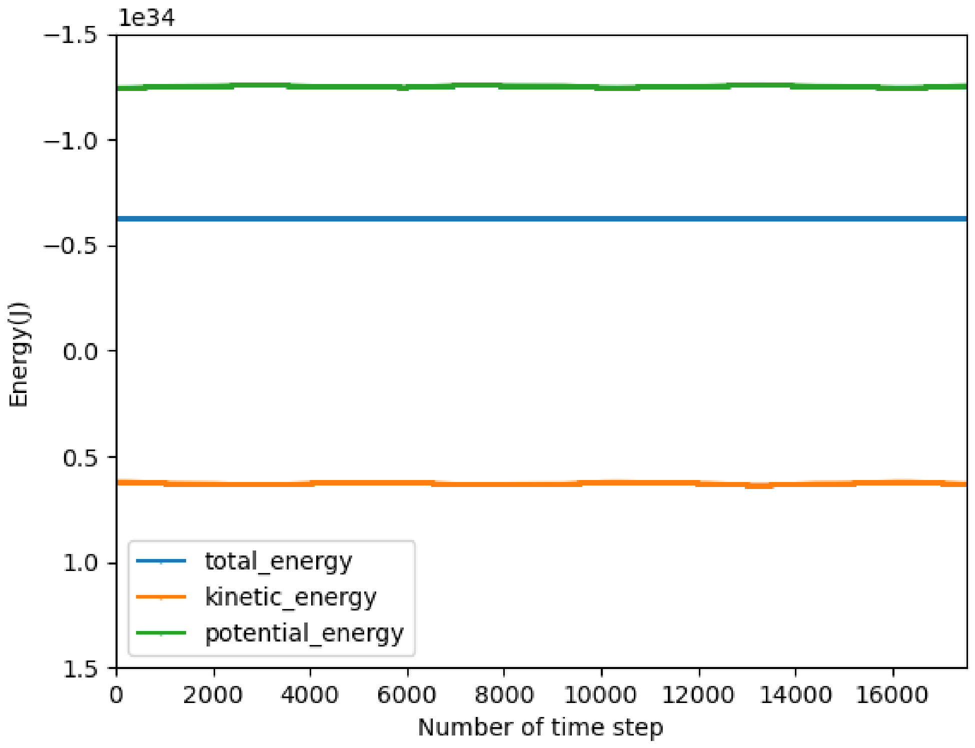

Figure 2 Total, kinetic, and potential energy of the system

The result has shown a consistency in the total energy in the system, while kinetic and potential energy fluctuating minimally over each time step. This shows the energy of the system is conserved over time, which support the statement “The Beeman algorithm is a symplectic integrator, which means that the total energy of the system should be conserved over time.”[3]

1.4.2 Satellite to Mars

The experiments investigated viable initial launching conditions a simulated space probe mission. The probe is assumed to have left Earth’s surface and ready to escape the gravitational pull, it is considered that the initial position is 1000 km above ground which is still in LEO(Low Earth Orbit) range[4]. The range for probe’s initial magnitude of velocity is between 20km/s to 37km/s, as the initial speed can reach 37km/s in a gravity assisted launch as shown in the following graph:

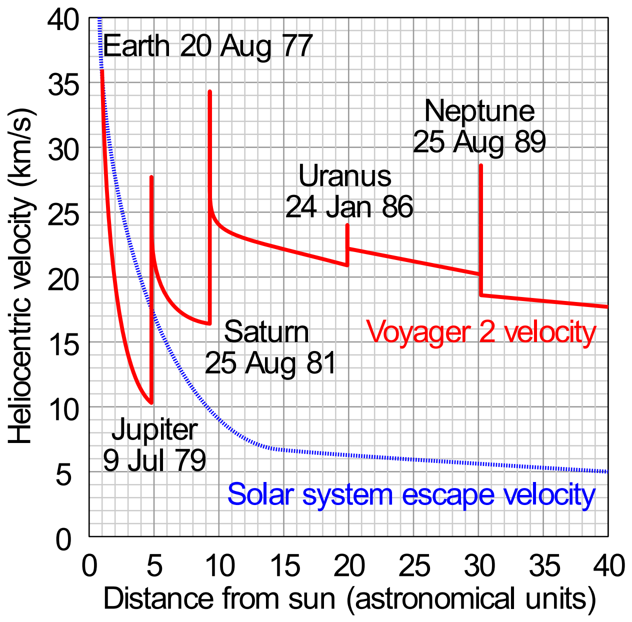

Figure 3 Plot of Voyager 2′s heliocentric velocity against its distance from the Sun[5]

The initial position of Earth is set to be (1.50e11,0) and Mars is set to be (0,-2.28e11), exactly 90 degrees apart rotationally. The probe has been launched at position (1.5e11,-7.378e6) with a range of angles from -15 to -75 degrees. The vertical initial position is determined by adding 1000km onto the radius of Earth: 6378km[2].

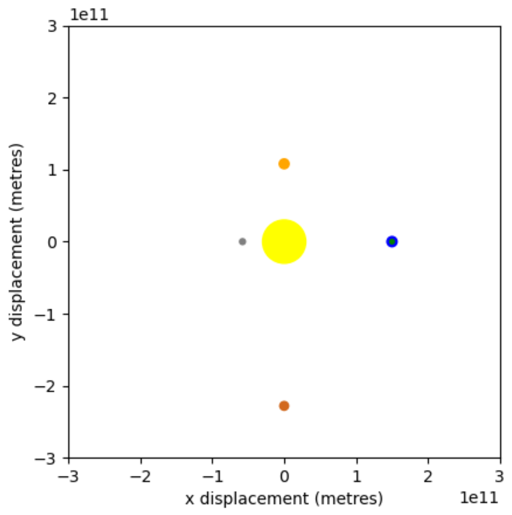

Figure 4 The initial system configuration for probe launching

The range of distance of which it's considered the probe has reached Mars as well as return to Earth is between 300km and 45000km, it is based on the Mars Orbit Insertion distance of the Reconnaissance Orbiter[6].

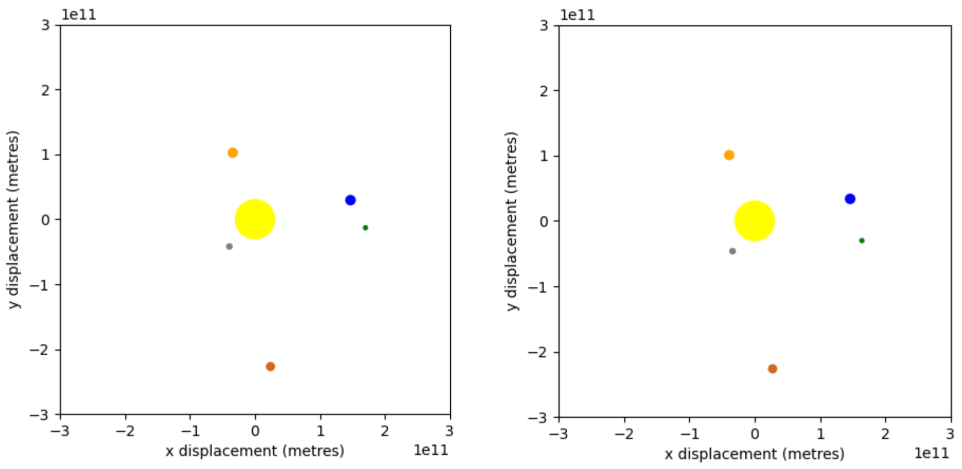

Figure 5 Probe(green) has just left Earth launching at 30(left) and 60(right) degrees

And the following viable initial conditions are found:

Initial launching speed (km/s)	Initial launching angle (degrees)	Time taken in space flight to reach Mars (days)	Time difference with Viking 2 probe (days)	The probe returned to Earth (Y/N)
26.406	15.52	141	192	N
26.534	30.00	119	214	N
27.769	44.83	100	233	N
30.853	59.82	84	249	N
36.764	74.77	68	265	N

Table 3 Viable initial launching conditions

By plotting the data above into a graph:

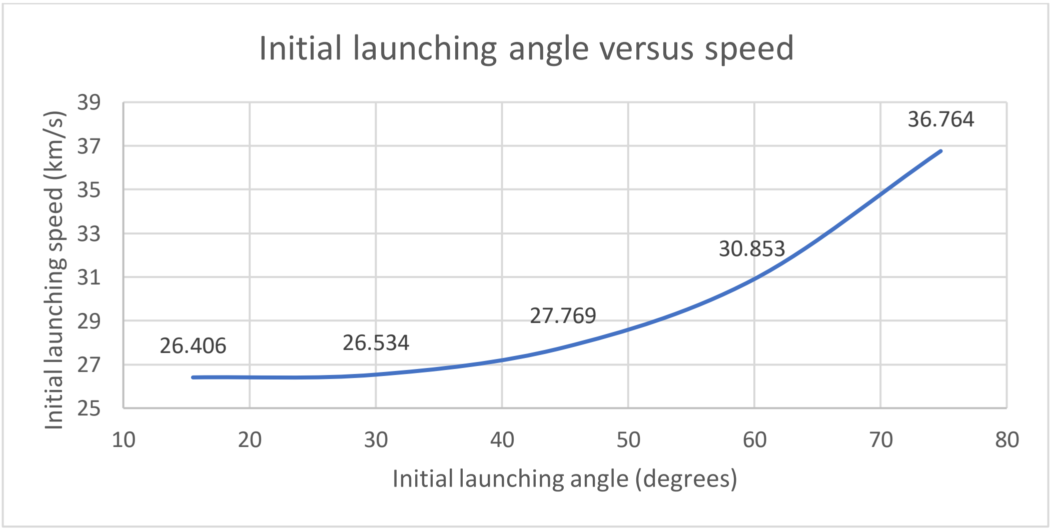

Figure 6 Initial launching angle versus speed

As demonstrated in the figure, there is a clear exponential relationship between the initial speed required and the initial angle. A modified plot allows the relation to be more linear:

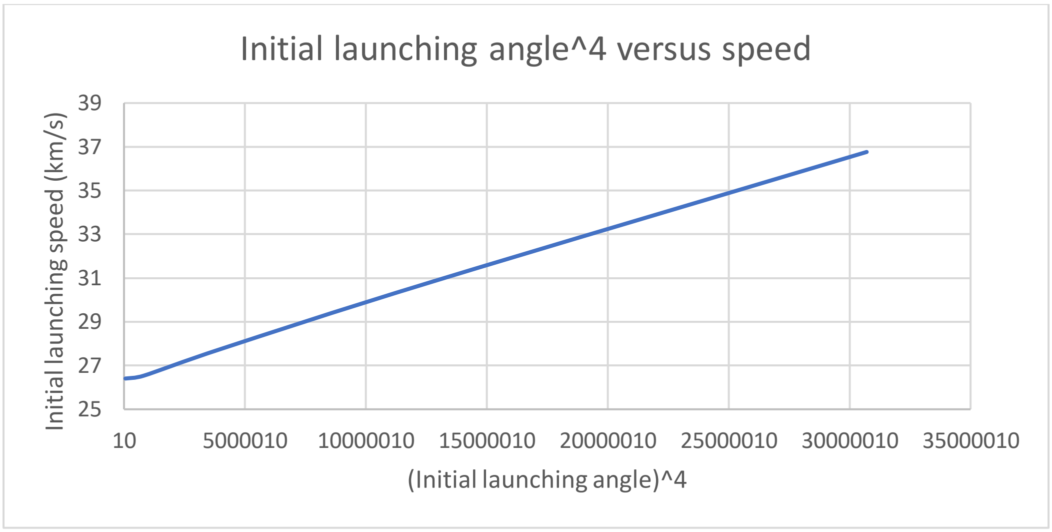

Figure 7 Initial launching angle^4 versus speed

Then the tangent line can be calculated:

$$m = \frac{36.764 - 30.583}{74.77^4 - 59.82^4} = 3.2 \times 10^{-7}$$

$$c = 36.764 - 74.77^4 \times 3.2 \times 10^{-7} = 26.751$$

$$initial\ launching\ speed = 3.2 \times 10^{-7} initial\ launching\ angle^4 + 26.751$$

Thus, a relationship is found between the initial launching speed and angle given that the initial positions of bodies is the same as mentioned above.

4. Conclusion

There are many possibilities for improvement: adding better features to estimate orbital periods, the efficiency of automatically finding viable initial conditions for probes needs significant enhancement, etc.

The goal of investigating the inner solar system, orbital periods, energy conservation, and possible probe launching conditions are met, to an extent.

The final implementation of the program created a fairly accurate model with its total energy conserved which simulates the position of bodies in the system over time graphically and a relationship between the initial launching speed and angle of the probe in a limited setting is found and mathematically modelled.

References

[1] “compsim07.pdf.” University of Edinburgh.

[2] “Planetary Fact Sheet.” Nasa.gov, 2018, nssdc.gsfc.nasa.gov/planetary/factsheet/.

[3] “Project.” Course Material, www2.ph.ed.ac.uk/AardvarkDeployments/Protected/92478/views/files/Python/Deployments/CompSimLearn2021/deploymentframeset.html#tree=Python:Organisations:CompSimNotes. Accessed 1 Apr. 2021.

[4] Wikipedia Contributors. “Low Earth Orbit.” Wikipedia, Wikimedia Foundation, 2 Nov. 2019, en.wikipedia.org/wiki/Low_Earth_orbit.

[5] Cmglee. Plot of Voyager 2′S Heliocentric Velocity against Its Distance from the Sun, 30 May 2011, en.wikipedia.org/wiki/List_of_artificial_objects_leaving_the_Solar_System#/media/File:Voyager_2_velocity_vs_distance_from_sun.svg. Accessed 1 Apr. 2021.

[6] mars.nasa.gov. “Mars Orbit Insertion.” Mars.nasa.gov, mars.nasa.gov/mro/mission/timeline/mtmoi/. Accessed 1 Apr. 2021.