Investigating if the Beeman Algorithm is Symplectic and Determining the Optimum Transfer Orbit for a Satellite

Giorgio Bruni Campanella

Computer Simulation Project Report

Thursday, 1st of April 2021

Abstract

The symplectic nature of the Beeman algorithm was investigated and the initial velocity required for a satellite to intercept Mars from Earth was determined. This was achieved by using a Python script that simulates an orbital system for given input parameters. The results show that the Total energy of the Solar system's first four planets is conserved and was determined to be $-6.1402577 \cdot 10^{33}$ J. The initial velocity required to create a transfer orbit to allow a satellite to intercept Mars form Earth was determined to be 32804.17 ms^{-1}.

1 Introduction and theory

For an N-body system where there are multiple orbiting bodies, the velocity and position vectors of the orbiting bodies can not be found by analytically integrating the equation of acceleration. Instead, for such cases, a numerical integration method can be used to find the position and velocity vectors for every body in the system. One such method is the Beeman's Algorithm. The Beeman's Algorithm is a symplectic numerical integrator which can be used in conjunction with an equation for acceleration to find the position and velocity of the bodies for a given timestep. The Beeman's algorithm is given by the equations

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

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

where for a body; $\vec{r}(t)$ is the current position vector, $\vec{v}(t)$ is the current velocity vector, $\vec{a}(t+\Delta t)$ is the acceleration vector for the new timestep, $\vec{a}(t)$ is the current acceleration vector and $\vec{a}(t-\Delta t)$ is the previous acceleration vector [1].

To find the acceleration for each timestep, Newton's second Law, Newton's Law of Gravitation and the Law of Superposition are used to generate an equation that sums all the accelerations for each pair of bodies. This equation is given by

$$\vec{a}_j(t) = G \sum_{i \neq j} \frac{m_i}{|\vec{r}_{ij}(t)|^2} \hat{r}_{rj}(t) \tag{3}$$

where $\vec{r}_{ij}$ is the position vector between the two bodies, $|\vec{r}_{ij}|$ is the magnitude of the $\vec{r}_{ij}$ vector, G is the gravitational constant and m_i is the mass of the other body.

The Total Kinetic energy of the system can be found by summing the kinetic energy of all the bodies in said system. As a result, the total kinetic energy of the system is given by

$$K(t) = \sum_i \frac{1}{2} m_i v_i^2(t) \tag{4}$$

Similarly, the gravitational potential energy of the system can by found by summing the potential energy between each pair of bodies. As a result, the total Potential energy of the system is given by

$$U(t) = -\frac{1}{2} \sum_{i \neq j} \frac{G m_i m_j}{r_{ij}(t)} \tag{5}$$

where m_i and m_j are the masses of the pairs and $r_{ij}(t)$ is the distance between them.

Experiment 1:

The Beeman algorithm is what is know as a symplectic integrator. As a result, it is expected that throughout the calculations of the position and velocity vectors, the total energy of the system will be conserved. By using a Python script that simulates the Solar system using the Beeman algorithm, the resulting Total energies of the system will be investigated to see if they are conserved throughout the simulation.

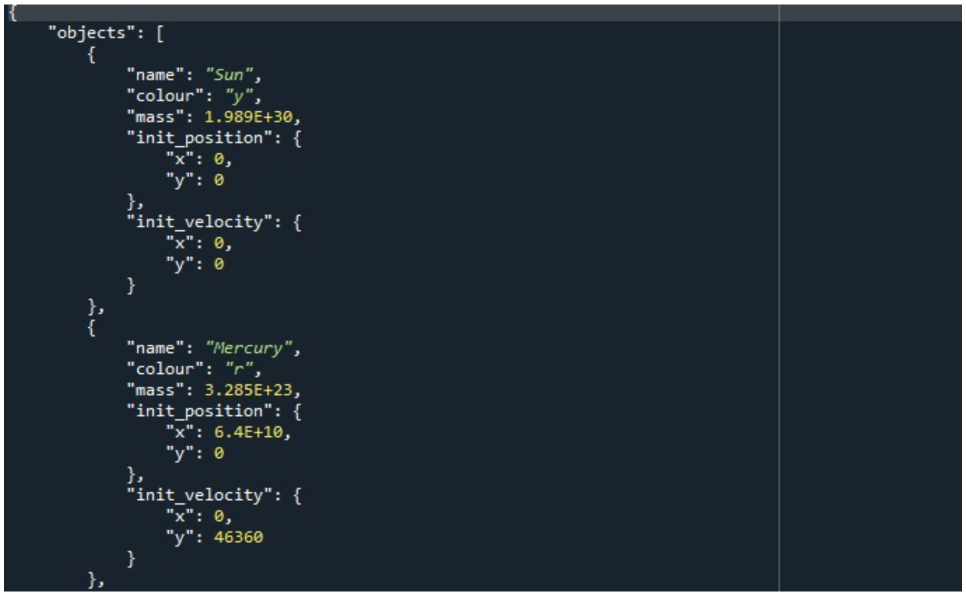
Experiment 2:

Hohman transfer orbits are commonly used to create an intercept between two orbiting bodies. By using the Python Solar system simulation, the optimal initial velocity for a satellite will be determined so that from the surface of the Earth, a transfer orbit is created that results in an intercept between the satellite and Mars.

2 Methods

The Python script for the Orbital motion simulation is split up into two main classes: Body and Simulate. The Body class contains all the methods relating to reading data from external files and creating 'Numpy' arrays, while the Simulate class contains the code that governs the simulation of the bodies(calculating changes in parameters and visualisation). The Orbital motion simulation works by reading the input parameters of the bodies from a .JSON file and constructing 'Numpy' arrays for it's respective parameter. Using these 'Numpy' arrays and the Beeman Method, the relevant calculations for the current timestep are conducted and the resulting position vectors for each body are used to output a frame for an animation. The total energy of the system for each timestep is also calculated and written out to a separate .txt file. This process is then repeated for each new timestep, creating an animation of the orbits. For this specific simulation script, 'On the Fly' animation was used as it allowed for easier debugging and data collection as the output data corresponded to what was happening in the animation.

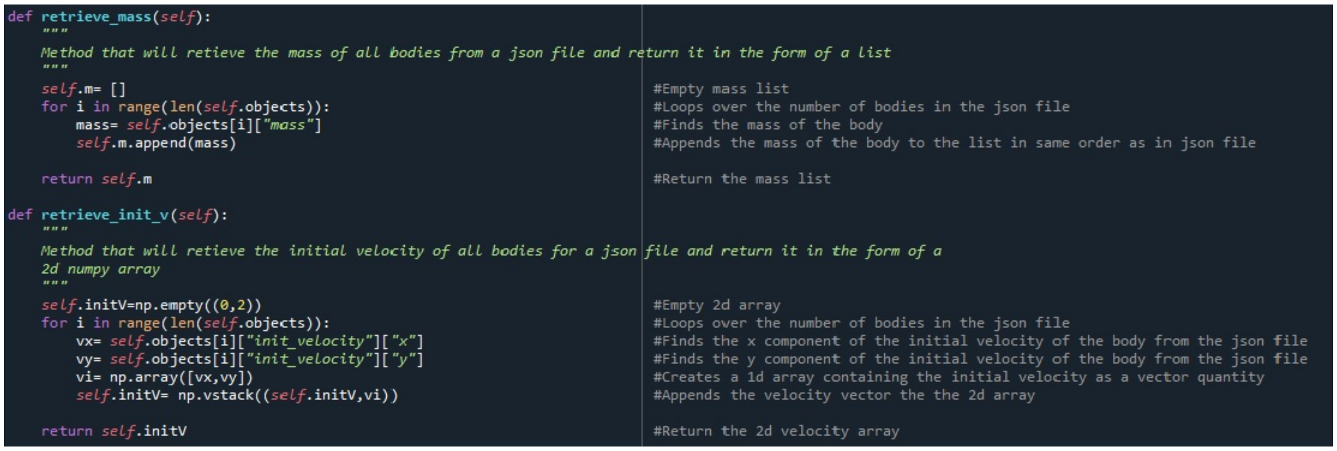
A JSON file was used to store the input parameter as it allowed for each parameter to be categorised, permitting more elegant methods of data retrieval in the simulation code. It also results in the easier addition of new input parameters or a new body. An example of JSON file used can be seen in Figure 1.



```json
{
    "objects": [
        {
            "name": "Sun",
            "colour": "y",
            "mass": 1.989E+30,
            "init_position": {
                "x": 0,
                "y": 0
            },
            "init_velocity": {
                "x": 0,
                "y": 0
            }
        },
        {
            "name": "Mercury",
            "colour": "r",
            "mass": 3.285E+23,
            "init_position": {
                "x": 6.4E+10,
                "y": 0
            },
            "init_velocity": {
                "x": 0,
                "y": 46360
            }
        },
```

Figure 1: Image of the JSON file for input parameters.

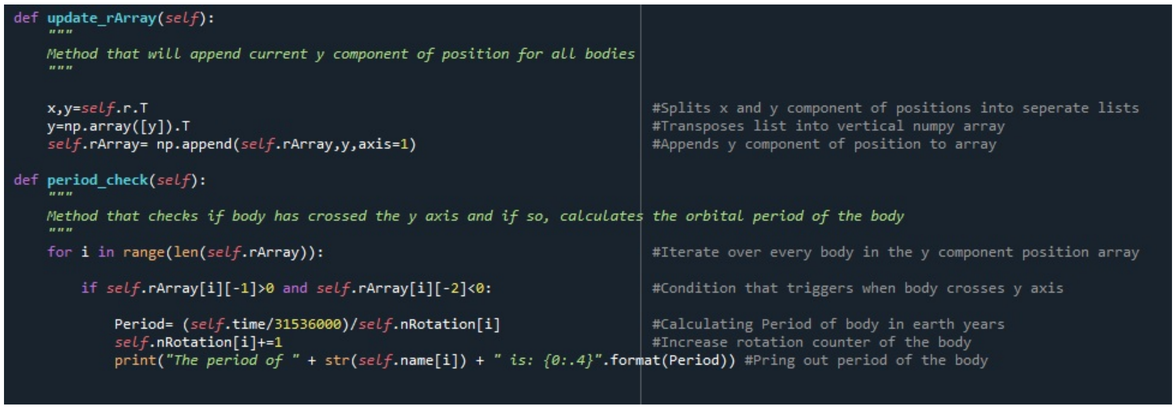
All the data of the bodies are stored either in the form of a: 1D 'Numpy' array of Dimension (N) (if data is a scalar or a string) or a 2D 'Numpy' array of dimensions (Nxn) (if data is a vector), where n is the number of dimensions for the vector and N is the index that refers to which body the row of the array corresponds to. This method of storing the parameters allows for all the data of one parameter to be stored in the same array, reducing and simplifying the number of variables in the code. The arrays are constructed by reading each specified value for the entire Object in the JSON file and appending it to an empty array. An example of which is shown in Figure 2.



```python
def retrieve_mass(self):
    """
    Method that will retieve the mass of all bodies from a json file and return it in the form of a list
    """
    self.m= []                                              #Empty mass list
    for i in range(len(self.objects)):                      #Loops over the number of bodies in the json file
        mass= self.objects[i]["mass"]                       #Finds the mass of the body
        self.m.append(mass)                                 #Appends the mass of the body to the list in same order as in json file

    return self.m                                           #Return the mass list

def retrieve_init_v(self):
    """
    Method that will retieve the initial velocity of all bodies for a json file and return it in the form of a
    2d numpy array
    """
    self.initV=np.empty((0,2))                              #Empty 2d array
    for i in range(len(self.objects)):                      #Loops over the number of bodies in the json file
        vx= self.objects[i]["init_velocity"]["x"]           #Finds the x component of the initial velocity of the body from the json file
        vy= self.objects[i]["init_velocity"]["y"]           #Finds the y component of the initial velocity of the body from the json file
        vi= np.array([vx,vy])                               #Creates a 1d array containing the initial velocity as a vector quantity
        self.initV= np.vstack((self.initV,vi))              #Appends the velocity vector the the 2d array

    return self.initV                                       #Return the 2d velocity array
```

Figure 2: Image of method that constructs the mass and velocity arrays.

After the new position array was calculated for every timestep, a vertical array containing the y component of the new position was appended to a (Nxm) array (where m is the number of timesteps that have occurred in the simulation) containing all the y coordinate throughout the simulation. This allows for a quick and easy way to access the current and previous values of the y component of position for any body without having to worry about the overwriting of the class variable for new position. With this information, the code can check if a body crosses the y axis and therefore, when it completes a full rotation. If this is the case, the orbital period is then calculated and printed to the console of the IDE. The code responsible for this is shown in Figure 3.



```python
def update_rArray(self):
    """
    Method that will append current y component of position for all bodies
    """

    x,y=self.r.T                                            #Splits x and y component of positions into seperate lists
    y=np.array([y]).T                                       #Transposes list into vertical numpy array
    self.rArray= np.append(self.rArray,y,axis=1)            #Appends y component of position to array

def period_check(self):
    """
    Method that checks if body has crossed the y axis and if so, calculates the orbital period of the body
    """
    for i in range(len(self.rArray)):                       #Iterate over every body in the y component position array

        if self.rArray[i][-1]>0 and self.rArray[i][-2]<0:   #Condition that triggers when body crosses y axis

            Period= (self.time/31536000)/self.nRotation[i]  #Calculating Period of body in earth years
            self.nRotation[i]+=1                            #Increase rotation counter of the body
            print("The period of " + str(self.name[i]) + " is: {0:.4}".format(Period)) #Pring out period of the body
```

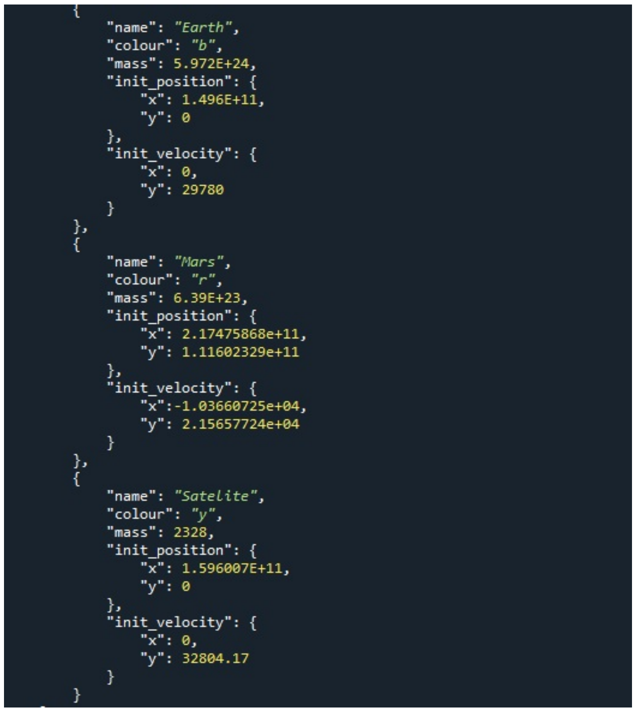
Figure 3: Image of method that appends y component of position and calculates the orbital period of bodies.

Experiment 1:

The initial input parameters for the Sun and the first four planets of the Solar system were placed into the JSON file (as seen in Figure 1). The initial parameters of mass, radius and velocity were sourced from the NASA Factsheet website [2]. The simulation was then allowed to run for $\simeq 1$ Earth year and then stopped. The Total energy output folder was then opened in 'Excel', stored for data analysis and cleared. This process was then repeated with varying values of Δt.

Experiment 2:

The JSON file for Experiment 1 was modified to include the parameters of a satellite being launched from Earth. The mass of the satellite used for this experiment was the mass of Viking I [3] and the radius was set so that it would correspond to the surface of the Earth. The initial position of Mars was then changed so that it would be 44° ahead of the initial position of Earth. This is so that when the simulation starts Mars, Earth and the satellite are positioned in the most optimal low-energy transfer window [4]. An example of the input parameters used for this experiment are shown in Figure 4.



```
    {
        "name": "Earth",
        "colour": "b",
        "mass": 5.972E+24,
        "init_position": {
            "x": 1.496E+11,
            "y": 0
        },
        "init_velocity": {
            "x": 0,
            "y": 29780
        }
    },
    {
        "name": "Mars",
        "colour": "r",
        "mass": 6.39E+23,
        "init_position": {
            "x": 2.17475868e+11,
            "y": 1.11602329e+11
        },
        "init_velocity": {
            "x":-1.03660725e+04,
            "y": 2.15657724e+04
        }
    },
    {
        "name": "Satelite",
        "colour": "y",
        "mass": 2328,
        "init_position": {
            "x": 1.596007E+11,
            "y": 0
        },
        "init_velocity": {
            "x": 0,
            "y": 32804.17
        }
    }
```

Figure 4: Image of the JSON file for input parameters.

The simulation was then run while printing the distance between the satellite and Mars multiple times, varying the initial velocity of the satellite by hand each time until the simulation animation showed a close intercept between the two bodies.

3 Results and Analysis

Experiment 1:

Using the timestep for the simulation, the time for each of the data points for the Total energy of the system were calculated. Using the data points, a mean value of the Total energy was determined and an error for these values was given by $\delta = \sigma/\sqrt{N}$, where N is the number of measurements.

For the simulation using $\Delta t = 77500s$, the values for Total energy varied between $-6.14020 \cdot 10^{33}$

J and $-6.14230 \cdot 10^{33}$ J; meanwhile for the simulation using $\Delta t = 100000s$, the values for Total energy varied between $-6.13940 \cdot 10^{33}$ J and $-6.13957 \cdot 10^{33}$ J. The mean value for Total energy of the system was determined to be $-6.1402577(9) \cdot 10^{33}$ J. Plots of the variation of Total energy of the system are shown in Figures 5 and 6.

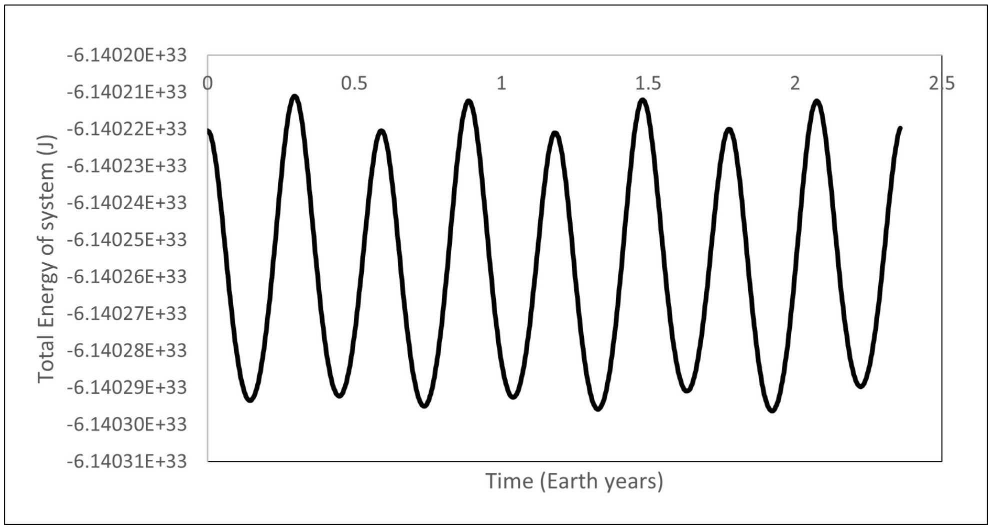


Figure 5: Plot of Total energy of system vs time for timestep $\Delta t = 77500s$

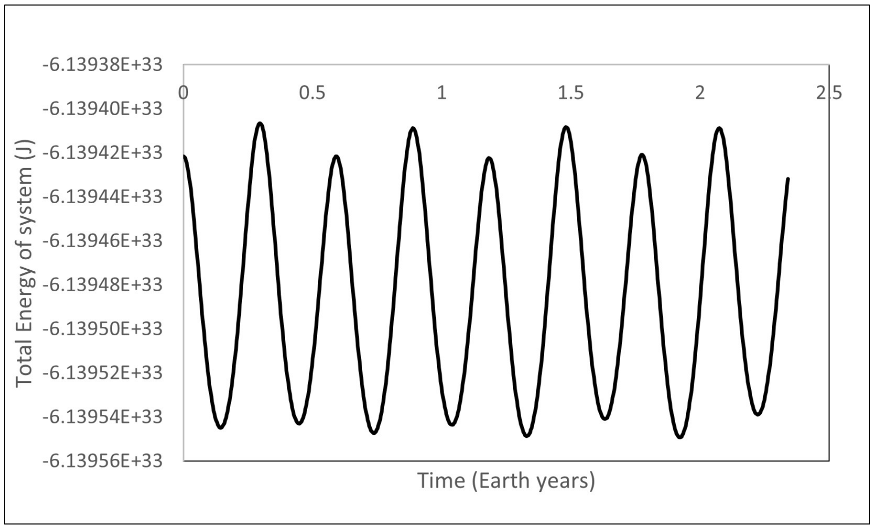


Figure 6: Plot of Total energy of system vs time for timestep $\Delta t = 100000s$

Experiment 2:

From several runs of the simulation while manually varying the initial velocity of the satellite, it was found that if the satellite is given an initial velocity of $32804.17ms^{-1}$, the satellite will intercept Mars as it approaches the perihelion of it's first orbit. The time taken between initial launch of the satellite and it's intercept is 1.24 Earth years and the closest distance between the satellite and the surface of Mars is 50990 Km. By subtracting the velocity of Earth from the initial velocity of the satellite, it was determined that to achieve this transfer orbit, the rockets of the satellite must provide a 'velocity boost' of $3024.17ms^{-1}$. If the satellite were to continue on this transfer orbit, it was determined that after 9.1 Earth years, it would have an intercept with Earth.

4 Discussion

Experiment 1:

The experimentally determined value for the Total energy of the first four planets of the solar system using the Beeman algorithm was found to be $-6.1402577(9) \cdot 10^{33}$ J for a timestep of $\Delta t = 77500s$. This is similar to the literature value for the total energy of this orbital system, which is $-6.19819 \cdot 10^{33}$ [5]. However, the experimentally determined error tells us that the experimental value is 64366σ from the literature value. This is clearly wrong, and therefore indicates that the experimental error for the total energy is not appropriate.

Figures 5 and 6 show that while the value for the Total energy oscillate about a certain value, the mean total energy stays constant throughout time. This indicates that although the Beeman algorithm may not be incredibly accurate, it is a symplectic numerical integration method and therefore can be used to model orbital motion in simulations.

Another detail that can be seen in Figures 5 and 6 is that as the value for the timestep decreases, the amplitude of oscillations for the Total energy also becomes smaller, which would point to the finding that the smaller the value of the timestep in the simulation, the more accurate the Beeman algorithm would be, and therefore, the more accurate the results from the simulation are.

The inaccuracies of the Beeman algorithm are consistent with what would be expected, as during the derivation for equations (1) and (2), higher order terms are dropped, resulting in the calculated values from the Beeman method having an inherent error. As a result, while a decent method for finding the Total energy over a long period of time, the Beeman algorithm is a poor way of finding the instantaneous energy of the system [6].

In order to improve the results from this experiment, it would be recommended to run the simulations using extremely low values for timestep. While this may have the down side of having the simulation take exponentially longer to complete and having more data to process, the error in the Total energy of the system will be mitigated for the most part.

Experiment 2:

The experimentally determined initial velocity required to create a transferred orbit from Earth to Mars for a satellite was found to be $32804.17ms^{-1}$, for the journey to take 1.24 Earth years, for the closest point of the flyby be 50990 Km and for the satellite to have an intercept with Earth after 9.1 Earth years.

While ultimately, the closest distance between the satellite and Mars was not close enough for the gravitational pull of Mars to provide a slingshot effect on the satellite, the distance is close enough that a stable orbit around Mars can be achieved with small 'rocket bursts', and therefore, is a satisfying result for this experiment.

When comparing to the actual satellite the simulation was based on (i.e.: Viking I) to the simulation, the time for the real Viking I took approximately 1.09 Earth years to reach Mars, while the simulated satellite took 1.24 Earth years. However, this discrepancy in time is to be expected, as the transfer orbits of the two satellites intercepted Mars at different points. Viking I intercepted Mars as it approached the aphelion of it's first orbit, while the simulated satellite intercepted Mars as it approached the perihelion of it's first orbit [7]. A diagram of the different orbit paths is shown in Figure 7.

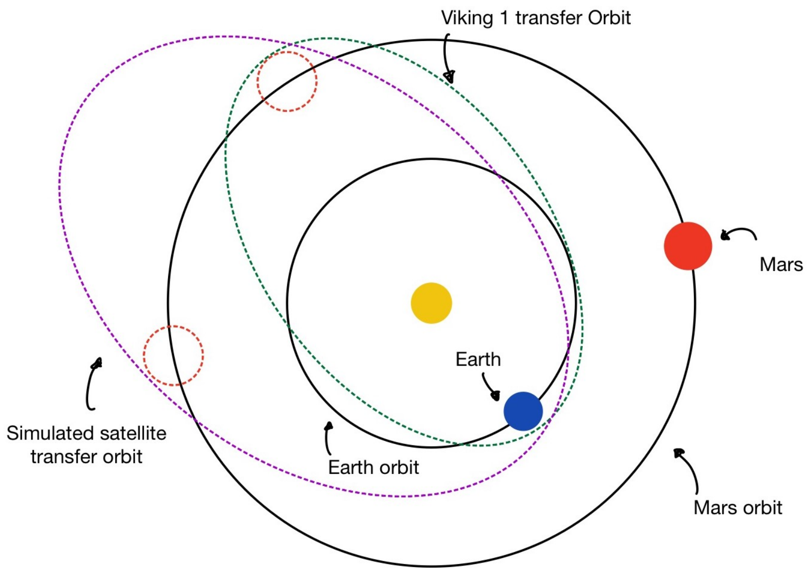


Figure 7: Diagram showing transfer orbits of simulated satellite vs Viking I

To improve the result of this experiment and achieve a more optimised transfer orbit, it would be recommended to use the same approach orbit as Viking 1. And when increasing the initial velocity of the satellite, to do so such that the direction of the velocity vector points towards the prograde, as that is the optimum direction where the eccentricity of the orbit will increase by exploiting the Oberth effect [8].

While the simulation code works normally in most cases, as the bodies are modelled as point masses, there is a possibility that two bodies may fly close by each other in the animation, but in reality the two bodies would have collided with each other, resulting in all the subsequent data recorded as being unphysical. To improve the code, a method that detects when the radii of two bodies intersect would be recommended.

Another suggestion to improve the code would be to utilise a more accurate numerical integration method, such as the Runge-Kutta integration method While this would yield extremely accurate results for the simulation, it would also require exponentially more computational power.

5 Conclusion

A Python script was used to simulate the orbital motion of the inner Solar system. Using this simulation and its calculations, experiments could be conducted. The Beeman algorithm was proven to be symplectic yielding a Total energy of the orbital system value of $-6.1402577(9) \cdot 10^{33}$ J. The initial velocity required to create a transfer orbit to allow a satellite to intercept Mars form Earth was determined to be 32804.17 ms^{-1}.

References

[1] D. Beeman, "Some multistep methods for use in molecular dynamics calculations", Journal of computational physics **20**, 130–139 (1976).

[2] D. Williams, *Planetary fact sheet - metric* (https://nssdc.gsfc.nasa.gov/planetary/factsheet/ , Accessed on 31st March 2021, 2021).

[3] G. A. Soffen and C. W. Snyder, "The first viking mission to mars", Science **193**, 759–766 (1976).

[4] NASA, *Let's go to mars! calculating launch windows* (https://www.jpl.nasa.gov/edu/teach/activity/lets-go-to-mars-calculating-launch-windows/ , Accessed on 31st March 2021, 2021).

[5] T. Watkins, *The potential, kinetic and total energies of the planets* (http://www.applet-magic.com/planetenerg Accessed on 31st March 2021).

[6] A. K. Mazur, "Common molecular dynamics algorithms revisited: accuracy and optimal time steps of störmer–leapfrog integrators", Journal of Computational Physics **136**, 354–365 (1997).

[7] Nasa, *Viking* (http://www.applet-magic.com/planetenergy.htm, Accessed on 31st March 2021, 1975).

[8] R. Adams and G. Richardson, "Using the two-burn escape maneuver for fast transfers in the solar system and beyond", in 46th aiaa/asme/sae/asee joint propulsion conference & exhibit (2010), p. 6595.