Computation and Simulation of Hohmann Transfer using Python

Anirudh Malik (2020PSC1013)

Manish Sharma (2020PSC1015)

Suhas Adiga (2020PSC1105)

S.G.T.B. Khalsa College, University of Delhi, Delhi-110007, India.

November 13, 2021

Project Report Submitted to

Dr. Mamta and Dr. H. C. Ramo

as part of internal assessment for the course

?32223902 - Computational Physics Skills**?**

Abstract

In 1925, Walter Hohmann discovered an orbit to do low cost interplanetary missions which is now called Hohmann Tranfer Orbit. In this, we make the spacecraft revolve around the Sun in an elliptical orbit which intersects with the origin and the destination planets' orbits. Then we make sure that the spacecraft is launched at a time when the angle between the origin and the destination planet is such that when the spacecraft is at its apogee it encounters the destination planet. In this paper we present the result and analysis of this method to reach various planets in our solar system with least energy given from our end. We have made use of Python and solved the coupled differential equations using Leapfrog Integration and plotted the resultant Hohmann Tranfer Orbit using Matplotlib.

Contents

1 Introduction

Almost everything around us that travel at speed less than the **'c'** (c=3 X 10^8 m/s) and the particles that are of macroscopic scale follows laws of mechanics. The thought of mechanics almost dates to the period of Greeks.They were the first one's to discover that Earth is round, and estimated the radius of the Earth accurately. They also attempted to measure Earth-Sun and Earth-Moon distances. Eudoxus and Aristotle who belonged to Plato's academy was worried about the movement of stars and planets also gave a perfect theory about Epicycles[1] and retrograde motion[2] of the planets. They are also believed to have proposed the Geocentric model of planetary system then. Nearly 1400 years later, it was Nicholas Copernicus,a Polish Astronomer who disproved Ptolemy's Geocentric model as it couldn't explain retrograde motion of planets accurately. He gave a reasonably fair explanation to retrograde motion by assuming that Sun is at the centre of Planetary system. Hence giving a foundation to Heliocentric ideology. Later Tyco Brahe did his observation from an observatory he had set up in an island in Copenhagen,Denmark from 1576-1597, his data set was almost accurate upto 2-3 minutes of arc but he couldn't analyse the data set as it was huge.Followed by his death, it was his student Kepler who observed the data carefully and gave '*Kepler's Laws of Planetary Motions*'[**1**], which we will be mostly using it in many of our equations to calculate the time taken by our spacecraft to reach destination planet from origin planet.Later using this ideologies we all know the contribution Newton and Galileo have had for this field. Though later Einstein disproved mechanics through his '*Theory of Relativity*', but till today mechanics can still explain lot of Planetary motions. The field of mechanics which deals with planetary sciences and orbits is called '**Orbital Mechanics'**.

Ever since our childhood, after reading about Kalpana Chawla's tragic death, or Cassini-Huygen's first soft landing on Saturn's moon Titan in 2005, Messenger's first orbit around Mercury in 2011, Mangalyaan's successfull insertion in Mars's orbit in 2014 or the recent Parker probe mission to Sun, it has ever wondered on how their trajectories to reach their target works. Hence we took it as a challenge to work on trajectory to reach Mars from Earth, but the enthusiasm within us after working on the code helped us to embed the code work for all interplanetary transfer.

In this project we have solved the problem of **'Interplanetary transfer of a Satellite'**.The interplanetary transfer can be mainly done by two types of transfer :

- Bi-elliptic Transfer
- Hohmann Transfer

Bi-elliptic transfer is a three impulse transfer, that is spacecraft is fired at three places in order to

[1]Epicyclic motion refers to a circle moving on another circle

[2]Retrograde motion refers to movement where planet goes backward then forward and again backward.

be placed in a required orbit involving two elliptical orbit to reach the destination planet.While Hohmann transfer is a two impulse transfer, where the spacecraft is fired at two different place in order to reach the destination planet. This type of transfer only has one elliptical orbit, which reaches the destination planet at the apogee.In both type of transfer the energy and angular momentum is conserved, which mainly helps in solving equations easily.The main advantage of Hohmann transfer is, energy is conserved here and this can be verified as the spacecraft would be reaching at apogee which is the shortest distance i.e along major axis of Hohmann Transfer orbit. One can argue that we can just reach destination planet along a straight line joining the two, but for such type of transfer the spacecraft needs to be fired throughout its motion hence making it not economical, but here in this type of transfer its fired for a small duration of time and then the gravitational force on satellite due to sun helps it to reach it's destination hence consuming the least energy. Hohmann Transfer is named after German Engineer,Walter Hohmann who published a paper on this type of transfer in his 1925 book '*Die Erreichbarkeit der Himmelskörper (The Attainability of Celestial Bodies)*'.[**2**]

Fig 1.1: Walter Hohmann [**3**]

Important Data				
Planet	Orbital Radius (in AU)	Time Period (in Earth Years)	Eccentricity (e)	Mass(in $10^{-6} X M_{Sun}$)
Mercury	0.387	0.240	0.205	0.166014
Venus	0.722	0.616	0.007	2.08106272
Earth	1.000	1.000	0.017	3.003486962
Mars	1.520	1.880	0.094	0.3232371722
Jupiter	5.200	11.860	0.049	954.7919
Saturn	9.580	29.440	0.057	285.885670
Uranus	19.200	83.960	0.046	43.66244
Neptune	30.100	164.770	0.011	51.51384
Pluto	39.500	247.780	0.250	0.0163

Table 1.1: This table has all the data used in the code to compute Δv_1 and Δv_2 [4]

In the upcoming section we will be explaining about the theoretical approach we took to solve problem including the steps we took to make the equations dimensionless.In Methodology section, we have explained on the computational methods we used to solve this problem followed by the analysis of set of results we have produced followed by summary.Interested ones can go through the appendix where the derivation of formulas is there and even the python code is added.

2 Theory

- **Problem Statement:**
  Compute the validity of Hohmann Transfer orbit from a planet 'A' (Origin Planet) to planet 'B' (Destination Planet) and simulate the transfer orbit.Calculate the time required to reach the destination planet.

2.1 Formulas With Dimension

Let $\mathbf{V}_{Perigee}$ represent the velocity of spacecraft at perigee (or) velocity of spacecraft when it's given the first impulse near the Origin Planet.
Let $\mathbf{V}_{Apogee}$ represent the velocity of spacecraft at Apogee (or) velocity of spacecraft just before the second impulse is given near the Destination Planet.
Let $\mathbf{V}_{Origin}$ represent the orbital velocity of Origin planet around Sun and $\mathbf{V}_{Destination}$ represent the orbital velocity of Destination planet around Sun.
Let $\mathbf{M}_{Sun}$ represent the mass of sun and $\mathbf{M}_{Sun}$= **1.98 X 10^{30} Kg** and $\mathbf{M}_{Spacecraft}$ be mass of the spacecraft.
Let $\mathbf{R}_{Origin}$ be the radial distance of origin planet from sun (i.e distance between origin planet and sun),$\mathbf{R}_{Destination}$ be the radial distance of destination planet from sun (i.e distance between destination planet and sun) and **T** is the time taken by spacecraft from perigee to apogee.
Let $\Delta\mathbf{v}_1$ be the extra velocity (first impulse) given at perigee in order to reach the destination planet and $\Delta\mathbf{v}_2$ be the extra velocity (second impulse) given at apogee in order to stay in the orbit of the destination planet and **G** is Universal Gravitation Constant and **G=6.67 X $10^{-11} Nm^2kg^{-2}$.**

$$\mathbf{V_{Origin}} = \sqrt{\frac{\mathbf{GM_{Sun}}}{\mathbf{R_{Origin}}}} \tag{1}$$

$$\mathbf{V_{Destination}} = \sqrt{\frac{\mathbf{GM_{Sun}}}{\mathbf{R_{Destination}}}} \tag{2}$$

$$\mathbf{V_{Perigee}} = \sqrt{\frac{\mathbf{2GM_{Sun}R_{Destination}}}{\mathbf{R_{Origin}(R_{Origin}+R_{Destination})}}} \tag{3}$$

$$\mathbf{V_{Apogee}} = \sqrt{\frac{\mathbf{2GM_{Sun}R_{Origin}}}{\mathbf{R_{Destination}(R_{Origin}+R_{Destination})}}} \tag{4}$$

$$\mathbf{\Delta v_1} = \sqrt{\frac{\mathbf{GM_{Sun}}}{\mathbf{R_{Origin}}}}\left(\sqrt{\frac{\mathbf{2R_{Destination}}}{\mathbf{R_{Origin}+R_{Destination}}}} - \mathbf{1}\right) \tag{5}$$

$$\mathbf{\Delta v_2} = \sqrt{\frac{\mathbf{GM_{Sun}}}{\mathbf{R_{Destination}}}}(\mathbf{1} - \sqrt{\frac{\mathbf{2R_{Origin}}}{\mathbf{R_{Origin} + R_{Destination}}}}) \tag{6}$$

$$\mathbf{T} = (\frac{\mathbf{R_{Origin} + R_{Destination}}}{\mathbf{2}})^{\frac{3}{2}} \tag{7}$$

2.2 Formulas in Dimensionless form

To make the above given equations dimensionless we will fix **G=1** and M_{Sun}=**1** and this change will not change the result in plot. Usually on changing the value of constants, the time taken to reach destination planet from origin planet changes and similarly the distance of planets from the sun also needs to be scaled.But here in our code, it's programmed in such a way that the following changes need not be made.Here in the code we have used a formula to calculate the time required to reach the different set of planets and since the major axis of orbits of different planets is scaled, the transfer orbit's major axis is also scaled by same factor hence this also doesn't create much of a problem in our simulation.The modified equation which is dimensionless is as follows:

$$\mathbf{V'_{Origin}} = \sqrt{\frac{\mathbf{1}}{\mathbf{R_{Origin}}}} \tag{8}$$

$$\mathbf{V'_{Destination}} = \sqrt{\frac{\mathbf{1}}{\mathbf{R_{Destination}}}} \tag{9}$$

$$\mathbf{V'_{Perigee}} = \sqrt{\frac{\mathbf{2R_{Destination}}}{\mathbf{R_{Origin}(R_{Origin} + R_{Destination})}}} \tag{10}$$

$$\mathbf{V'_{Apogee}} = \sqrt{\frac{\mathbf{2R_{Origin}}}{\mathbf{R_{Destination}(R_{Origin} + R_{Destination})}}} \tag{11}$$

$$\mathbf{\Delta v'_1} = \sqrt{\frac{\mathbf{1}}{\mathbf{R_{Origin}}}}(\sqrt{\frac{\mathbf{2R_{Destination}}}{\mathbf{R_{Origin} + R_{Destination}}}} - \mathbf{1}) \tag{12}$$

$$\mathbf{\Delta v'_2} = \sqrt{\frac{\mathbf{1}}{\mathbf{R_{Destination}}}}(\mathbf{1} - \sqrt{\frac{\mathbf{2R_{Origin}}}{\mathbf{R_{Origin} + R_{Destination}}}}) \tag{13}$$

$$\mathbf{T} = (\frac{\mathbf{R_{Origin} + R_{Destination}}}{\mathbf{2}})^{\frac{3}{2}} \tag{14}$$

where $V'_{Perigee}$ represent the velocity of spacecraft at perigee (or) velocity of spacecraft when it's given the first impulse near the Origin Planet in **dimensionless form**.
where V'_{Apogee} represent the velocity of spacecraft at Apogee (or) velocity of spacecraft just

before the second impulse is given near the Destination Planet in **dimensionless form**. where V'_{Origin} represent the orbital velocity of Origin planet around Sun and $V'_{Destination}$ represent the orbital velocity of Destination planet around Sun in **dimensionless form**. where $\Delta\mathbf{v}'_1$ be the extra velocity (first impulse) given at perigee in order to reach the destination planet and $\Delta\mathbf{v}'_2$ be the extra velocity (second impulse) given at apogee in order to stay in the orbit of the destination planet in **dimensionless form**.

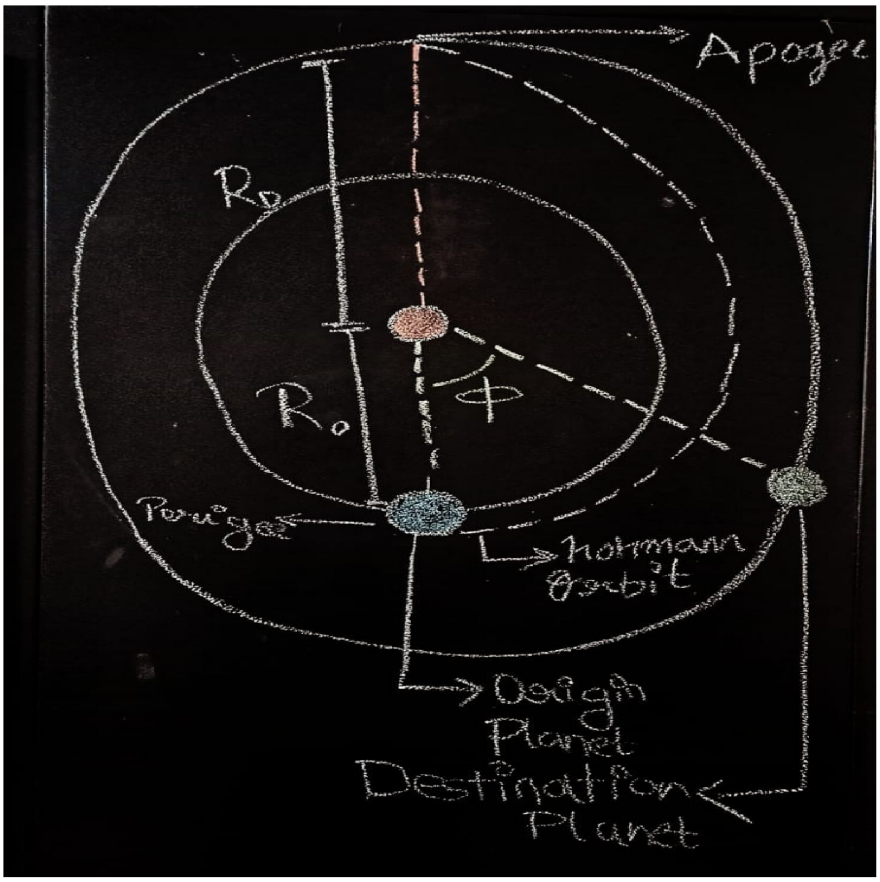


Fig 1.2: Diagrammatic representation of Hohmann Transfer [3]

In the aboce given diagram we can clearly see that the space craft is launched from origin planet (coloured blue) from perigee and the Destination planet (coloured green). The eccentric anomaly here is 'ϕ'. The height at which the satellite is launched can be neglected, as height, h<<<< R_O (where R_O is same as R_{Origin}). The spacecraft launched from perigee takes the elliptical path (dotted line) and reaches the destination planet at apogee.We an clearly see that the spacecraft has travelled 'π' radians and destination planet has travelled '$\pi - \phi$' radians when they meet at Apogee.The satellite is given Δv_1 at perigee in order to reach the destination planet at apogee,and again Δv_2 in order to match orbital velocity of destination planet and stay along with it throughout. But one also has to see

[3]This diagram was drawn by hand by one of our teammate

that, for this type of transfer to be successful there is a particular ϕ for which only it is successful transfer via '**Least Energy Method**'. And for any interplanetary transfer, both the planets should be in the same plane.For example, for a hohmann transfer from Earth to Mars, the window is open only for 26 months.We can also save some energy by using *Gravity Assists* i.e using of gravitational force of different planets and spacecraft at closest approach.

Assumptions:

1) All planets are in same plane.

2)Gravitation effect of Sun is only dominant in the entire solar system.

3)The positions of planets in the solar system is random and is fixed at their eccentric anomaly. So for example if we are doing a transfer from Earth to Mars, the position for these two planets is fixed while of the others is random and this can be seen when we do the execution many times.

3 Methodology

3.1 Use of Leapfrog Integration:

The numerical method we used is *Leapfrog Integration method*. Leapfrog method is generally useful in solving the equation of motion (i.e. the Newton' s second law) of a dynamical system in classical mechanics, which is of the form

$$\frac{d^2y}{dt^2} = a(y)$$

The above equation of motion can also be written in the form

$$a = \frac{dv}{dt}$$

and

$$v = \frac{dy}{dt}$$

Now our aim is to calculate position y(t) and velocity v(t) by solving two coupled first order ordinary differential equations. For this, first of all, we descretize the time domain

Fig 1.3: The figure shows the descretization of the time domain, starting with t_0 , into evenly spaced time points t_i where $t_i = t_0 + i\Delta t$ (i=1,2,3...n and size of each time step $\Delta t = t_1 - t_0 = t_2 - t_1$ and so on). The values of y and v at the first point t_0 is known to us by the initial conditions. Moreover, the values of y and v at time t_i (i^{th} time point)are denoted by y_i and v_i where $y_i = y(t_i) = y(t_0 + i\Delta t)$ and $v_i = v(t_i) = v(t_0 + i\Delta t)$. **[5]**

To achieve the second order accuracy, in the Leapfrog method, the position y is evaluated at the end-points of the time points (at t_0, t_1, t_2 ...) and velocity v is evaluated at the mid-points of the time points (at $t_{1/2}, t_{3/2}, t_{5/2}$ ...), i.e. y and v are staggered in such a way that they "leapfrog" over each other as shown in the figure.

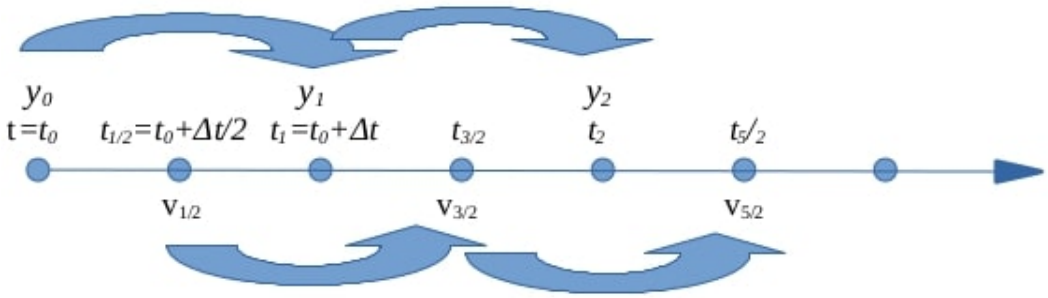


Fig 1.4: Diagrammatic Representation of Leapfrog Integration **[6]**

With this definition, the Leapfrog scheme can be obtained that advances y_i to y_{i+1} and $v_{i+1/2}$ to $v_{i+3/2}$:

$$y_{i+1} = y_i + v_{i+1/2}\Delta t$$

$$v_{i+1/2} = v_{i-1/2} + a_i\Delta t$$

The Leapfrog scheme is second order accurate. Hence, in terms of accuracy, the Leapfrog scheme is better than the Euler's scheme but inferior to the Runge-Kutta 4^{th} order scheme. Yet we chose Leapfrog for 2 major reasons. The first is the time-reversibility of the Leapfrog method i.e. we can calculate the solution in forward n time steps, and then reverse the direction of integration and can obtain the solution in backwards n time steps to arrive at the same starting time. The second strength is that the scheme conserves the energy of dynamical systems. Particularly, this strength becomes crucial when computing orbital dynamics. In comparison, many other integration schemes such as the Runge-Kutta 4^{th} order, do not conserve energy.

3.2 Implementation in the code

In the code, we start by assigning a random value of eccentric anomaly(the angle that a planet makes with respect to semi-major axis of the planet) to all the planets except the 2 planets included in the Hohmann Transfer since these planets need a specific eccentric anomaly for a successful transfer. Then we calculate and assign the specific values required to the planets included in the Hohmann Transfer. Using these values of eccentric anomaly we calculate the initial position and initial velocity of each planet. Once the initial conditions are calculated, we can use those in the *Leapfrog Integration equations* by dividing the timestep (Δt) by 2 and then calculating the position and velocity after $\Delta t/2$ time interval. We use the same method repeatedly after every $\Delta t/2$ time to get position and velocity at every half-step. Now, moving on to the trajectory of the spacecraft. At the start of the simulation, the initial eccentric anomaly and initial position of the spacecraft is same as the origin planet. Then we calculate the value of initial velocity of the spacecraft according to the derived equations for a successful transfer. Once we have given the initial position and velocity we can use the same method as we used for the planets, using *Leapfrog Integration*, we can calculate the position and velocity of the spacecraft at every instance. This will result in an elliptical orbit around the Sun which intersects the orbits of the origin and the destination planets. Once we have reached the desired destination planet, we calculate and implement another velocity impulse calculated using the formulas derived above to align the spacecraft's orbit with the destination planet's orbit. After this impulse, the spacecraft stays with the planet and hence completing a successful **Hohmann Tranfer**.

3.3 Matplotlib

After calculating all the necessary values we move on to a graphical representation of the data with *matplotlib* package in python. We plot all the planets with accurate distances and accurate elliptical orbits. Then we plot the values of position calculated for each planet and the spacecraft at every instant. We provide two options in our program. First one being a static plot showing the entire Hohmann Transfer of the spacecraft as an image/graph. The second one being an animation made using the *FuncAnimation* function in *matplotlib* in which positions calculated at each timestep are shown as time passes.

4 Analysis of Numerical Results

4.1 Hohmann Transfer from Earth to Mars

(Using the formula from **equation (7)**), it takes spacecraft 0.71 Earth years to reach Mars from Earth but we have shown how will the path be if it travels for 2.6 earth years, so as to clarify everything to the reader.)

1. **Simulation 1**

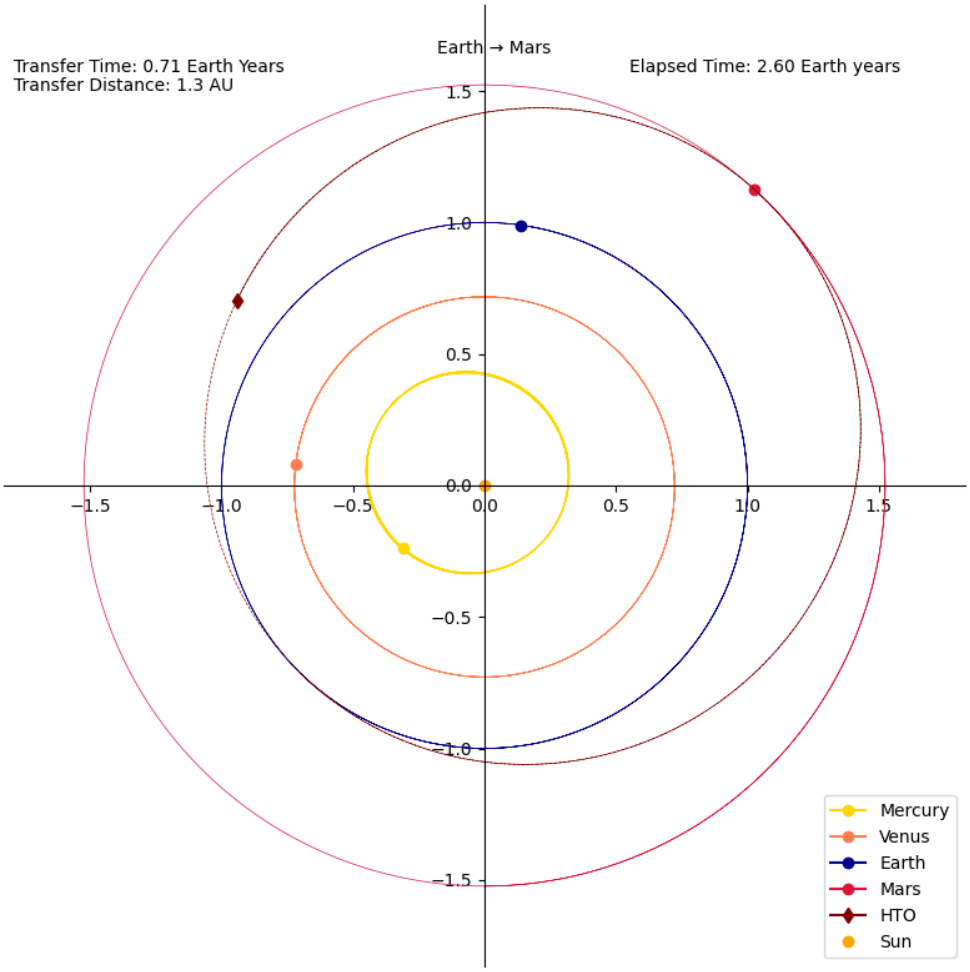


Fig 1.6: Hohmann Transfer from Earth to Mars without second impulse at apogee

Link to animation: Click here

In this type of transfer, the spacecraft left from Earth to Mars meets at apogee, but since second impulse isn't given the satellite returns back to Earth in the described path as shown. The spacecraft here takes 0.71 years to reach apogee and then again 0.71 years to come back to Earth. So now around 1.42 years have elapsed and in remaining 0.58 years

the spacecraft is almost about to reach the apogee point, but one can see that Mars is lagging now, since the second time when spacecraft is launched from Earth the eccentric anomaly isn't the same that is required for successful transit. In spite that we have checked for 2.6 earth years, so that the reader can clearly see what happens when mars reaches the original position where it first started.

2. **Simulation 2**

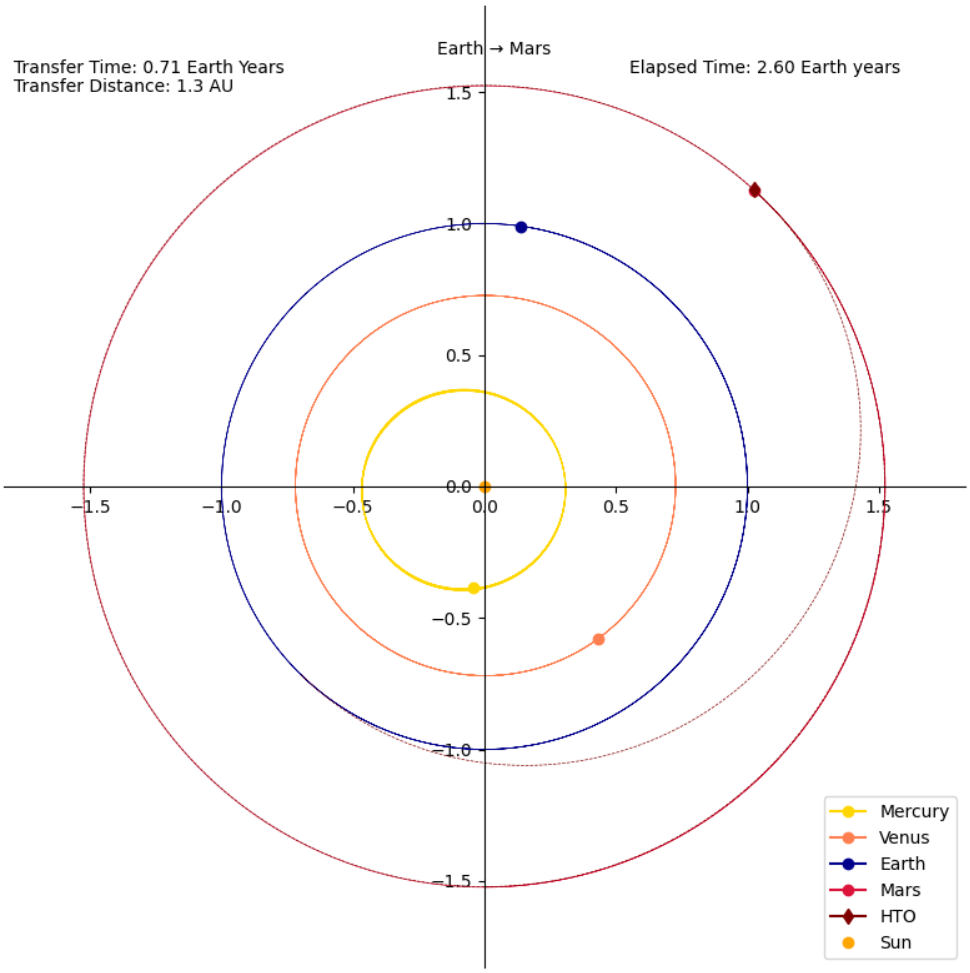


Fig1.8 Hohmann Transfer from Earth to Mars with impulse at apogee.

Link to animation: Click here

In this type of transfer the spacecraft left from Earth to Mars continues to stay in the orbit of Mars, as second impulse given at apogee helps to match velocity of spacecraft to that of Mars orbital velocity around Sun. So one can clearly see that the graph plotted is for 2.60 Earth years and spacecraft takes around 0.71 years to reach Mars from Earth and then stays in along Mars. So to come back to same point Mars takes around 1.89 Earth years hence for a person on Earth he can see back the satellite at same point after 2.60 years.

4.2 Hohmann Transfer from Venus to Mars

1. **Simulation 1**

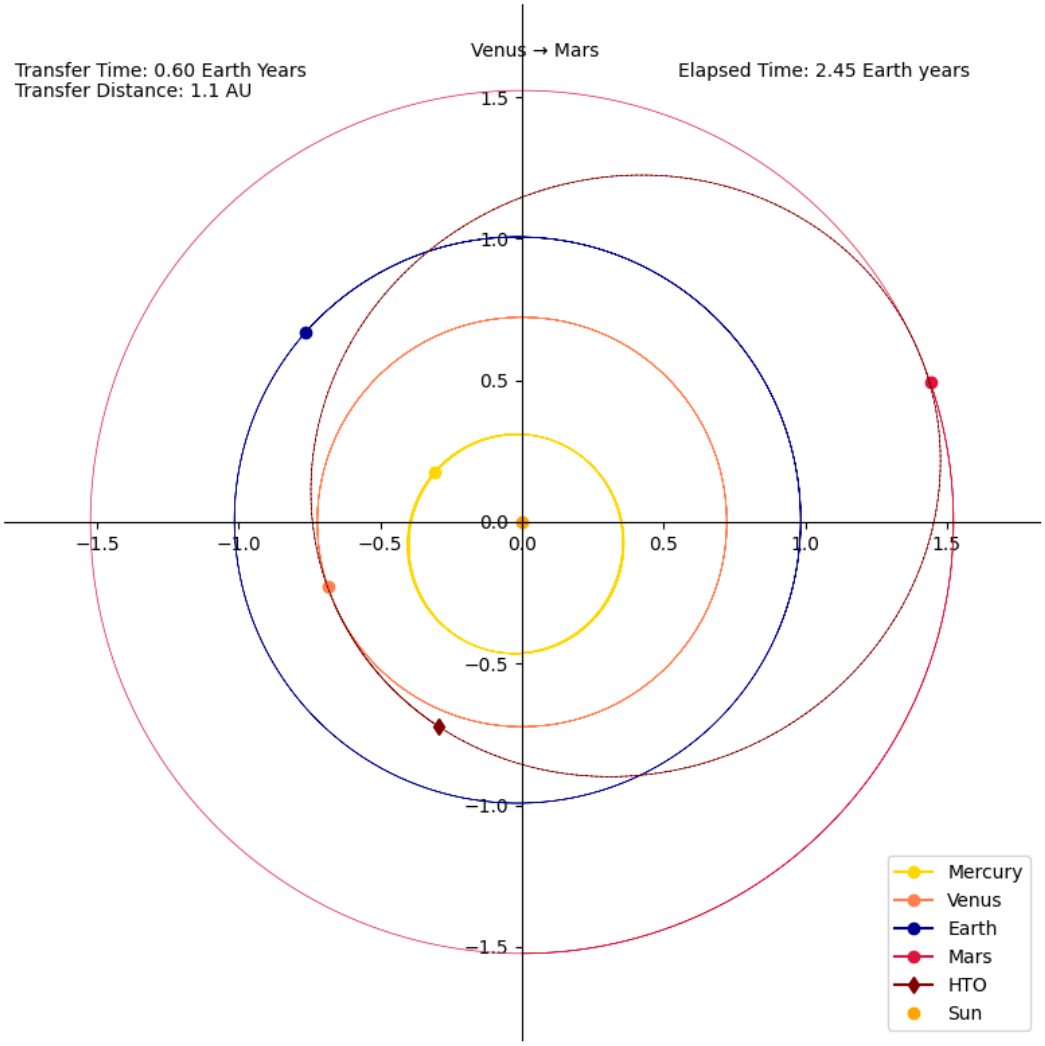


Fig 1.10: Hohmann Transfer from Venus to Mars without second impulse at apogee

Link to animation: Click here

In this type of transfer, the spacecraft left from Venus to Mars meets at apogee, but since second impulse isn't given the satellite returns back to Venus in the described path as shown.The spacecraft here takes 0.65 years to reach apogee and then again 0.65 years to come back to Venus. So now around 1.30 years have elapsed and in remaining 1.15 years the spacecraft is almost about to reach the apogee point, but one can see that Mars is lagging now, since the second time when spacecraft is launched from Venus the eccentric anomaly isn't the same that is required for successful transit.

2. **Simulation 2**

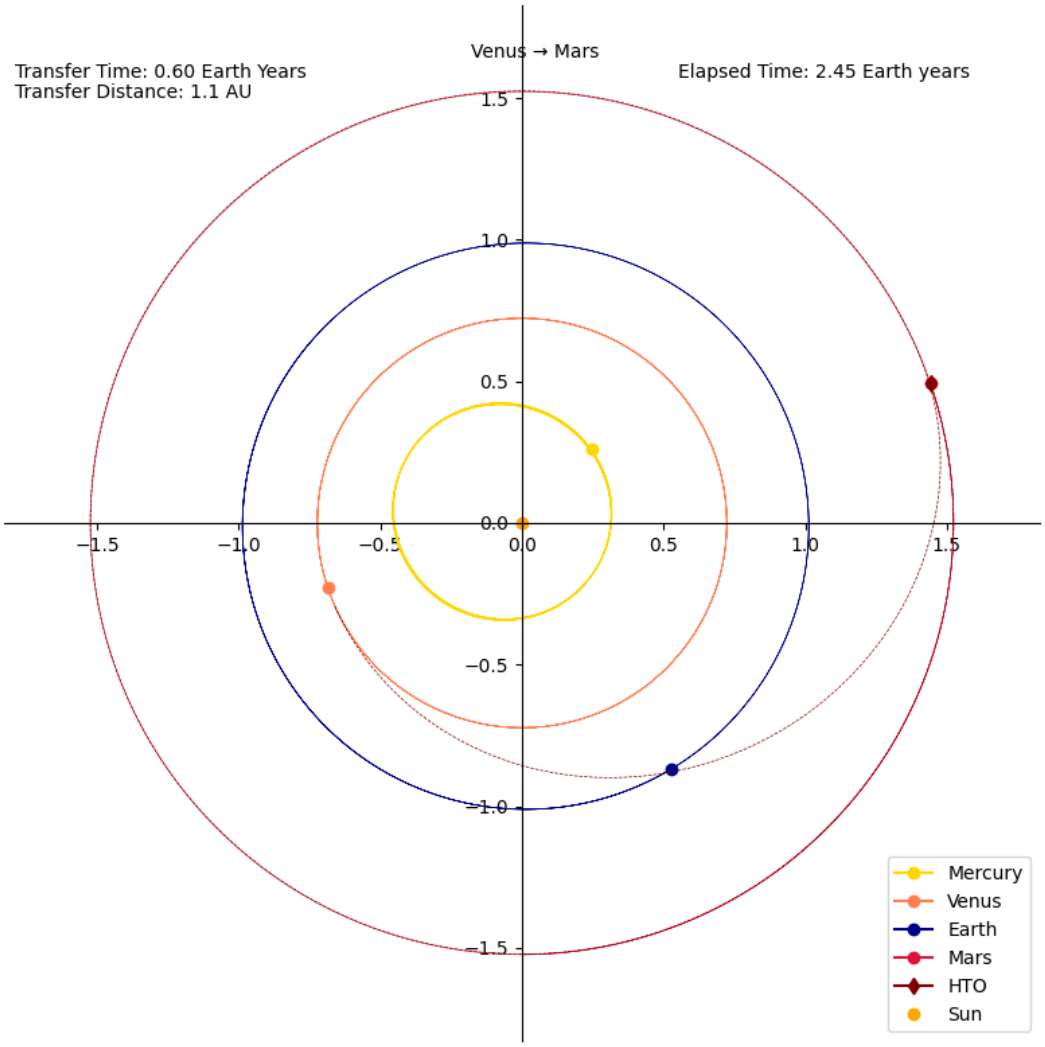


Fig1.12 Hohmann Transfer from Venus to Mars with impulse at apogee

Link to animation: Click here

In this type of transfer the spacecraft left from Venus to Mars continues to stay in the orbit of Mars, as second impulse given at apogee helps to match velocity of spacecraft to that of Mars orbital velocity around Sun.So one can clearly see that the graph plotted is for 2.45 Earth years and spacecraft takes around 0.65 years to reach Mars from Venus and then stays in along Mars. So to come back to same point Mars takes around 1.81 Earth years hence we can see back the satellite at same point after 2.45 Earth Years.

5 Summary

We can conclude this project report by saying that we were successful in plotting an accurate *Hohmann Transfer Orbit* by using the derived formulas for 2 sets of planets. As we observed that it is a fuel efficient method and velocity was added at only two points during the transfer. This efficiency makes it one of the best orbital transfer methods.

The simulation was also successful in calculating the correct values and explaining the concept well using visual graphs and animations. Usage of the *Leapfrog Integration method* made the process of calculating values of position and velocity at every point much simpler, faster and efficient. The implementation of GUI for the user interface made the overall program very user friendly and easy to use. All in all, the entire coding part was a huge success as we were able to achieve all the milestones we sought to achieve.

Our experience throughout this project was very good. We got to learn a lot of new mathematical as well as computational concepts. For example, *Leapfrog Integration method*, we first got to know about this concept only due to this project and now after understanding it, it seems like such a simple concept with such a huge area for implementation. This project also sparked our interest in orbital mechanics.

The coding part was the one where all of us learnt the most. We got to learn so much about how to make graphs in *matplotlib* and customize them according to our requirements and make them look appealing to the user. The GUI was an entirely new concept for us. For this project, we had to learn the *tkinter* package of python from scratch. Overall this project was a huge learning opportunity for us and we believe we made the best use of this opportunity.

At last, we would like to thank our teachers for all the support, guidance and motivation throughout this project. The teachers helped us in understanding the mathematical concepts, simplifying the problem, improving our code and finally making this project report. Without the support of the teachers, all this would not have been possible for us.

References

[1] Verma, M K, 2008, Introduction to Mechanics, Pg No: 1-10

[2] Hohmann,Walter,1930,National Air and Space Museum Archives, Smithsonian Institution, https://airandspace.si.edu/multimedia-gallery/6493hjpg

[3] Jimodell,A Thought Experiment: Walter Hohmann,2010.

[4] Planetary Fact Sheet (Metric),NASA, https://nssdc.gsfc.nasa.gov/planetary/factsheet/

[5] Dr.Sanjay Kumar (2017), Leap Frog Method and its Application, Pg No :2 .

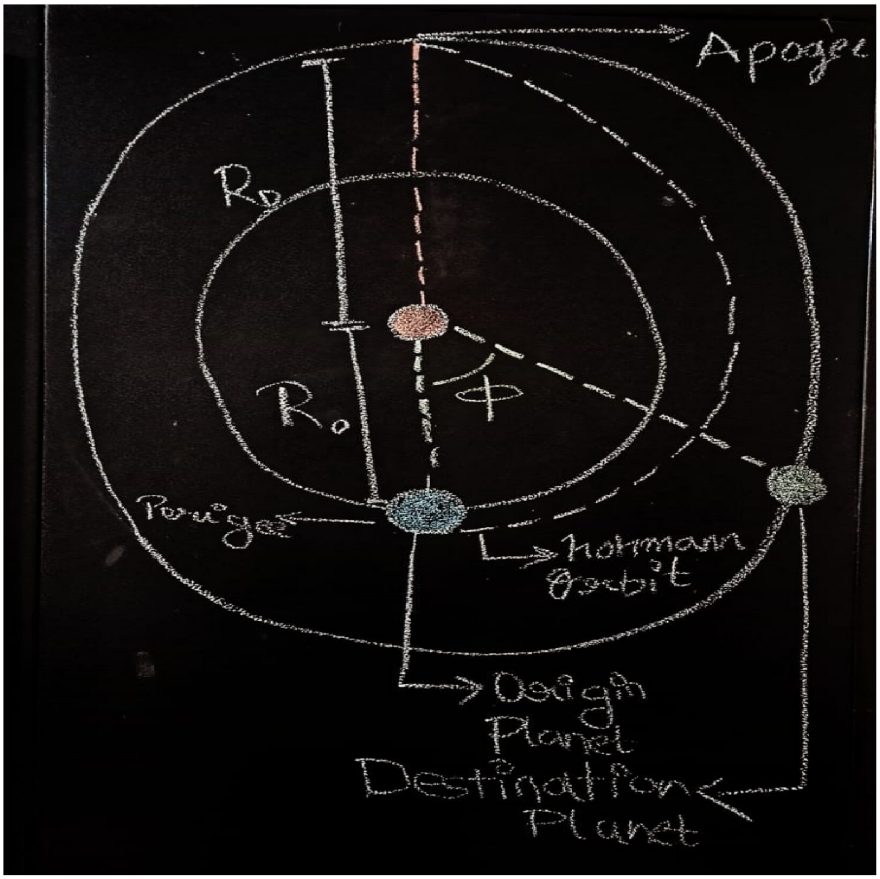
[6] Dr.Sanjay Kumar (2017), Leap Frog Method and its Application, Pg No :3 .

[7] Roy,A E (1977),Astrodynamics (for teachers),Physics Education,Volume 12, Number 7, doi:10.1088/0031-9120/12/7/022

[8] Widnall,S, 16.07 Dynamics (Fall 2008), MIT OCW.

[9] Orbital Mechanics, Gonzalez,Alfonso.

A Derivations



Appendix Fig-I: **Graphical Representation of Hohmann Transfer**

When the spacecraft is launched from Earth,it is also orbiting around sun so it's having a velocity, V_{Origin} and additional impulse given is Δv_1, so we can say this:

$$V_{Orbital} + \Delta v_1 = V_{Perigee} \tag{15}$$

Similarly when the spacecraft reaches the apogee point, it's having a velocity V_{Apogee} and a additional impulse of Δv_2 is given to match the orbital velocity of destination planet around sun i.e $V_{Destination}$.The equation can be given as:

$$V_{Apogee} + \Delta v_2 = V_{Destination} \tag{16}$$

In this system, angular momentum is conserved:

$$L_{initial} = L_{final}$$

$$\mathbf{m_{Spacecraft} V_{Perigee} R_{Origin}} = \mathbf{m_{Spacecraft} V_{Apogee} R_{Destination}} \tag{17}$$

$$V_{Perigee} = V_{Apogee}(\frac{R_{Destination}}{R_{Origin}}) \tag{18}$$

In this system,Energy is also conserved :

$$E_{initial} = E_{Final}$$

$$\mathbf{\frac{1}{2} m_{Spacecraft} v^2_{Perigee}} - \mathbf{\frac{GM_{Sun} m_{Spacecraft}}{R_{Origin}}} = \mathbf{\frac{1}{2} m_{Spacecraft} v^2_{Apogee}} - \mathbf{\frac{GM_{Sun} m_{Spacecraft}}{R_{Destination}}} \tag{19}$$

Cancel $m_{Spacecraft}$ on both sides,

$$\frac{1}{2} v^2_{Perigee} - \frac{GM_{Sun}}{R_{Origin}} = \frac{1}{2} v^2_{Apogee} - \frac{GM_{Sun}}{R_{Destination}} \tag{20}$$

Re-arranging terms,

$$\frac{1}{2}(v^2_{Perigee} - v^2_{Apogee}) = (GM_{Sun})(\frac{1}{R_{Origin}} - \frac{1}{R_{Destination}}) \tag{21}$$

Substitute equation (18) in equation (21),

$$\frac{v^2_{Perigee}}{2}(\frac{R^2_{Destination} - R^2_{Origin}}{R^2_{Destination}}) = GM_{Sun}\frac{(R_{Destination} - R_{Origin})}{R_{Origin} R_{Destination}}) \tag{22}$$

On using $a^2 - b^2 = (a - b)(a + b)$ in equation (22)

$$\mathbf{v^2_{Perigee}}(\frac{\mathbf{R_{Destination} + R_{Origin}}}{\mathbf{R_{Destination}}}) = \mathbf{2}(\frac{\mathbf{GM_{Sun}}}{\mathbf{R_{Origin}}}) \tag{23}$$

Hence,

$$\mathbf{v_{Perigee}} = \sqrt{\frac{\mathbf{2GM_{Sun} R_{Destination}}}{\mathbf{R_{Origin}(R_{Destination} + R_{Origin})}}} \tag{24}$$

Similarly,

$$\mathbf{v_{Apogee}} = \sqrt{\frac{\mathbf{2GM_{Sun} R_{Origin}}}{\mathbf{R_{Destination}(R_{Destination} + R_{Origin})}}} \tag{25}$$

Using equation (24) & (25) in equation (15) (16) we get,

$$\mathbf{\Delta v_1} = \sqrt{\frac{\mathbf{GM_{Sun}}}{\mathbf{R_{Origin}}}}(\sqrt{\frac{\mathbf{2R_{Destination}}}{\mathbf{R_{Origin} + R_{Destination}}}} - 1) \tag{26}$$

$$\mathbf{\Delta v_2} = \sqrt{\frac{\mathbf{GM_{Sun}}}{\mathbf{R_{Destination}}}}(\mathbf{1} - \sqrt{\frac{\mathbf{2R_{Origin}}}{\mathbf{R_{Origin} + R_{Destination}}}}) \tag{27}$$

From Kepler's III Law, $T^2 = R^3$, when R is in AU.
The time period of Satellite is '2T' when it returns back to earth but now T is required as we are worried only about reaching Apogee.
Now we know that semi major axis of Hohmann Orbit is equal to $R_{Origin} + R_{Destination}$
therefore time taken by spacecraft to reach apogee is :

$$\mathbf{T} = (\frac{\mathbf{R_{Origin} + R_{Destination}}}{\mathbf{2}})^{\frac{3}{2}} \tag{28}$$

Let's now calculate the eccentric anomaly (ϕ),
Let $T_{Destination}$ be time period of Destination planet and then following relation can be obtained by considering angular displacement :

$$\frac{T}{T_{Destination}} = \frac{180° - \phi}{360°} \tag{29}$$

Note: Here all terms used carry the same meaning as defined before in **Theory**

B Programs

Listing 1: Hohmann Transfer-Team25

```
import matplotlib
matplotlib.use('TkAgg')      =
import matplotlib.animation as animation
from matplotlib.backends.backend_tkagg
import FigureCanvasTkAgg, NavigationToolbar2Tk
import matplotlib.pyplot as plt
import numpy as np
import tkinter as tk
from tkinter import ttk
from _tkinter import TclError

class Planet():
'''
The planet class contains the physics, the orbital motion of the planet around the
central body
'''


def __init__(self, a, T, e, m, sim, c):

# Physical properties
self.semiMajor = a
self.orbitalPeriod = T
self.eccentricity = e
self.mass = m * 1e-6     # m is mass in solar mass units

# Calculate radius of region where planet's gravitational  influence on spacecraft
# dominates all other influences
self.sphereOfInfluence = self.semiMajor * np.power(self.mass, 2/5)

# Included in simulation
self.simulate = sim
self.colour = c
self.linestyle = '-'
self.marker = 'o'

# Simulation quantities
self.eccentricAnomaly = None
# later angle in radians
```

```
self.position = None
# later given as (x, y) coordinate
self.velocity = None
#later given as (vx, vy) components


def distanceToSun(self):
x, y = self.position
return np.sqrt(x**2 + y**2)

def getVelocity(self):
x, y = self.velocity
return np.sqrt(x**2 + y**2)

def calculateInitialPosition(self):
# Use distance and anomaly to calculate initial positions x0, y0
distance = self.semiMajor * (1. - self.eccentricity)
return (distance*np.sin(self.eccentricAnomaly),
-distance*np.cos(self.eccentricAnomaly))

def calculateInitialVelocity(self, delta):
# Calculating the x- and y-components requires the specific timestep of the
simulation
# Taylor approximation of velocities at step n=1/2
velocity = np.sqrt(2./self.distanceToSun() - 1./self.semiMajor)
return (velocity * np.cos(self.eccentricAnomaly)
-self.position[0]/self.distanceToSun()**3 * delta/2,
velocity * np.sin(self.eccentricAnomaly)
- self.position[1]/self.distanceToSun()**3 * delta/2)

def updatePosition(self, delta):
# Return tuple with updated position
return tuple(i + delta * vi for i, vi in zip(self.position, self.velocity))
# for i in x, y

def updateVelocity(self, delta):
# Return tuple with updated velocity, using leap-frog integration scheme
return tuple(i - (delta * ri) / self.distanceToSun()**3 for i,
ri in zip(self.velocity, self.position))
```

```
class SolarSystemSimulation():
'''
This class handles the simulation parameters
'''

Planets = {}

def __init__(self):

self.delta = gui.stepsize.get() * 2*np.pi / 365.256   # integration timestep in Earth
days
self.steps = round(365.256 * gui.duration.get() / gui.stepsize.get())  # integration
years

self.originPlanet = ''  # HTO Planets
self.destinationPlanet = ''



# ------
# Initialize Planets
# Units of AU and Year
Mercury= Planet(a=0.387, e=0.205, T=0.241, m=0.166014,      sim=gui.mercury.get(),
c ='Gold')
Venus= Planet(a=0.723, e=0.007, T=0.615, m=2.08106272,    sim=gui.venus.get(),
c ='Coral')
Earth= Planet(a=1.,     e=0.017, T=1.,     m=3.003486962,  sim=gui.earth.get(),
c ='DarkBlue')
Mars= Planet(a=1.524, e=0.094, T=1.88,  m=0.3232371722, sim=gui.mars.get(),
c ='Crimson')
Jupiter= Planet(a=5.200, e=0.049, T=11.860,m=954.7919, sim=gui.jupiter.get(),
c ='orange')
Saturn= Planet(a=9.58,  e=0.057, T=29.5,  m=285.885670,    sim=gui.saturn.get(),
c ='Khaki')
Uranus= Planet(a=19.20, e=0.046, T=84,     m=43.66244,      sim=gui.uranus.get(),
c ='Turquoise')
Neptune= Planet(a=30.06, e=0.011, T=164.8, m=51.51384,      sim=gui.neptune.get(),
c ='RoyalBlue')
Pluto= Planet  (a=39.50, e=0.250, T=247.78,m=0.0163,        sim=gui.pluto.get(),
c ='Yellow')
```

```
# For easier interaction between tkinter and class code, place the planet instances
in a dict
SolarSystemSimulation.Planets = {
'Mercury':   Mercury,
'Venus':     Venus,
'Earth':     Earth,
'Mars':      Mars,
'Jupiter':   Jupiter,
'Saturn':    Saturn,
'Uranus':    Uranus,
'Neptune':   Neptune,
'Pluto':     Pluto
           }

# ------
# Hohmann Transfer Orbit Parameters
if gui.plotHohmann.get():
# User has to have selected origin and destination of Hohmann Transfer. If not,
# this function just returns a message and does nothing
try:
self.originPlanet, self.destinationPlanet =
gui.origin.get(gui.origin.curselection()),
gui.destination.get(gui.destination.curselection())
except TclError:
print('\n ! You have to select origin and destination planet of
the Hohmann Transfer Orbit ! \n')
return None

# ------
# If planet is in simulation, determine its initial position and velocity
for key, planet in SolarSystemSimulation.Planets.copy().items():
if planet.simulate or key in [self.originPlanet, self.destinationPlanet]:
# Randomly place the planets on their respective orbits
planet.eccentricAnomaly = np.random.uniform(0, 2*np.pi)

planet.position = planet.calculateInitialPosition()
planet.velocity = planet.calculateInitialVelocity(self.delta)

else: # if planets are not part of the simulation, remove them
SolarSystemSimulation.Planets.pop(key)
```

```
def timeStep(self):
# Update planet position and velocity after one timestep
for planet in SolarSystemSimulation.Planets.values():
planet.position = planet.updatePosition(self.delta)
planet.velocity = planet.updateVelocity(self.delta)


class Spacecraft(Planet):

def __init__(self, origin, destination, delta):

self.origin = origin              # Instance of Planet Class
self.destination = destination    # Instance of Planet Class

self.position = origin.position
self.eccentricAnomaly = origin.eccentricAnomaly
self.velocity = self.calculateInitialVelocity(delta)

self.colour = 'Maroon'
self.linestyle = '—'
self.marker = 'd'
self.simulate = False             # This is to exclude HTO from plot limits check

self.orbitInserted = False
def calculateInitialVelocity(self, delta):
# Initial velocity is defined by the distance the spacecraft has to travel
dO = self.origin.semiMajor * (1 - self.origin.eccentricity)  # Leave at perihelion
dD = self.destination.semiMajor # Arrive at aphelion
# The velocity of the spacecraft is defined by the HTO parameters
velocity = np.sqrt(1/dO) + np.sqrt(1/dO) * (np.sqrt(2*dD / (dD + dO)) - 1)
return (velocity*np.cos(self.eccentricAnomaly)
- self.origin.position[0]/self.origin.distanceToSun()**3 * delta/2,
velocity*np.sin(self.eccentricAnomaly)
- self.origin.position[1]/self.origin.distanceToSun()**3 * delta/2)

def performOrbitInsertion(self, Sim):
if not Sim.Planets['HTO'].orbitInserted:
# Adjusts the spacecraft's velocity if it has not been done yet
semiMajorOrigin = Sim.Planets[Sim.originPlanet].semiMajor
semiMajorDestination = Sim.Planets[Sim.destinationPlanet].semiMajor

requiredAcceleration = np.sqrt(1/semiMajorDestination) * (1 -
```

```
np.sqrt(2 * semiMajorOrigin / (semiMajorOrigin + semiMajorDestination)))
# x-component
# Acceleration is split up into x- and y-component
Sim.Planets['HTO'].velocity = (Sim.Planets['HTO'].velocity[0]
+ requiredAcceleration*(Sim.Planets['HTO'].velocity[0]/Sim.Planets['HTO']
.getVelocity()),
Sim.Planets['HTO'].velocity[1] + requiredAcceleration*(Sim.Planets['HTO']
.velocity[1]/Sim.Planets['HTO'].
getVelocity()))
Sim.Planets['HTO'].orbitInserted = True

class App:
# ------
# Set up control window
def __init__(self, master):
self.master = master

# ------
# Upper Panel: Select Planets to Plot
self.planet_frame = ttk.LabelFrame(master, text='Choose Planets to Plot')
self.planet_frame.grid(row=1, columnspan=5, sticky='EW')

# Planets
self.mercury = tk.BooleanVar()
self.venus = tk.BooleanVar()
self.earth = tk.BooleanVar()
self.mars = tk.BooleanVar()
self.jupiter = tk.BooleanVar()
self.saturn = tk.BooleanVar()
self.uranus = tk.BooleanVar()
self.neptune = tk.BooleanVar()
self.pluto = tk.BooleanVar()

self.plot_mercury = ttk.Checkbutton(self.planet_frame, text='Mercury',
variable=self.mercury)
self.plot_mercury.grid(row=1, column=0, sticky='EW', padx=5, pady=5)
self.plot_venus = ttk.Checkbutton(self.planet_frame, text='Venus',
variable=self.venus)
self.plot_venus.grid(row=2, column=0, sticky='EW', padx=5, pady=5)
self.plot_earth = ttk.Checkbutton(self.planet_frame, text='Earth',
variable=self.earth)
self.plot_earth.grid(row=3, column=0, sticky='EW', padx=5, pady=5)
```

```
self.plot_mars = ttk.Checkbutton(self.planet_frame, text='Mars', variable=self.mars)
self.plot_mars.grid(row=4, column=0, sticky='EW', padx=5, pady=5)
self.plot_jupiter = ttk.Checkbutton(self.planet_frame, text='Jupiter',
variable=self.jupiter)
self.plot_jupiter.grid(row=1, column=1, sticky='EW', padx=5, pady=5)
self.plot_saturn = ttk.Checkbutton(self.planet_frame, text='Saturn',
variable=self.saturn)
self.plot_saturn.grid(row=2, column=1, sticky='EW', padx=5, pady=5)
self.plot_uranus = ttk.Checkbutton(self.planet_frame, text='Uranus',
variable=self.uranus)
self.plot_uranus.grid(row=3, column=1, sticky='EW', padx=5, pady=5)
self.plot_neptune = ttk.Checkbutton(self.planet_frame, text='Neptune',
variable=self.neptune)
self.plot_neptune.grid(row=4, column=1, sticky='EW', padx=5, pady=5)
self.plot_pluto = ttk.Checkbutton(self.planet_frame, text='Pluto',
variable=self.pluto)
self.plot_pluto.grid(row=1, column=2, sticky='EW', padx=5, pady=5)

# ------
# Lower Panel: Hohmann Transfer Orbit selection
self.transfer_frame = ttk.LabelFrame(master, text='Choose Origin and Destination')
self.transfer_frame.grid(row=5, columnspan=5, sticky='EW')

self.origin = tk.Listbox(self.transfer_frame, exportselection=0)
self.origin.grid(row=6, column=0, sticky='EW')
for planet in planets:
self.origin.insert(0, planet)

self.destination = tk.Listbox(self.transfer_frame, exportselection=0)
self.destination.grid(row=6, column=2, sticky='EW')
for planet in planets:
self.destination.insert(0, planet)
self.plotHohmann = tk.BooleanVar()
self.orbitInsertion = tk.BooleanVar()
self.plotHohmann_check = ttk.Checkbutton(self.transfer_frame,
text='Plot Hohmann Transfer Orbit', variable=self.plotHohmann)
self.plotHohmann_check.grid(row=7, column=0, sticky='EW', padx=5, pady=5)

self.orbitInsertion_check = ttk.Checkbutton(self.transfer_frame,
text='Orbit Insertion', variable=self.orbitInsertion)
self.orbitInsertion_check.grid(row=7, column=2, sticky='EW', padx=5, pady=5)
```

```
# ------
# Other settings

self.stepsize = tk.DoubleVar()        # Integration timestep in Earth days
tk.Entry(self.transfer_frame,
textvariable=self.stepsize).grid(row=10, column=0, sticky='EW')
tk.Label(self.transfer_frame, text='Timestep in Earth
days').grid(row=10, column=2, sticky='E')

self.duration = tk.DoubleVar()
tk.Entry(self.transfer_frame,
textvariable=self.duration).grid(row=11, column=0, sticky='EW')
tk.Label(self.transfer_frame, text='Integration Time in Earth
Years').grid(row=11, column=2, sticky='E')

self.plot_button = ttk.Button(master, text='Suggest Simulation
Parameters', command=self.suggestSimParameters)
self.plot_button.grid(column=1, row=12, sticky='EW')

self.plot_button = ttk.Button(master, text='Plot',
command=self.plot)self.plot_button.grid(column=1, row=13,
sticky='EW')

self.animate_button = ttk.Button(master, text='Animate',
command=self.simulation_animation)
self.animate_button.grid(column=1, row=14, sticky='EW')

self.duration.set(0.72)
self.stepsize.set(1.0)

def plot(self):
# ------
# Start simulation
Sim = SolarSystemSimulation()

# ------
# Create new window to plot the simulation
sim_window = tk.Toplevel()
sim_window.title('Hohmann Transfer Orbit')
sim_window.focus_force()     # Auto-focus on window

fig, ax, elapsedTime = self.setupPlot(Sim)
```

```
# Check if HTO is plotted
if gui.plotHohmann.get():
Sim, travelTime, transferDistanceAU = self.hohmannTransfer(Sim)

# ------
# Add Information to Plot
info = self.addInformation(ax)
info['elapsedTime'].set_text('Elapsed Time: {:.2f} Earth
years'.format(gui.duration.get()))

if gui.plotHohmann.get():
info['originDestination'].set_text('%s \u2192 %s' %
(Sim.originPlanet, Sim.destinationPlanet))
info['transferTime'].set_text('Transfer Time: %.2f Earth Years'
%travelTime)
info['transferDistance'].set_text('Transfer Distance: %.1f AU' %
transferDistanceAU)

# ------
# Create plotting canvas
canvas = FigureCanvasTkAgg(fig, sim_window)
canvas.draw()
canvas.get_tk_widget().pack(side=tk.BOTTOM, fill=tk.BOTH,
expand=True)

# Add matplotlib toolbar to the plot
toolbar = NavigationToolbar2Tk(canvas, sim_window)
toolbar.update()
canvas._tkcanvas.pack(side=tk.TOP, fill=tk.BOTH, expand=True)

# ------
# Simulate and plot

# For each planet in simulation, calculate trajectories
planetTrajectories = {}
for key, planet in Sim.Planets.items():
planetTrajectories[key] = [planet.position]

sphereOfInfluence = Sim.Planets[Sim.destinationPlanet].sphereOfInfluence  # in AU

for i in range(Sim.steps):
```

```
Sim.timeStep()
for key, planet in Sim.Planets.items():
planetTrajectories[key].append(planet.position)

# If orbit insertion is performed, check if spacecraft has arrived at planet
if gui.plotHohmann.get():
if gui.orbitInsertion.get():
distanceToDestination =
np.sqrt((Sim.Planets['HTO'].position[0]-Sim.Planets[Sim.destinati
onPlanet].position[0])**2 +
(Sim.Planets['HTO'].position[1]-Sim.Planets[Sim.destinationPlanet
].position[1])**2)

if distanceToDestination < sphereOfInfluence:
Sim.Planets['HTO'].performOrbitInsertion(Sim)

# Plot final trajectories
for key, planet in Sim.Planets.items():
ax.plot(*zip(*planetTrajectories[key]), ls=planet.linestyle,
c=planet.colour, lw=0.5)
ax.plot(*planetTrajectories[key][-1], marker=planet.marker,
c=planet.colour)

#Add Sun
sun, = ax.plot(0, 0, c='orange', marker='o', ls='')

canvas.draw()

def setupPlot(self, Sim):
# ------
# Plot set-up
fig = plt.figure(figsize=(8, 8))
# Determine plot limits by finding planet with largest semi-major
axis included in simulation
lim = 0.4
for planet in Sim.Planets.values():
if planet.simulate:
if planet.semiMajor > lim:
lim = planet.semiMajor
lim *= 1.2
ax = fig.add_subplot(111, xlim=(-lim, lim), ylim=(-lim, lim),
aspect='equal', autoscale_on=False)
```

```
#ax.set(xlabel='x / AU', ylabel='y / AU')
elapsedTime = ax.text(0.65, 0.95, '', transform=ax.transAxes)

# Move axis to centre
ax.spines['left'].set_position('center')
ax.spines['bottom'].set_position('center')

# Eliminate upper and right axes
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')

# Show ticks in the left and lower axes only
ax.xaxis.set_ticks_position('bottom')
ax.yaxis.set_ticks_position('left')

plt.tight_layout()
return fig, ax, elapsedTime

def addInformation(self, ax):
info = {}
# Upper part shows basic information
info['originDestination'] = ax.text(0.45, 0.95, '',
transform=ax.transAxes)
info['transferTime'] = ax.text(0.01, 0.93, '
transform=ax.transAxes)
info['transferDistance'] = ax.text(0.01, 0.91, '',
transform=ax.transAxes)
info['elapsedTime'] = ax.text(0.65, 0.93, '',
transform=ax.transAxes)
info['destinationDistance'] = ax.text(0.65, 0.91, '',
transform=ax.transAxes)
info['transferStatus'] = ax.text(0.65, 0.89, '',
transform=ax.transAxes)
return info

def hohmannTransfer(self, Sim):
# The initial position and velocities of the origin and
destination planet have to be adjusted
# and their eccentricities set to zero
for i, planet in enumerate([Sim.Planets[Sim.originPlanet],
Sim.Planets[Sim.destinationPlanet]]):
planet.eccentricity = 0
```

```
# Planets need angular offset in order for the spacecraft to
arrive at the correct time
if i == 0:
# The destination planet has to have an angular offset with
respect to the origin planet
# in order for the spacecraft to arrive at the destination planet
at the correct time
# The offset depends on the travel time. Using Kepler's third law
in units of 1 AU and 1 Earth year: T^2/a^3=const
dO = planet.semiMajor * (1 - planet.eccentricity)  # Leave at
perihelion
dD = Sim.Planets[Sim.destinationPlanet].semiMajor                #
Arrive at aphelion

semiMajorHTO = (dO + dD) / 2
travelTime = np.sqrt(semiMajorHTO**3) / 2     # Only half-way
required
# Calculate required angular offset
offset = travelTime/Sim.Planets[Sim.destinationPlanet].orbitalPer
iod * 360.
planet.eccentricAnomaly = np.radians(offset + 180)

elif i ==1:
planet.eccentricAnomaly = 0
planet.position = planet.calculateInitialPosition()
planet.velocity = planet.calculateInitialVelocity(Sim.delta)

# ------
# The HTO is simulated using the spacecraft class.
# Initial velocity and position determine the trajectory,
therefore
# we can use the same Sim.timeStep function to calculate the
orbit
HTO = Spacecraft(origin=Sim.Planets[Sim.originPlanet],
destination=Sim.Planets[Sim.destinationPlanet], delta=Sim.delta)
# Now all left to do is to regard spacecraft as fake planet
# and append to list of simulated planets
Sim.Planets['HTO'] = HTO
return Sim, travelTime, semiMajorHTO

def simulation_animation(self):
# ------
```

```
# Start simulation
Sim = SolarSystemSimulation()

# ------
# Create new window to plot the simulation
sim_window = tk.Toplevel()
sim_window.title('Hohmann Transfer Team_25')
sim_window.focus_force()     # Auto-focus on window

# Check if HTO is plotted
if gui.plotHohmann.get():
Sim, travelTime, transferDistanceAU = self.hohmannTransfer(Sim)

fig, ax, elapsedTime = self.setupPlot(Sim)
# ------
# Create plotting canvas
canvas = FigureCanvasTkAgg(fig, sim_window)
canvas.draw()
canvas.get_tk_widget().pack(side=tk.BOTTOM, fill=tk.BOTH,
expand=True)

info = self.addInformation(ax)

# For each planet in simulation, add an empty line and store the instance in dict
planetTrajectories = {}
for key, planet in Sim.Planets.items():
line, = ax.plot([], [], ls=planet.linestyle, c=planet.colour,
lw=0.5)
sphere, = ax.plot([], [], marker=planet.marker, c=planet.colour)
planetTrajectories[key] = (line, sphere)

# Add Sun
sun, = ax.plot([], [], c='orange', marker='o', ls='')

def init():
# Initialize the animation
elapsedTime.set_text('')

# If HTO is plotted, include information
if gui.plotHohmann.get():
info['originDestination'].set_text('%s \u2192 %s' %
(Sim.originPlanet, Sim.destinationPlanet))
```

```
info['transferTime'].set_text('Transfer Time: %.2f Earth Years'
%travelTime)
info['transferDistance'].set_text('Transfer Distance: %.1f AU'
%transferDistanceAU)
info['destinationDistance'].set_text('Distance to Destination:')
info['transferStatus'].set_text('Transfer Status: in progress..')

trajectories = []
for line, sphere in planetTrajectories.values():
line.set_data([], [])
trajectories.append(line)
sphere.set_data([], [])
trajectories.append(sphere)

sun.set_data([], [])
trajectories.append(sun)

return trajectories

def animate(i):
# Animate the simulation

# ------
# Update the planet positions
Sim.timeStep()

# Update the planet trajectories
trajectories = []  # lines in plot
for key, planet in Sim.Planets.items():
# Append new position to line of trajectory
planetTrajectories[key][0].set_data(tuple(x + [xi] for x, xi in
zip(planetTrajectories[key][0].get_data(), planet.position)))
trajectories.append(planetTrajectories[key][0])

# Move planet sphere to current trajectory
planetTrajectories[key][1].set_data(planet.position)
trajectories.append(planetTrajectories[key][1])

sun.set_data([0], [0])
trajectories.append(sun)

elapsedTimeNumber = i * gui.stepsize.get() / 365.256
```

```
info['elapsedTime'].set_text('Elapsed Time: {:.2f} Earth
Years'.format(elapsedTimeNumber))
trajectories.append(info['elapsedTime'])

# HTO updates
if gui.plotHohmann.get():

xSpacecraft, ySpacecraft = Sim.Planets['HTO'].position

distancesToSpacecraft = {}
for key, planet in Sim.Planets.items():
if key != 'HTO':
distancesToSpacecraft[key] =
np.sqrt((xSpacecraft-planet.position[0])**2 +
(ySpacecraft-planet.position[1])**2)

# Give distance to target planet
info['destinationDistance'].set_text('Distance to Destination:
%.1fAU' % distancesToSpacecraft[Sim.destinationPlanet])
trajectories.append(info['destinationDistance'])

# See if spacecraft has arrived in sphere of influence of planet
sphereOfInfluence =
Sim.Planets[Sim.destinationPlanet].sphereOfInfluence  # in AU
if distancesToSpacecraft[Sim.destinationPlanet] <=
sphereOfInfluence:
# The HTO succeeded
info['transferStatus'].set_text('Transfer Status: \u2714
completed')
trajectories.append(info['transferStatus'])
# If orbit insertion is true and has not yet been performed,
perform second acceleration
if gui.orbitInsertion.get():
Sim.Planets['HTO'].performOrbitInsertion(Sim)

return trajectories

ani = animation.FuncAnimation(fig, animate, frames=Sim.steps,
interval=0, blit=True,
init_func=init, repeat=False)
canvas.draw()
def suggestSimParameters(self):
```

```
# ------
# Initialize simulation
Sim = SolarSystemSimulation()

# Suggest timestep based on planet with smallest orbit included
in simulation
smallestOrbit = min([planet.orbitalPeriod for planet in
Sim.Planets.values()])
gui.stepsize.set(round(smallestOrbit * 365.263 / 100, 2))

if gui.plotHohmann.get():
# Set duration of simulation to 110% of the HTO travelTime
dO = Sim.Planets[Sim.originPlanet].semiMajor * (1 -
Sim.Planets[Sim.originPlanet].eccentricity)  # Leave at
perihelion
dD = Sim.Planets[Sim.destinationPlanet].semiMajor # Arrive at
aphelion

semiMajorHTO = (dO + dD) / 2
travelTime = np.sqrt(semiMajorHTO**3) / 2     # Only half-way
required
gui.duration.set(round(1.1*travelTime , 2))

else:
# Set to one orbital period of planet with largest orbiatl period
in simulation
gui.duration.set(round(max([planet.orbitalPeriod for planet in
Sim.Planets.values()]), 2))
if __name__ == '__main__':
# Some definitions
planets = ['Pluto','Neptune', 'Uranus', 'Saturn', 'Jupiter',
'Mars', 'Earth', 'Venus', 'Mercury']

# ------
# Start GUI
root = tk.Tk()
root.wm_title('Hohmann Transfer Team 25')
gui = App(root)
root.mainloop()
```

C Contribution of team mates

Contribution of ¨*Anirudh Malik*¨

- In Formulation of the problem:
  Making the coupled differential equations dimensionless.
- In Programming:
  Formulation of functions that calculate additional information related to Hohmann Transfer to be shown in the graph.
- In Plotting Graphs:
  Adding additional calculated information in graph.
- In Report Writing:
  Result, Analysis and Summary.

Contribution of ¨*Manish Sharma*¨

- In Formulation of the problem:
  Introducing and implementing Leapfrog Integration method to solve the coupled differential equations.
- In Programming:
  Implementation of Leapfrog Integration to calculate and update velocity and position at every point and implementation of GUI.
- In Plotting Graphs:
  Animation in matplotlib.
- In Report Writing:
  Abstract, Methodology and Algorithm.

Contribution of ¨*Suhas Adiga*¨

- In Formulation of the problem:
  Theory and derivation of the Hohmann Transfer formulas.

- In Programming:

  Collecting data and information about the planets to plot accurate elliptical orbits.

- In Plotting Graphs:

  Plotting a static graph.

- In Report Writing:

  Introduction, Theory, Appendix and Program.