A dissertation report on

# “Interactive Learning Platform for Orbital Mechanics Using Python”

Submitted to



In partial fulfilment of the requirements for the award of degree

# Bachelor of Technology In

**Aerospace Engineering**

### Submitted by:

**Ramkiran L. Manjunath**

17030141AE007 17030141AE009

[lramkiranBTECH17@ced.alliance.edu.in](mailto:lramkiranBTECH17@ced.alliance.edu.in) [manjunathBTECH17@ced.alliance.edu.in](mailto:manjunathBTECH17@ced.alliance.edu.in)

### Monisha Patel A. Thoshitha R. Kumar

17030141AE013 17030141AE027

[pamonishaBTECH17@ced.alliance.edu.in](mailto:pamonishaBTECH17@ced.alliance.edu.in) [k](mailto:kuthoshithaBTECH17@ced.alliance.edu.in)[uthoshithaBTECH17@ced.alliance.edu.in](mailto:uthoshithaBTECH17@ced.alliance.edu.in)

### Under the guidance of

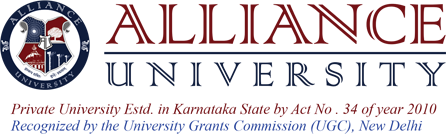
**Prof. Gisa G.S. Prof. Yadu Krishnan**

Assistant Professor Assistant Professor Department of Aerospace Engineering, Department of Aerospace Engineering,

Alliance College of Engineering and Design, Alliance College of Engineering and Design, Alliance University, Bengaluru. Alliance University, Bengaluru.

### Department of Aerospace Engineering Alliance College of Engineering and Design Alliance University, Bengaluru - 562106 Batch - 2017-21

**Year - 2021**



### CERTIFICATE

This is to certify that **Mr. Ramkiran L. (17030141AE007), Mr. Manjunath (17030141AE009), Ms. Monisha Patel A. (17030141AE012)** and **Ms. Thoshitha R. Kumar (17030141AE027)** students of **Aerospace Engineering, Bachelor of Technology 2017-21** batch at **Alliance College of Engineering and Design (ACED), Alliance University, Bengaluru** has completed the project report titled ***“Interactive Learning Platform for Orbital Mechanics Using Python”*** under our guidance in partial fulfillment for the award of Bachelor of Technology degree in Aerospace Engineering, Alliance University, Bangalore during the year 2020-2021.

### Prof. Gisa G.S Prof. Yadu Krishnan

Internal Guide Internal Guide

Department of Aerospace Engineering Department of Aerospace Engineering ACED, Alliance University ACED, Alliance University

Bengaluru Bengaluru

### Prof. Velmurugarajan K. Dr. Reeba Korah

Head of the Department Interim Dean

Department of Aerospace Engineering Department of Aerospace Engineering ACED, Alliance University ACED, Alliance University

Bengaluru Bengaluru

### External Viva

**1.**

### 2.

**Name of Examiners Signature with date**

# DECLARATION

We, Ramkiran L, Manjunath, Monisha Patel A, Thoshitha R. Kumar students of 8th Semester Bachelor of Technology in Aerospace Engineering, Alliance College of Engineering and Design (ACED), Alliance University, Bengaluru, hereby declare that the entire project work entitled “**Interactive Learning Platform for Orbital Mechanics Using Python**” is an authentic record of the work that has been carried out independently by us during final year of our B.Tech at ACED, under the esteemed guidance of **Prof. Gisa G.S** and **Prof. Yadu Krishnan S.**, Assistant Professors, Department of Aerospace Engineering, Alliance college of Engineering and Design, Alliance University.

This project report is submitted in partial fulfillment of requirements for the award of the degree of Bachelor of Technology in Aerospace Engineering. The results embodied in this dissertation are original and it has not been submitted in part or full for any degree in any University.

**Place:** Bengaluru

**Date:**17/062021

### Ramkiran.L Manjunath

**17030141AE007 17030141AE009**

**Monisha Patel A. Thoshitha R. Kumar 17030141AE012 17030141AE027**

# ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without the mention of the people, who are responsible for the completion of the project and who made it possible.

We take this opportunity to thank our beloved Interim Dean **Dr. Reeba Korah**, ACED, Alliance University, Bangalore for providing excellent academic environment in the college and her never–ending support to the B-Tech program.

We would like to convey our sincere gratitude to **Prof. K. Velmurugarajan**, Head of Department of Aerospace Engineering, ACED, Alliance University, Bangalore.

We would like to thank our internal guide **Prof. Gisa G.S** and **Prof. Yadu Krishnan S.**, Assistant Professors, Department of Aerospace Engineering, ACED, Alliance University, Bangalore for their support and encouragement given to carry out the project.

We would also like to thank our college staff members and well-wishers who directly or indirectly helped, motivated to complete this project successfully.

Lastly, we thank God almighty, our family, professors and friends for their constant encourage- ment without which this project would not have been possible.

# Contents

[List of Figures vii](#_TOC_250024)

[List of Tables 1](#_TOC_250023)

1. [Introduction 2](#_TOC_250022)
   1. [Features 2](#_TOC_250021)
   2. [Libraries Used 3](#_TOC_250020)
   3. [Market Research 3](#_TOC_250019)
   4. [Objective 5](#_TOC_250018)
   5. [Front End Development](#_bookmark11) 5
      1. [Introduction](#_bookmark12) 5
      2. [Home Pag](#_bookmark13)e 5
2. [Details of the Featur](#_bookmark15)es 6
   1. [Calculation of Orbital Elements](#_bookmark16) 6
   2. [2D and 3D Orbits](#_bookmark19) 9
   3. [Euler Angle 10](#_TOC_250017)
   4. [Sphere of Influence 11](#_TOC_250016)
   5. [Orbital Transfer 12](#_TOC_250015)
   6. [Calculation of Julian Day 13](#_TOC_250014)
   7. [Calculation of Parameters of the Orbit 14](#_TOC_250013)
   8. [Sensitivity Analysis 15](#_TOC_250012)
   9. [Position of One Spacecraft Relative To Another 16](#_TOC_250011)
   10. [Lagrangian Points 16](#_TOC_250010)
3. [Database Management System 17](#_TOC_250009)
   1. [Introduction 17](#_TOC_250008)

iv

1. [Conclusion 17](#_TOC_250007)
2. [Future Scope 18](#_TOC_250006)
3. [Code for the Features 19](#_TOC_250005)
   1. [Calculation Of Orbital Elements 19](#_TOC_250004)
   2. [Euler Angles 21](#_TOC_250003)
   3. [Sphere of Influence 22](#_TOC_250002)
   4. [Calculation of Julian Day 23](#_TOC_250001)
   5. [Calculation of Various Parameters of the Orbit 23](#_TOC_250000)
   6. [Sensitivity Analy](#_bookmark52)sis 26
   7. [Lagrangian P](#_bookmark53)oints 26
   8. [Calculation Of Relative Position of two Spacecrafts](#_bookmark54) 27
   9. [CRUD File Operations](#_bookmark55) 27

[A Dat](#_bookmark46)abase: 28

[Refer](#_bookmark57)ences 29

v

# List of Figures

1. MOPy 2
2. Global Market Trend according to Morgan Stanley 4
3. Indian Space Budget over the years. 4
4. Home Page of MOPy 6
5. Classical Orbital Elements 6
6. Alternate Orbital Elements 6
7. Euler Vector 10
8. Sphere of Influence of Earth, which is approx 140 times it’s radius 11
9. [Hohmann Transf](#_bookmark29)er 12
10. [Bi-Elliptical Hohmann Transfer](#_bookmark30) 13

vi

# List of Tables

1. List of Features present in MOPy 2
2. Inputs and outputs for Calculation of Orbital Elements 6
3. I/O for 2D and 3D orbits 9
4. I/O for conversion between Euler angle and DCM 10
5. I/O for Sphere of Influence 11
6. I/O for Orbital transfer 12
7. I/O for Julian-day 13
8. I/O for Various Parameters of the Orbit 14
9. [Formulae for Various parameters](#_bookmark35) 15
10. [I/O for Sensitivity Analy](#_bookmark37)sis 15
11. [I/O for Position of One Spacecraft Relative To Another](#_bookmark39) 16
12. [I/O for Lagrangian Points](#_bookmark41) 16
13. [Planetary Details](#_bookmark56) 28

vii

# Introduction

MOPy, a learning tool designed to learn and practice various orbital mechanics concepts. It is designed in a way such that it can be a user-friendly tool that can be operated with ease even by the user who has very limited knowledge about the concepts of orbital mechanics. It provides users to learn about a particular concept with a brief explanation so the user can gain the theoretical knowledge required through visualizations. The sophisticated 3D environment benefits the user to visualize the fundamental concepts easily. It assists the user to verify manually calculated data. It serves as advanced virtual calculator. MOPy is an open source software with GNU-GPL v3 license**lic** which anyone can use or work with. It runs on windows platforms at present.

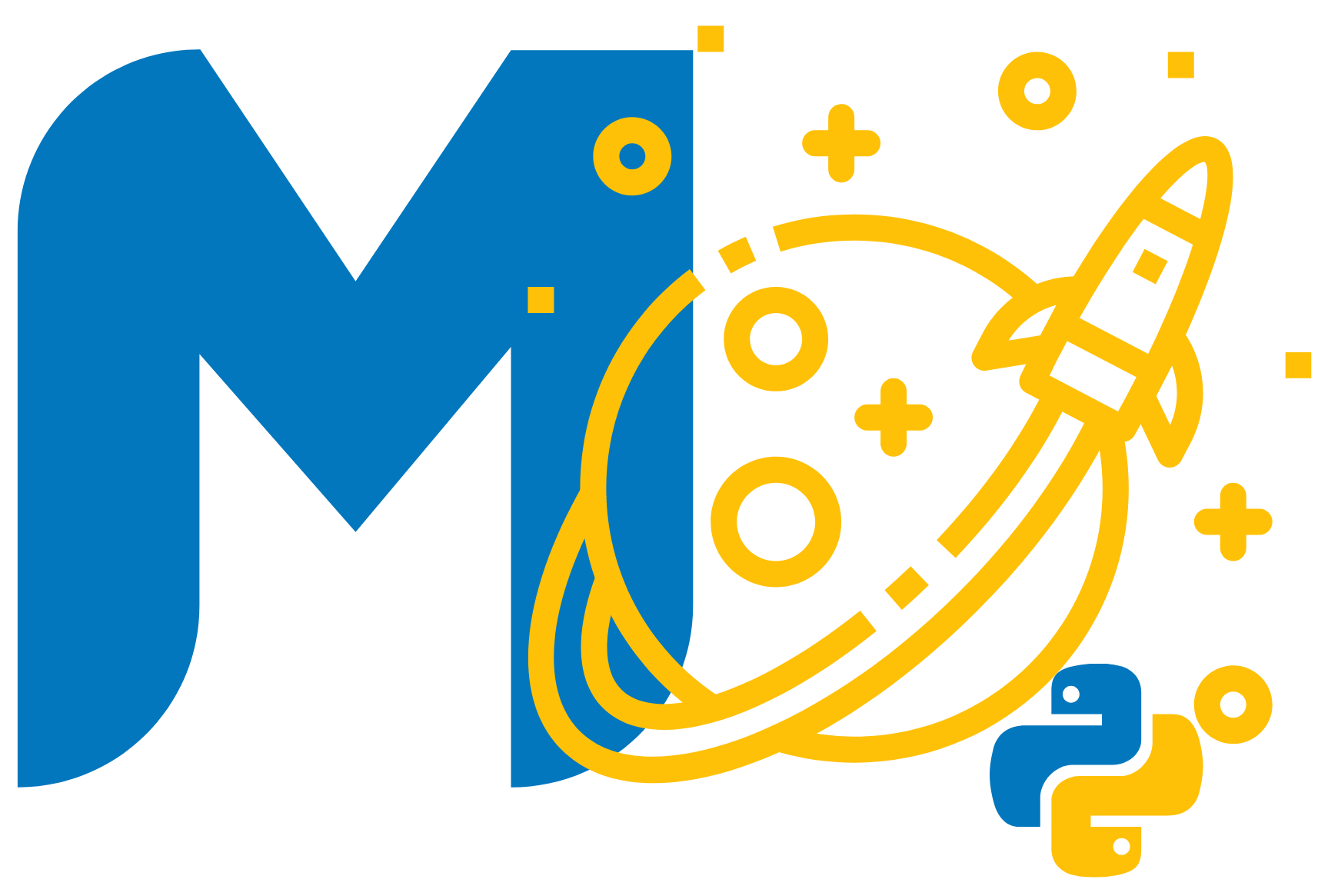


Figure 1: MOPy

## Features

|  |  |
| --- | --- |
| Sl. No | **Section** |
| 1 | Calculation of Orbital Elements |
| 2 | 2D and 3D orbit |
| 3 | Various Parameters at any given point |
| 4 | 2D and 3D Orbits |
| 5 | Calculation of Julian Day |
| 5 | Euler Angles |
| 6 | Sphere Of Influence |
| 7 | Sensitivity Analysis |
| 8 | Position of one Spacecraft w.r.t Another |
| 9 | Calculation of State and Velocity Vector |
| 10 | Orbital Transfer |

Table 1: List of Features present in MOPy

1

## Libraries Used

* + 1. **NumPy**: This brings MATLAB like functionality of using Matrices and their operations to python. This enables us to do a lot of stuff without much hassle.
    2. **SciPy** - This enables us to add many features involving more complex computing scenarios as it has features for scientific and technical computing. For example, it has different kinds of solvers for integration which we can use for solving acceleration vector equation to obtain the position vector for an orbit.
    3. **Matplotlib** - This is a plotting tool that is a extension on NumPy that gives the functionality of plotting many different kinds of graph. This library is somewhat similar to the plotting feature of MATLAB.
    4. **Panda3D** - This is a Game Engine based on C++ that takes in syntax from both C++ and Python. This provides real-time 3D visualizations and simulations based on the code.
    5. **SQLite3** - The entire details of the planetary bodies like the orbital elements, planetary ephemeris and others are stored in a local database. SQLite is used as it enables the offline functionality.
    6. **Qt Deisgner** - This enables MATLAB’s App Designer like feature of dragging and dropping the UI elements and creating the GUI. This is based on Qt, a cross platform GUI toolkit developed by the Qt Company
    7. **PyQt5** & **PySide2** - These both are the python binding libraries of Qt.
    8. **PyInstaller** - This library lets us convert our python code(.py) into executable file(.exe)

## Market Research

There are mainly two kinds of softwares.

* + 1. Simulation Based programs like STK, FreeFlyer etc.,
    2. Sandbox Based programs like Universe SandBox, Kerbal Space Program etc.,

The simulation based programs are mainly used to simulate missions and solve problems based on the instance. Both the applications given in the example are used in the industry for all kinds of missions, ranging from very small scale missions that are performed by the students to complicated missions that are performed by NASA and ISRO.

The Sandbox based programs are the stuff that are run by the physics engine that are baked into it. They use a 3D visualization toolkit or engine which lets the user easily interact with the UI, and change the parameters directly from the environment.

2

MOPy is a mix of both with enabling the user start from basic and learn the concepts with the help of the3D environment. The environment changes in realtime as the user changes the parameters thus helping the user understand the concepts eaisly and clearly.

In the analysis done by Morgan Stanley named “Investing in Space Exploration”, it is stated as - The revenue generated by the global space industry may increase to more than $1 trillion by 2040.

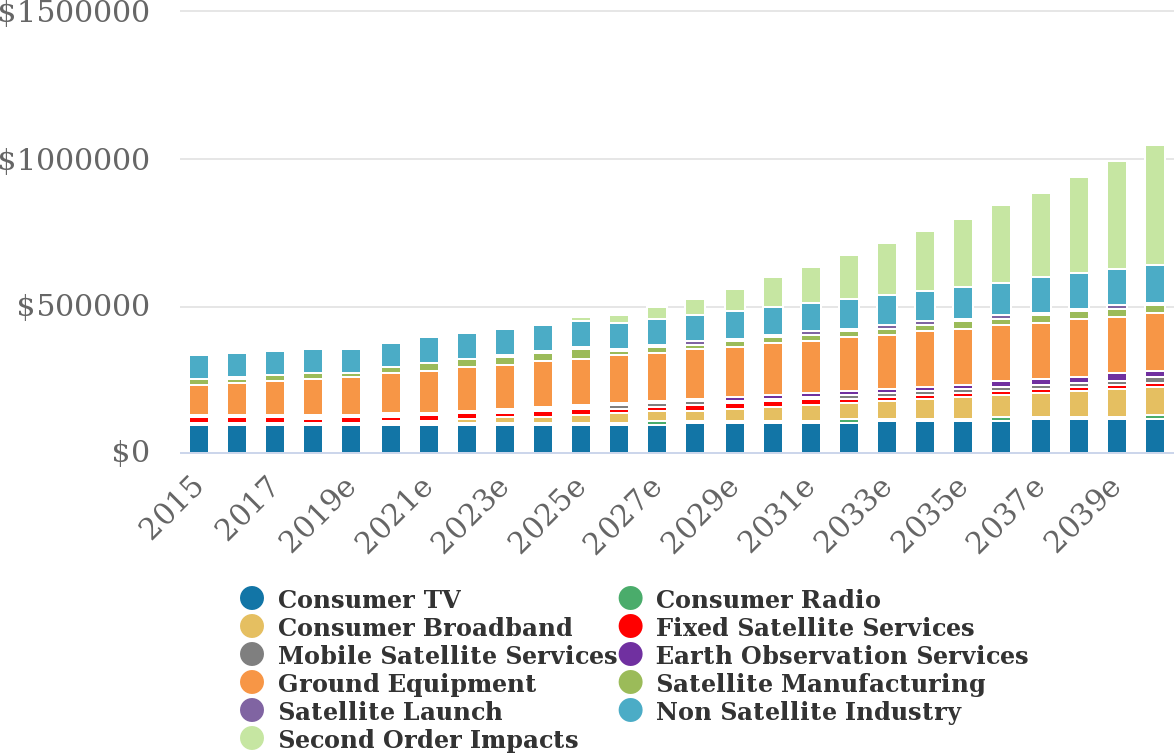


Figure 2: Global Market Trend according to Morgan Stanley.

A report by Antrix and PwC, it is stated that the indian space sector can become a $50 Billion industry, or about one per cent of India’s projected $5 Trillion economy, by 2024 from the current $7 billion, according to a report by the Antrix and PwC.

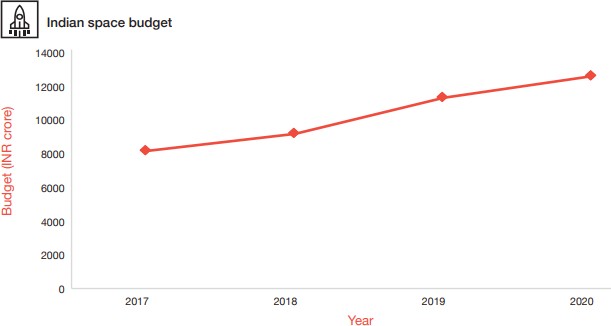


Figure 3: Indian Space Budget over the years.

3

## Objective

The development in the field of space technology is constantly increasing as shown in the aforementioned studies. Such being the case, many students are showing interest to learn more and more about space technology and its related concepts. Considering all such possibilities, we have come up with an idea to develop a learning tool beneficial to learn more about Orbital Mechanics The main objectives of this project are as follows:

* + 1. Design and develop a software to learn concepts and solve Problems related orbital mechanics to understand the basics.
    2. Provide a user friendly learning tool such that the user can operate even with the minimum knowledge about the concepts of orbital mechanics.
    3. Help user to visualize the virtual view of the space mission.
    4. Motivate the new comers to learn more abut the space studies.

## Front End Development

### Introduction

Front-end development deals with the Graphical User-Interface aspect of the software. It is the key developmental process that defines how the user experiences the features we have developed. The interface between the user and the back end code is GUI. The inputs from the user is taken from the GUI. So the design must be intuitive and clear. There are many libraries that can be used to develop a GUI like PyQt, Pyside, Kivy, Tkinter, etc. In our case, we have opted for PyQt5, Pyside2 and designed GUI in Qt-Designer. Then linked the Back-End scripts through the Integrated Development Environment(IDE) by Microsoft i.e, Visual Studio Code.

### Home Page

When the application opens the Home Page will load. In it, there is a Dropdown box containing all the features available, from which the user can choose which feature they want to use. When the user selects any of the features from the drop-down and clicks on the go button at the bottom, they are navigated to that screen where they can use the feature they selected. And then if they want to navigate back to the Home-Page they can click on the Home button provided at the top left corner of the screen. All the features are listed in the upper-mentioned table.

4



Figure 4: Home Page of MOPy

# Details of the Features

## Calculation of Orbital Elements

This can convert state velocity vector into orbital elements and vice versa. The inputs are as follows

Inputs State and Velocity Vectors Orbital Elements

Outputs Orbital Elements

State and Velocity Vectors

Table 2: Inputs and outputs for Calculation of Orbital Elements

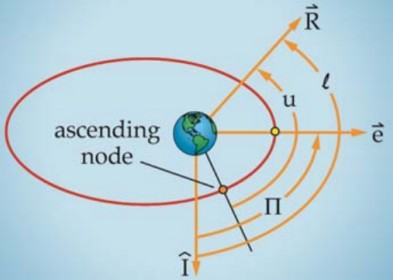
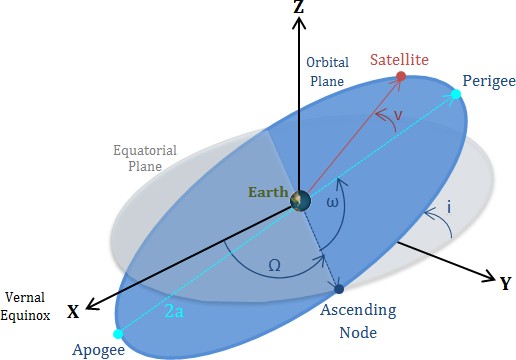


Figure 5: Classical Orbital Elements Figure 6: Alternate Orbital Elements

5

## Classical Orbital Elements

* + 1. **Semi-major Axis(***a***):** This is a constant that defines the size of the orbit. In Circular it is the radius of the circle. It is the longest diameter of an ellipse.

*a* = *ra* + *rp*

2

*h*2*/µ a* = 1 *− e*2

(1)

(2)

*where,*

*ra* = *Radius of Apogee*(*F*1*B*) *rp* = *Radius of Perigee*(*F*1*A*)

*µ* = *Standard Gravitational Parameter,*

* + 1. **Eccentricity(***e***):** This the parameter of the conic section which determines its shape. It is defined as the ratio of distance b/w two foci to the Major-axis.

*c*

*e* =

*a*

1 *v*2 *− µ*  *→−*

*→− →−*

*→−*

(3)

*e* =

*µ*

*where,*

*r × r −* ( *r . v* ) *× v*

(4)

*ra* & *rp* = *Radius of Apogee* & *Perigee respectively, µ* = *Standard Gravitational Parameter,*

*→−v* & *v* = *V elocity vector* & *Magnitude of V elocity V ector at r,*

*→−r* & *r* = *Radius V ector* & *Magnitude of Radius V ector.*

* + 1. **Inclination(***e***)**: It is the tilt of the orbit w.r.t the equatorial plane, measured at ascending node. It can also be defined as the angle from *K*ˆ unit vector to the specific angular momentum vector *→h*.

!

*i* = cos*−*1 *→−h .→−K* (5)

*|h|*

*where,*

*→−h* = *Specific Angular Momentum,*

*→−K* = *Unit V ector in Z direction,*

*|h|* = *Magnitude of Specific Angular Momentum.*

* + 1. **Right Ascension of Ascending Node(**Ω**):** RAAN horizontally orients the ascending node of the orbit w.r.t the Equatorial plane’s vernal equinox, measured in an equatorial plane This is not defined when the inclination is 0 or 180 degree. It lies between 0 to 360 degrees.

!

Ω = cos*−*1 *→−I .→−n* (6)

*|n|*

*where,*

*→−n* = *Nodal vector*

*Itjs the vector that joins the ascending node and the descending node.*

6

* + 1. **Argument of Perigee(***ω***):** Argument of Perigee defines the orientation of the ellipse in the orbital plane, as an angle measured from the ascending node to the periapsis measure in the direction of the spacecraft’s motion. This is not defined when inclination is 0 or 180 degree or eccentricity is 0. It lies between 0 to 360 degree.

*ω* = cos*−*1 *→−n .→−e* (7)

*|n||e|*

*where,*

*→−e* = *EccentricityV ector.*

* + 1. **True Anomaly(***ν***):** True Anomaly defines the position of the spacecraft w.r.t perigee. It’s the angle between the spacecraft and the perigee of the orbit. It lies between 0 to 360 degree.

*ν* = cos*−*1 *→−e .→−r* (8)

*|e||r|*

*where,*

*→−r* = *RadiusV ector*

## Alternate Orbital Elements

1. **Longitude of Perigee(***Pi***)** is the angle from the principle direction to perigee. This is used whenever inclination is either 0*o* or 180*o* as there is no ascending node. It’s lies between 00 to 3600. In the [figure(6)](#_bookmark18) Longitude of perigee is represented as “Π”.

Π = cos*−*1 *→−I .→−e* (9)

!

*|e|*

1. **True Longitude** is the angle from the principle direction to the spacecraft’s position. This is used whenever there is no perigee and the the inclination is either 00 or 1800. It’s lies between 00 to 3600. It lies between 0 to 360 degree. In the figure(6)True Longitude is represented as “*l*” In the figure(6)True Longitude is represented as “*l*”. In the figure(6) Longitude of perigee is represented as “Π”.

*l* = cos*−*1 *→−I .→−r* (10)

!

*|r|*

1. **Argument of Latitude(***u***)** is the angle from ascending to the spacecraft’s position. This is used whenever a perigee is absent(i.e., e=0,Circular Orbit). It’s lies between 00 to 3600.

In the figure(6) Argument of latitude is represented as “*u*”.

*u* = cos*−*1 *→−n .→−r* (11)

*|n||r|*

7

## 2D and 3D Orbits

Based on the inputs given the application can plot either 2D or 3D or both. If the inputs are just Semi major axis and eccentricity or Radius of perigee and apogee or, *r*1*, v*1*, γ*1 then the resultant plot will be in the perifocal frame. If the inputs are state and velocity vector or orbital elements then the resultant will be a 3D orbit.

**Inputs** For 2D - *a, e/ra, rp/r*1*, v*1*, γ*1

For 3D - Orbital Elements, State Vectors

**Outputs** Orbit in the Perifocal Frame

Orbit in a virtual 3D Environment Table 3: I/O for 2D and 3D orbits

There are two ways to plot a orbit, if the plot is on a 2D plane, i.e., Perifocal Plane, the the trajectory equation can be used to plot it easily.

*h*2*/µ*

*r* =

1 *− e* cos(*θ*)

If the plot is 3D then solving for the trajectory analytically can be tricky and expensive computing wise depending on the orbit. But, solving it numerically will yeild the orbit with wide range of accuracy depending on the numerical method. The method has mainly two properties to be considered for at this level, the computing speed and the accuracy. A good compromise between these both is desirable for most of the cases. In this case we need a ODE solver. Here are a few types of numerical methods:

* + 1. Euler Method
    2. Backward Euler Method
    3. First-order Exponential integrator Method
    4. Generalizations
    5. Parallel-in-time Methods
    6. Integrals over Infinite intervals

In a programming language based on these methods ODE solvers are written. The most popular ones are:

1. **lsoda** - This automatically selects a solver which is appropriate for the given equation.
2. **lsode** - Since lsoda turns out to be slow, then the DE might need a stiff solver(i.e., very small time steps), this will choose only the stiff solvers
3. **ode23** - Simultaneously uses fourth and fifth order RK formulas to make error estimates and adjust the time step accordingly. For nonstiff ODEs

8

1. **ode45** - Uses simultaneously second and third order Runge Kutta formulas to make estimates of the error, and calculate the time step size. Since the second and third order RK require less steps, ode23 is “less expensive” in terms of computation demands than ode45, but is also lower order. Use for nonstiff ODEs

For MOPy lsoda is sufficient enough for now. So lsoda is used to solve for the acceleration vector. If the acceleration vector is solved twice then we obtailn the position vectors using which a 3D plot can be plotted. The acceleration vector is:

*Entertheaccelerationvector*

## Euler Angle

The inputs for this are as in [table(4).](#_bookmark22) This code can be fed the model in 3D environment and the orientation of the model can be manipulated with this.

**Inputs** Directional Cosine Matrix Euler Angles

**Outputs** Euler Angles

Directional Cosine Matrix

Table 4: I/O for conversion between Euler angle and DCM

Euler Angles: The Euler angles are three angles introduced by Leonhard Euler to describe the orientation of a rigid body with respect to a fixed coordinate system. They can also represent the orientation of a mobile frame of reference in physics or the orientation of a general basis in 3-dimensional linear algebra. Euler angles can be defined by elemental geometry or by composition of rotations (matrix). The three elemental rotations may be extrinsic (rotations about the axes xyz of the original coordinate system, which is assumed to remain motionless), or intrinsic (rotations about the axes of the rotating coordinate system XYZ, solidary with the moving body, which changes its orientation after each elemental rotation).Euler angles are typically denoted as *ψ, θ, φ*

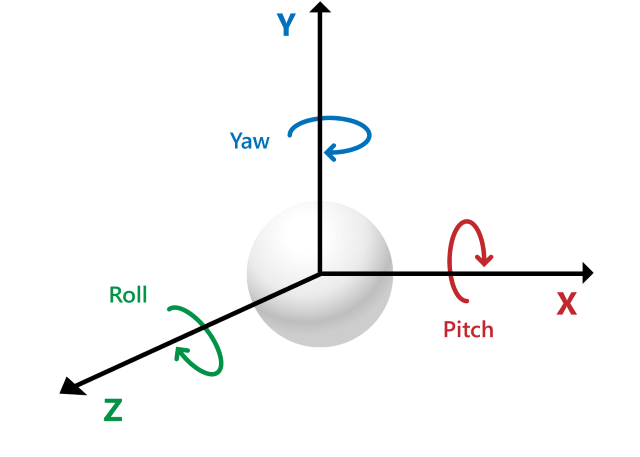


Figure 7: Euler Vector 9

## Sphere of Influence

In this section, the user can either input the values or interact with the model in the 3D environment to see the corresponding output.

**Inputs** Minor Body

**Outputs** Radius of SOI in desired units

3D visualization of sphere of influence in virtual environment Table 5: I/O for Sphere of Influence

The sun is the dominant celestial body in the solar system. It has a mass of over 300,000 earths. The sun’s gravitational pull holds all the planets in its hold according to Newton’s law of gravity. However, near a given planet, the influence of its own gravity exceeds that of the sun. For example, at its surface the earth’s gravitational force is over 1600 times greater than the sun’s. The inversesquare nature of the law of gravity means that the force of gravity *Fg* drops off rapidly with distance r from the center of attraction. If *Fg*0 is the gravitational force at the surface of a planet with radius *r*0, then Figure 8.5 shows how rapidly the force diminishes with distance. At ten body radii, the force is 1% of its value at the surface. Eventually, the force of the sun’s gravitational field overwhelms that of the planet.

To estimate the planets Sphere of Influence, the three body problem system comprising of planet p of mass MiB\_ mass, the sun s and its mass MaB\_ mass with the distance between then being R and the equation on hand is:

*rSOI* = *R*

*MiB*\_*mass* 2*/*5

*MaB*\_*mass*

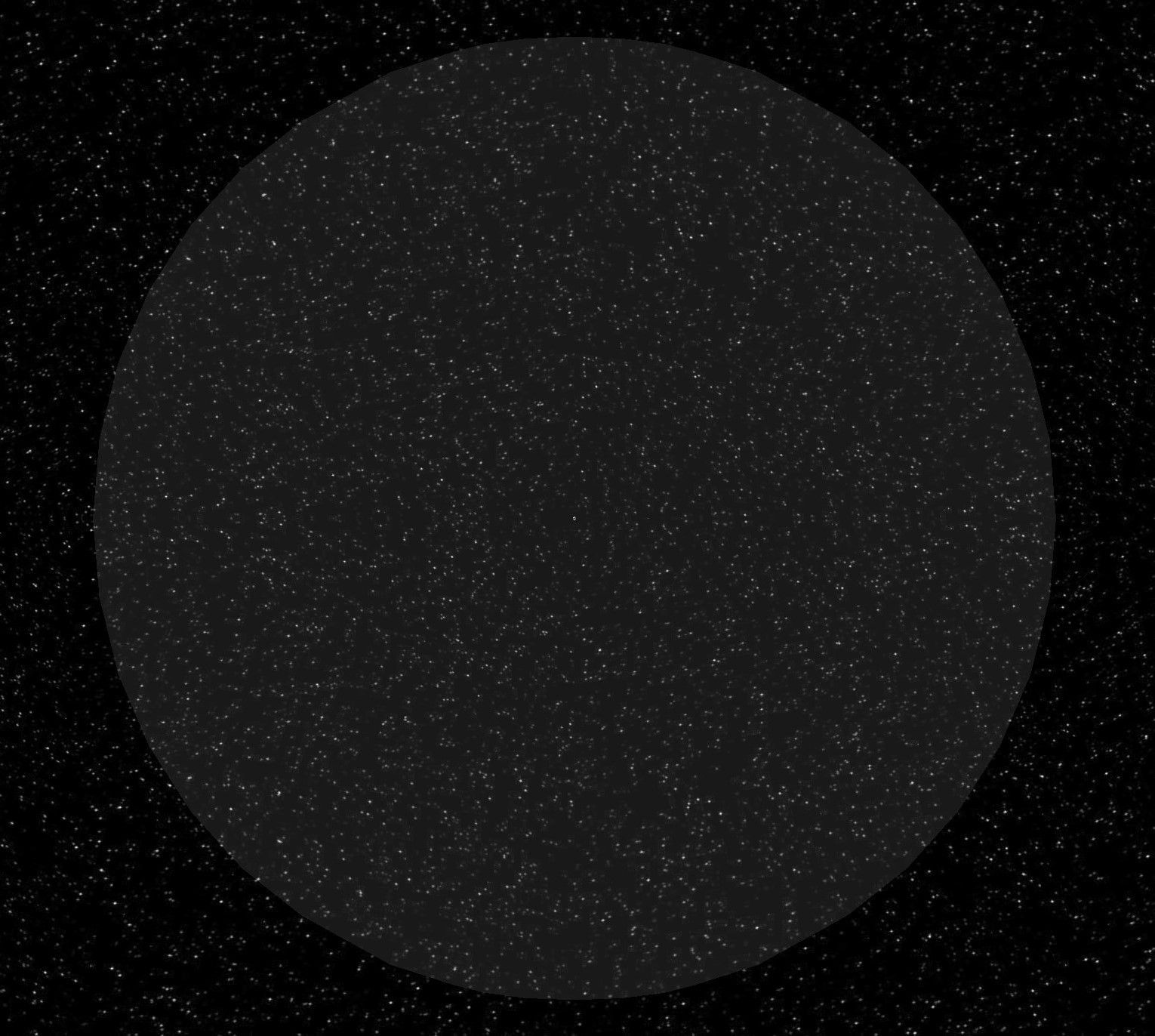


Figure 8: Sphere of Influence of Earth, which is approx 140 times it’s radius

10

## Orbital Transfer

This feature simulates the orbital transfer using various [numer](#_bookmark28)ical methods. For now, this can perform Hohmann Transfer. The Inputs and outputs are as in table(6)

**Inputs** Minor Body, Major Body, Orbital parameters of both initial and final orbit.

**Outputs** Values such as Radius of apogee, Radius of perigee, DeltaV, Time-period of the orbit etc 3D visualization of desired orbital transfer

Table 6: I/O for Orbital transfer

Transfer Orbit is nothing but the path followed by the satellite to travel from an initial orbit to final orbit or one planet to other planet. There are many types of transfer orbits in which most commonly used transfer is Hohmann Transfer because of its simplicity and its efficient to transfer. Transfer orbits are:

* + 1. **Hohmann Tranfer Orbit:** A Hohmann Transfer is a two-impulse(delta-v) elliptical transfer between two co-planar circular/elliptical orbits. The transfer orbit’s apoapsis lies on the target orbit and the periapsis of transfer orbit lies on the initial orbit.The fundamental assumption behind the Hohmann transfer, is that there is only one major body which exerts a gravitational force on the body of interest, such as a satellite. This is most commonly used transfer method for transferring an earth-based satellite from a low orbit to say a geosynchronous orbit.

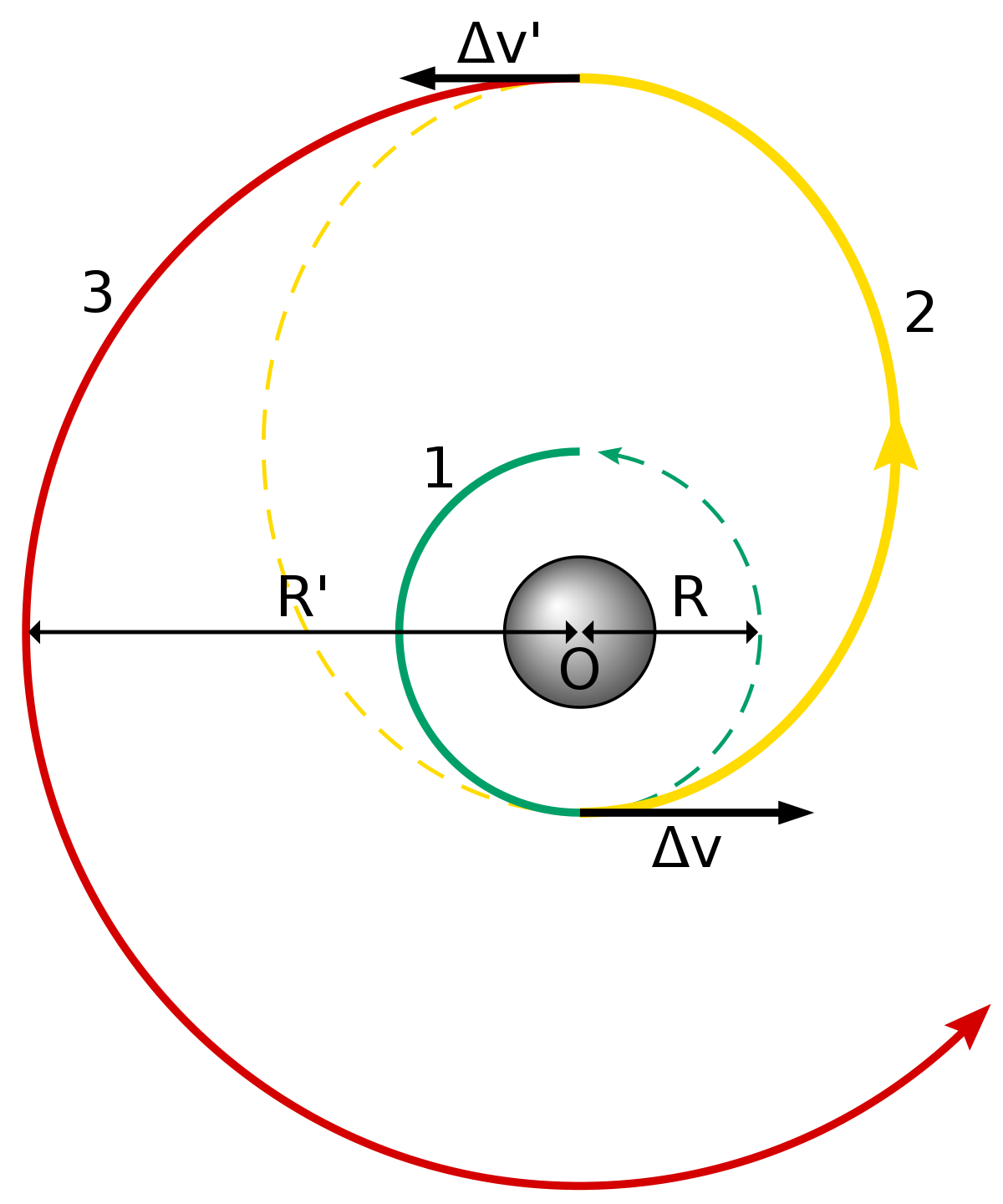


Figure 9: Hohmann Transfer

11

* + 1. **Bielliptical Transfer Orbit**: In the bi-elliptic transfer, the first transfer is a highly eccentric orbit with an apoapsis higher than the target orbit radius. Once the spacecraft has reached apoapsis, it performs a prograde burn raising its periapsis to the height of its target orbit. Finally, once it reaches periapsis, it performs an orbital insertion burn (Retrograde burn) to put it into a circular orbit. In some cases Bielliptical Transfer is preffered over Hohmann transfer because its slight fuel efficint transfer.

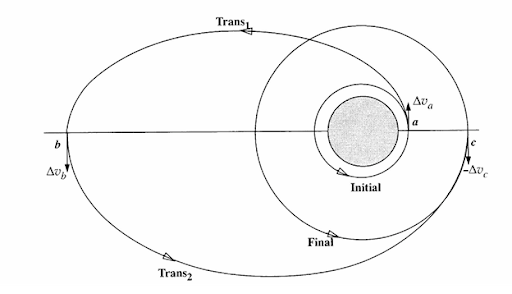


Figure 10: Bi-Elliptical Hohmann Transfer

* + 1. **Low Energy Transfer Orbit:** A low energy transfer is a path in space to reach a target with very little consumption of fuel.
    2. **Constant-Thrust Transfer Orbit:** In this transfer path the spacecraft’s engine will be producing constant thrust through out the path. The spacecraft points straight towards the target with constant acceration till it reaches the target.

## Calculation of Julian Day

In this section the user can choose between Gregorian calendar and Julian calendar and with the other inputs they can obtain the Julian day of the corresponding date.

**Inputs** YYYY-MM-DD , hh:mm:ss, Type of calender

**Outputs** Julian-day

Table 7: I/O for Julian-day

12

The Julian day is the continuous count of days since the beginning of the Julian period i.e., January 1, 4713 B.C. at 12 Noon, and is used primarily by astronomers for easily calculating elapsed days between two events. To calculate a Julian Day there are many formulae, but the generalised formula is: To convert from Gregorian calendar:

12

*−*

*JDN* =

 

1461 *Y* + 4800 + *M −* 14

12

+

4

367 *M −* 2 *−* 12 *M −* 14

12

To convert from Julian Calendar:

*Y* + 4900 + *M −* 14 100

4 + *D −* 32075

3  12 )

7 *Y* + 5001 + *M −* 9

*JDN* = 367 *× Y −*

7 275*M*

+ + *D* + 1729777

4 9

Once the Julian Day Number is obtained the time is taken into account. So, The Julian day becomes:

*JD* = *JDN* + *HH −* 12 + *MM* + *SS*

24 1440 86400

## Calculation of Parameters of the Orbit

The user can obtain various parameters by giving the details that they know of. If necessary they can plot the orbit too.

**Inputs** *a, e/ra, rp/r*1*, v*1*, γ*1/ Orbital Elements/State and velocity vectors

**Outputs** *µ, , h , ϵ*, Forces and Velocity at significant position Mean Motion, Time Period

Table 8: I/O for Various Parameters of the Orbit

The Semi major axis ,Eccentricity, Radius of perigee, Radius of apogee, Specific mechanical energy, Specific angular momentum, Time period , Mean motion, Velocity at perigee, Velocity at apogee, Gravitational force at perigee, Gravitational force at apogee, Semi-latus rectum, Velocity at semi-latus rectum, Escape velocity at perigee and Escape velocity at apogee all these parameters can be calculated. These parameters can be calculated at any given point of time in the ongoing mission. The major formula used in it are:

13

### Circular and Elliptical Parabolic Hyperbolic

*µ* = *G × MMajor*

*rp* = *a ×* (1 *− e*)

*l* = 2 *× rp*

*θ∞* = cos*−*1

1

( ) = *β*

*e*

1

*ra* = *a ×* (1 + *e*)

*ϵ* = *−µ*

2*a*

*h* = √*rp ×* (1 + *e*)*µ*

0

*h* = 2*µrp*

*GMm*

√

*δ* = 2 sin( )

*h* = √*µa*(*e*2 *−* 1)

*e*

*b* = *ae* sin(*θ∞*) = ∆

*Fg* =

*r*2

r

*V* = r2*µ − µ*

s4*π*2

*r a*

*×*

*V* = 2*µ*

*r r*

*V∞* = *eµ* sin(*θ∞*)*/h*

*µ*

*T* = *a*3

*µ*

2*π*

*n* =

*T*

*ϵ* =

2*a*

*b*2

*−*

*l* =

*a*

*h*2*/µ*

*l* = *a ×* (1 *− e*2)

*rp, ra* =

1 *± e*

Table 9: Formulae for Various parameters

## Sensitivity Analysis

This calculates the value of the error that rises up if the initial value is very slightly different than what it is supposed to be.

**Input** State Vector, Velocity Vector and two delta-v with slight difference

Percentage difference caused to the final orbit parameters due to slight difference in delta-v

**Output**

Table 10: I/O for Sensitivity Analysis

The analysis is done to know what is the effect, if small error occurs in position and velocity at the maneuver point while the space mission is on the trajectory. There is a derived formula to calculate the sensitivity analysis from which change in position and velocity can be obtained in terms of percentage . This helps in analysing the position and velocity tolerance of any particular mission and design accordingly.





2*µ*1

*v∞* +

+



*δR*2 = 2 *µ*1 *δrp*  *rpv∞ δvp*

*R*2 *R*1 [*V v*]2  *V vv∞rp rp*

1 *−*

*D*

2*µsun*

*D*

*D*

*V v vp* 

14

## Position of One Spacecraft Relative To Another

Based on the inputs of the user the relative velocity and orbit can be visualized in this section.

**Input** Major Body, State Vector and Velocity Vector of Minor Body

**Output** Graph Showing the Minor bodies in the orbit around Major Body Table 11: I/O for Position of One Spacecraft Relative To Another

In situations like a rendezvous maneuver the relative motion between the space vehicles is utilised. This can be done in two different ways, utilising the algorithm to calculate the relative orbit or utilise the camera viewpoint and plotting as the orbit propagates with the camera fixed on one of the spacecraft.

## Lagrangian Points

The Lagrangian points are points near two large orbiting bodies, the two objects exert an unbalanced gravitational force at a point, altering the orbit of whatever is at that point. At the Lagrange points, the gravitational forces of the two large bodies and the centrifugal force balance each other, The inputs and outputs are as follows.

**Input** Major Body, Minor Body and Distance Between them

**Output** Lagrangian points polar coordinates and Graph showing them Table 12: I/O for Lagrangian Points

Lagrange Points are positions in space around the planet/star, where the gravitational forces of two body like the Sun and the Earth produce enhanced regions of attraction and repulsion. These can be used by spacecraft to reduce fuel consumption required to remain in position. At these points gravitational pull of two large masses precisely equals the centripetal force required for a small object to move with them.

These points are named in honor of Italian-French mathematician Josephy-Louis Lagrange. There are five Langrengian points around the planet/satr, where a small mass can orbit in a constant pattern with two larger masses. This mathematical problem, known as the "General Three-Body Problem" was considered by Lagrange in his prize winning paper (Essai sur le Problème des Trois Corps, 1772). Of the five Lagrange points, three are unstable and two are stable. The unstable Lagrange points - labeled L1, L2 and L3 - lie along the line connecting the two large masses.The three of them lie along the line connecting the two large bodies. The stable Lagrange points - labeled L4 and L5 - form the apex of two equilateral triangles that have the large masses at their vertices. L4 leads the orbit of earth and L5 follows.

15

# Database Management System

## Introduction

Database is a collection of set of related data or information of any particular concept, generally stored using any electronic gadget . Database Management system is a system which uses integrated software to connect frontend user to connect to the database to access , modify and manage the data stored in it. There are two types of Databases:

* + 1. Relational Database - Organizes data into one or more table. each table comprising of number of rows and columns with each row identifiable with unique key.
    2. Non - Relational database - Organizes data in any form other than table. Such as graphs, flexible tables, documents etc.

In this project we have used Relational database management system with structured Query Language (SQL) to interact with the RDBMS. SQLite is the database engine used to run SQL query to perform CRUD ( Create , Retrieve, Update or Delete) operation in the database.

This database contains the data of planetary bodies such as Mass, radius, density, gravity, temperature, orbital parameters, number of moons etc., of all planets of our solar system.

# Conclusion

The outcome of the project is the lessons we learnt during the project. Communication is the key for collaborative work. This is one of the main aspect that we learned during this project. The other aspects that we were able to learn are:

1. Project Management
2. Time Management
3. Prioritize the list that is at hand based on various parameters

We were able to learn many libraries that are related to our field in python like SciPy, NumPy amongst others. These are extremely useful when we are conducting numerical analysis and numerical simulations. This gave the foundation for other few libraries that might be useful, like the Qt Library, SQlite3.

Panda3D was the library that was the most interesting library followed by SciPy, this enabled us to illustrate the ideas we had in our mind through the screen. This gave us much more potential to continue on many more endeavors.

16

# Future Scope

MOPy is a open source application with GNU - GPL v3 License, with this anyone who is interested in using this application can freely download and use it, and even modify the contents to their requirements. If they wish to contribute to this application they can fork the repository on GitHub[3] and add their contribution by sending a pull request. The reason GNU-GPL v3 license is used is that we aim to keep a quality control to the features that gets added to the application. We are always open to contributors.

There is a list of features with corresponding priorities that are planned to be added down the line. This will be constantly updated indefinitely. And if the user is not able to add a feature then they can request a feature that would be added to the list with a appropriate priority. There is discord server in which anyone can join and hold discussions with other participants. Contributors can hold discussions about the feature they are planning to add, others can discuss about the concepts and request features to be added.

As of now this runs on Windows only. This could be made to run on other platforms and even as a web application which would negate the need of a moderately powerful hardware. The database could be expanded with much more data than present.

This entire code can be converted into a library that anyone can download from PyPI. This would enable the user to utilize the features directly from the command line or their python projects. This would simplify their projects.

17

# Code for the Features

## Calculation Of Orbital Elements

1 from numpy. linalg import norm

2 from numpy import dot, pi, cross , multiply as multi

3 from math import acos

4

5 class Calculate:

6 I = [1, 0, 0]

7 J = [0, 1, 0]

8 K = [0, 0, 1]

9 G = 6.67e-20 # units are in km3 kg-1 s-2

10

11 @classmethod

12 def muvalue(cls, major\_body):

13 major\_body\_list = pandas. read\_csv(" Major\_and\_Minor\_Bodies. csv")

14 major\_bodies = list( major\_body\_list. Major\_Body)

15 major\_bodies\_mass = list( major\_body\_list. Mass)

16 major\_bodies\_radius = list( major\_body\_list. Radius)

17 sel\_major\_body = major\_bodies. index( major\_body)

18 major\_body\_mass = major\_bodies\_mass[ sel\_major\_body]

19 major\_body\_radius = major\_bodies\_radius[ sel\_major\_body]

20 mu = cls.G \* major\_body\_mass

21 return [mu, major\_body\_radius]

22

23 @classmethod

24 def correct\_ohm(cls, ohm, n\_vec):

25 if n\_vec[0] > 0 and n\_vec[1] > 0:

26 print(" This is a Prograde Elliptical Orbit and is in first Quadrant.")

27 if ohm >90:

28 ohm -= 360

29 elif n\_vec[0] < 0 and n\_vec[1] > 0:

30 print(" This is a Retrograde Elliptical Orbit and is in second Quadrant."

)

31 if ohm > 180:

32 ohm -= 360

33 elif n\_vec[0] < 0 and n\_vec[1] < 0:

34 print(" This is a Retrograde Elliptical Orbit and is in Third Quadrant.")

35 if ohm < 180:

36 ohm -= 360

37 elif n\_vec[0] > 0 and n\_vec[1] < 0:

38 print(" This is a Prograde Elliptical Orbit and is in fourth Quadrant.")

39 if ohm < 90:

40 ohm -= 360

41 return ohm

42

43 @classmethod

44 def other\_var(cls, pos\_vec , vel\_vec):

45 h\_vec = cross( pos\_vec , vel\_vec)

46 n\_vec = cross( cls.K, h\_vec)

47 return [ h\_vec , n\_vec]

48

49 @classmethod

50 def OE(cls, pos\_vec , vel\_vec , mu):

18

51 [ h\_vec , n\_vec] = Calculate. other\_var( pos\_vec , vel\_vec)

52 e\_vec = ( multi(( norm( vel\_vec)\* norm( vel\_vec)-(mu/ norm( pos\_vec))), pos\_vec)- multi( dot( pos\_vec , vel\_vec), vel\_vec))/( mu)

53 call. ecc\_o. setText( str( norm( e\_vec)))

54 inc = ( acos(( dot( h\_vec , cls.K))/ norm( h\_vec))) \* 180/ pi

55 call. inc\_o. setText( str( inc))

56 sma = 1/((2/ norm( pos\_vec))-(( norm( vel\_vec)\* norm( vel\_vec))/mu))

57 call. sma\_o. setText( str( sma))

58 return [sma, inc, e\_vec]

59

60 @classmethod

61 def ACOE(cls, pos\_vec , vel\_vec , e\_vec , inc):

62 [ h\_vec , n\_vec] = Calculate. other\_var( pos\_vec , vel\_vec)

63 if inc != 0 or 180 and norm( e\_vec) > 0: # Nothing is Zero/180

64 ohm = ( acos(( dot( cls.I, n\_vec))/ norm( n\_vec)))

65 ohm = Calculate. correct\_ohm(ohm, n\_vec)

66 call. ohm\_o. setText( str( ohm))

67 nu = ( acos(( dot( e\_vec , pos\_vec))/( norm( e\_vec)\* norm( pos\_vec))))

68 call. nu\_u\_o. setText( str(nu))

69 omega = ( acos(( dot( n\_vec , e\_vec))/ multi( norm( n\_vec),norm( e\_vec))))

70 call. omega\_l\_o. setText( str( omega))

71 return [ohm, omega , nu]

72 elif inc == 0 or 180: # Inclination is Zero

73 if norm( e\_vec) > 0: # Elliptical Orbit

74 nu = ( acos(( dot( e\_vec , pos\_vec))/( norm( e\_vec)\* norm( pos\_vec))))

75 call. nu\_u\_o. setText( str(nu))

76 Long\_of\_peri\_pi = acos( dot( cls.I, e\_vec)/( norm( cls.I)\* norm( e\_vec)))

77 return [ Long\_of\_peri\_pi , nu]

78 elif norm( e\_vec) == 0: # Circular Orbit

79 Tr\_long\_l = acos( dot( cls.I, pos\_vec)/( norm( pos\_vec)\* norm( cls.I)))

80 return [ Tr\_long\_l]

81 elif norm( e\_vec) == 0: # Circular orbit with inclination non- zero/pi

82 Arg\_of\_lattitude\_u = acos( dot( n\_vec , pos\_vec)/( norm( n\_vec)\* norm( pos\_vec))

)

83 return [ Arg\_of\_lattitude\_u]

84

85 @classmethod

86 def possibility(cls, major\_body , pos\_vec , vel\_vec):

87 [mu, major\_body\_radius] = Calculate. muvalue( major\_body)

88 pos = norm( pos\_vec)

89 vel = norm( vel\_vec)

90 sma = 1/((2/ pos)-( vel\* vel/mu))

91 e\_vec = ( multi(( norm( vel\_vec)\* norm( vel\_vec)-(mu/ norm( pos\_vec))), pos\_vec)- multi( dot( pos\_vec , vel\_vec), vel\_vec))/( mu)

92 r\_peri = sma\*(1 - norm( e\_vec))

93 if r\_peri <= major\_body\_radius:

94 print(" The Orbit is not possible as radius of perigee is less than radius of the major Body")

95 return False

96 return True

Listing 1: CoOE.py

19

## Euler Angles

1 from math import \*

2 from numpy import \*

3

4 class EA():

5

6 def RxRyRz( pitch\_theta , roll\_phi , yaw\_si):

7 Rx = matrix([[ 1, 0 , 0 ],

8 [ 0, cos( roll\_phi),sin( roll\_phi)],

9 [ 0,- sin( roll\_phi), cos( roll\_phi)]])

10

11 Ry = matrix([[ cos( pitch\_theta), 0,- sin( pitch\_theta)],

12 [ 0 ,1, 0],

13 [ sin( pitch\_theta), 0, cos( pitch\_theta)]])

14

15 Rz = matrix([[ cos( yaw\_si), sin( yaw\_si), 0 ],

16 [- sin( yaw\_si), cos( yaw\_si) , 0 ],

17 [ 0, 0, 1]])

18 return [Rx, Ry, Rz]

19

20 def DCMtoEA(DCM, order):

21 if order==’321 ’:

22 theta=- asin( DCM[0][2])

23 phi=atan( DCM[1][2]/ DCM[2][2])

24 si=atan( DCM[0][1]/ DCM[0][0])

25 elif order==’123 ’:

26 theta=asin( DCM[2][0])

27 phi=atan( DCM[2][1]/ DCM[2][2])

28 si=atan( DCM[1][0]/ DCM[0][0])

29 elif order==’132 ’:

30 theta=atan( DCM[2][0]/ DCM[0][0])

31 phi=atan( DCM[1][2]/ DCM[1][1])

32 si=- asin( DCM[1][0])

33 elif order==’312 ’:

34 theta=atan( DCM[0][2]/ DCM[2][2])

35 phi=asin( DCM[1][2])

36 si=atan( DCM[1][0]/ DCM[1][1])

37 elif order==’213 ’:

38 theta=atan( DCM[2][0]/ DCM[2][2])

39 phi=- asin( DCM[2][1])

40 si=atan( DCM[0][1]/ DCM[1][1])

41 elif order==’231 ’:

42 theta=atan( DCM[0][2]/ DCM[0][0])

43 phi=atan( DCM[2][1]/ DCM[1][1])

44 si=asin( DCM[0][1])

45 return [ theta , si, phi]

46

47 def EAtoDCM( theta , si, phi, order):

48 [Rx, Ry, Rz] = EA. RxRyRz( theta , si, phi)

49 if order==’321 ’:

50 RxRy = dot(Rx,Ry)

51 return ( dot(RxRy ,Rz))

52 elif order == ’123 ’:

53 x = Rx( phi\*pi/180)

54 y = Ry( theta\*pi/180)

55 z = Rz(si\*pi/180)

20

|  |  |  |
| --- | --- | --- |
| 56 | zy = dot(z,y) |  |
| 57 | return ( dot(zy,x)) |
| 58 | elif order == ’213 ’: |
| 59 | x = Rx( phi\*pi/180) |
| 60 | y = Ry( theta\*pi/180) |
| 61 | z = Rz(si\*pi/180) |
| 62 | zx = dot(z,x) |
| 63 | return ( dot(zx,y)) |
| 64 | elif order == ’312 ’: |
| 65 | x = Rx( phi\*pi/180) |
| 66 | y = Ry( theta\*pi/180) |
| 67 | z = Rz(si\*pi/180) |
| 68 | yx = dot(y,x) |
| 69 | return ( dot(yx,z)) |
| 70 | elif order == ’132 ’: |
| 71 | x = Rx( phi\*pi/180) |
| 72 | y = Ry( theta\*pi/180) |
| 73 | z = Rz(si\*pi/180) |
| 74 | yz = dot(y,z) |
| 75 | return ( dot(yz,x)) |
| 76 | elif order == ’231 ’: |
| 77 | x = Rx( phi\*pi/180) |
| 78 | y = Ry( theta\*pi/180) |
| 79 | z = Rz(si\*pi/180) |
| 80 | xz = dot(x,z) |
| 81 | return ( dot(xz,y)) |
| 82 | else: |
| 83 | return (’No transformation | matrix’) |
| 84 |  | |
| 85 | order = ’321 ’ # input(" Enter the order:") | |
| 86  87 | DCM = [[0.6405 , 0.75319 , -0.15038],[0.76736 , -0.63531 , 0.086824],[-0.030154 ,  -0.17101 , -0.98481]]  print(EA. DCMtoEA(DCM, order)) | |
| 88 |  | |
| 89 | print(EA. EAtoDCM (8.6489 \* 180/ pi, 49.619 \* 180/ pi, -5.039 \* 180/ pi, order)) | |

Listing 2: EA.py

## Sphere of Influence

1

from math import \*

import numpy as np

import matplotlib. pyplot as plt

from mpl\_toolkits. mplot3d import Axes3D import sqlite3

def SoI( MiB\_mass , MiB\_radius , r\_maj\_to\_min):

MaB\_mass = 1.989 e30

rSOI = ( r\_maj\_to\_min\*( MiB\_mass/ MaB\_mass)\*\*(2/5)) return [rSOI , rSOI/ MiB\_radius]

2

3

4

5

6

7

8

9

10

Listing 3: SoI

21

## Calculation of Julian Day

1

from math import \*

class JulianDay():

@classmethod

def tb\_method(cls,year , month , day, hour , minutes , seconds , accuracy): if 1901 <= year <= 2099:

JDN = 367 \* year + int((7 \* ( year + int(( month + 9)/12)))/4) + int(275 \* month/9) + day + 1721013.5

UT = hour + round(( minutes/60), accuracy) + round(( seconds/3600), accuracy)

JD = JDN + UT/24

return [JDN, JD] else:

# manju you can set limits when this option is selected between 1901 and

2099

print(" Not Possible from this method")

@classmethod

def julian\_day(cls,year , month , day, hour , minutes , seconds , accuracy , type\_of\_calender):

if type\_of\_calender == " Gregorian":

# julian day number from Gregorian calender date

JDN = int((1461 \* ( year + 4800 + ( month - 14)/12))/4) + int((367 \* month

- 2 - 12 \* (( month - 14)/12))/12) - int((3 \* (( year + 4900 + ( month - 14)/12)

/100))/4) + day - 32075

elif type\_of\_calender == " Julian":

# julian day number from julian calender date

JDN = 367 \* year - int((7 \* ( year + 5001 + ( month - 9)/7))/4) + int((275

\* month)/9) + day + 1729777 # JDN to JD

JD = JDN + round((( hour - 12)/24), accuracy) + round(( minutes/1440), accuracy

) + round(( seconds /86400), accuracy) return [JDN, round(JD, accuracy)]

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

Listing 4: CJSD.py

## Calculation of Various Parameters of the Orbit

1

import numpy as np

from math import \*

class GlobalCalculation(): I = [1, 0, 0]

J = [0, 1, 0]

K = [0, 0, 1]

G = 6.67e-20 # units are in km3 kg-1 s-2

@classmethod

def force\_at\_this\_point(cls, r, major\_body\_mass , minor\_body\_mass): F\_r = cls.G \* major\_body\_mass \* minor\_body\_mass / r \*\* 2 return F\_r

2

3

4

5

6

7

8

9

10

11

12

13

14

15

22

16 # all the units should be km

17 class CalculateCircularElliptical:

18 I = [1, 0, 0]

19 J = [0, 1, 0]

20 K = [0, 0, 1]

21 G = 6.67e-20 # units are in km3 kg-1 s-2

22 def

23

24

25

26

\_\_init\_\_(self , major\_body):

self. major\_body = major\_body return

27 @classmethod

28 def semiecc(cls, sma, mag\_e , major\_body):

29 r\_per = sma \* (1 - mag\_e)

30 r\_apo = sma \* (1 + mag\_e)

31 [ mean\_motion , T\_period , mag\_h , sme, slr] = CalculateCircularElliptical. orb\_const( r\_per , sma, mag\_e , major\_body)

32 return [ r\_per , r\_apo , mean\_motion , T\_period , mag\_h , sme, slr]

33

34 @classmethod

35 def perapo(cls, r\_per , r\_apo , major\_body):

36 sma = ( r\_per + r\_apo)/2

37 mag\_e = ( r\_apo - r\_per)/( r\_apo + r\_per)

38 [ mean\_motion , T\_period , mag\_h , sme, slr] = CalculateCircularElliptical. orb\_const( r\_per , sma, mag\_e , major\_body)

39 return [ mag\_e , sma, mean\_motion , T\_period , mag\_h , sme, slr]

40

41 @classmethod

42 def orb\_const(cls, r\_per , sma, mag\_e , major\_body):

43 major\_body\_mass = 5.972 e24 # self. major\_body \* 1

44 mu = cls.G \* major\_body\_mass

45 slr = sma\*(1 - mag\_e\*\*2)

46 mag\_h = ( r\_per\*(1+e)\*mu)\*\*0.5

47 T\_period = sqrt(((4\* pi\*\*2)/mu)\* sma\*\*3)

48 mean\_motion = T\_period/(2\* pi)

49 sme = -mu/(2\* sma)

50 return [ mean\_motion , T\_period , mag\_h , sme, slr]

51

52 @classmethod

53 def time\_since\_periapsis(cls, mag\_e , theta , T\_period):

54 E\_anomly = 2 \* atan(( sqrt((1 - mag\_e)/(1 + mag\_e)) \* tan( theta/2)))

55 M\_anomly = E\_anomly - mag\_e \* sin( E\_anomly)

56 t\_since\_perigee = T\_period \* M\_anomly/(2\* pi)

57 return t\_since\_perigee

58

59 @classmethod

60 def velocity\_at\_any\_point(cls, sma, r, major\_body):

61 major\_body\_mass = major\_body \* 1

62 mu = cls.G \* major\_body\_mass

63 v\_point = sqrt(((2 \* mu)/r) - (mu/ sma))

64 return v\_point

65

66 class CalculateParabola():

67 I = [1, 0, 0]

68 J = [0, 1, 0]

69 K = [0, 0, 1]

70 G = 6.67e-20 # units are in km3 kg-1 s-2

23

71

@classmethod

def const\_values(cls, r\_per , major\_body): major\_body\_mass = major\_body \* 1

mu = major\_body\_mass \* cls.G slr = 2 \* r\_per

v\_per = sqrt((2\* mu)/ r\_per) mag\_h = sqrt(mu\* r\_per \* 2) return [slr, v\_per , mag\_h]

@classmethod

def velocity\_at\_any\_point(cls, r, major\_body): major\_body\_mass = major\_body \* 1

mu = major\_body\_mass \* cls.G v = sqrt((2\* mu)/r)

return v

class CalculateHyperBola(): I = [1, 0, 0]

J = [0, 1, 0]

K = [0, 0, 1]

G = 6.67e-20 # units are in km3 kg-1 s-2

@classmethod

def const\_values(cls, major\_body , mag\_e , sma): mu = cls.G \* major\_body

theta\_inf = acos(1/ mag\_e) # angle between asympotes beta = theta\_inf

delta = 2 \* sin(1/ mag\_e) # turn angle

mag\_h = sqrt( sma \* mu \* ( mag\_e\*\*2 -1)) # specific angular momentum b = sma \* mag\_e \* sin( theta\_inf) # semi minor axis

Delta = b # aiming radius

v\_inf = mu \* mag\_e\* sin( theta\_inf)/ mag\_h # velocity at r = infinity sme = mu/(2 \* sma) # specific mechanical energy

slr = -b\*\*2/ sma # semi latus rectum

return [ theta\_inf , beta , delta , mag\_h , b, Delta , v\_inf , sme, slr]

@classmethod

def semiecc(cls, major\_body ,sma, mag\_e): mu = cls.G \* major\_body

[ theta\_inf , beta , delta , mag\_h , b, Delta , v\_inf , sme, slr] = CalculateHyperBola. const\_values( major\_body , sma, mag\_e)

r\_apo = ( mag\_h\*\*2/ mu)/(1 - mag\_e) r\_per = ( mag\_h\*\*2/ mu)/(1 + mag\_e)

return [ theta\_inf , beta , delta , mag\_h , b, Delta , v\_inf , sme, slr, r\_apo , r\_per]

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

101

102

103

104

105

106

107

108

109

110

111

112

113

114

Listing 5: VPCO

24

## Sensitivity Analysis

1

class SA( Major\_Body , R1\_bod):

def \_\_init\_\_( self):

G = 6.67e-20

mu\_Major\_Body = G \* self. R1\_bod \* 1 mu\_R1 = G \* self. R2\_bod \* 1

def SA(self , R1 , R2):

a = 2/(1 -( R1\* Vdv\*\*2)/(2\* self. mu\_Major\_Body)) coeff\_del\_Rp\_Rp = a \* ( self. mu\_R1 /( Vdv\*Vi\*Rp)) coeff\_del\_Vp\_Vp = a \* (( Vi+(2\* self. mu\_R1 /( Rp\*Vi)))/ Vdv)

del\_R2\_R2 = coeff\_del\_Rp\_Rp\*( del\_Rp\_Rp) + coeff\_del\_Vp\_Vp \* ( del\_Vp\_Vp) return [ del\_R2\_R2 ]

2

3

4

5

6

7

8

9

10

11

12

Listing 6: SA.py

## Lagrangian Points

1

from math import pi

from numpy import arange

import matplotlib. pyplot as plt

from scipy. optimize import root\_scalar

G = 6.67408e-11 m1 = 5.972 e24 m2 = 0.07346 e24 l = 384400000

x1 = m2 \* l / (m1 + m2) x2 = m1 \* l / (m1 + m2)

velocity = (G \* (m1 + m2)/ l) \*\* 0.5 period = 2 \* pi \* l / velocity theta\_dot = 2\* pi / period

def x\_eqn(xs):

return -G\*m1 /(( xs + x1) \* abs(xs +x1)) - G \* m2 / (( xs - x2)\* abs(xs-x2)) + \ theta\_dot\*\*2 \* xs

xvals = arange(-2\*l, 2\*l, 1000) yvals = x\_eqn( xvals)

L1 = root\_scalar( x\_eqn , bracket=[2 e8 , 3.5 e8 ]) print(L1. root+x1)

L2 = root\_scalar( x\_eqn , bracket=[3.9 e8 , 5e8 ]) print(L2. root+x1)

L3 = root\_scalar( x\_eqn , bracket=[-4e8 , -2e8 ]) print(L3. root+x1)

plt. plot( xvals , yvals) plt. grid()

plt. ylim(-0.01 , 0.01) plt. show()

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

Listing 7: LP.py

25

## Calculation Of Relative Position of two Spacecrafts

1

from math import \*

from numpy import \*

from numpy. linalg import norm

hA= cross(rA,vA) mag= norm(hA)

i= rA/ norm(rA) k= hA/ norm(hA) j= cross(k,i) Q=[i,j,k] print(Q)

omega=hA/( norm(rA)\*\*2) r\_relative=subtract(rB,rA) print( r\_relative)

v\_relative=( subtract(vB,vA))-( cross( omega , r\_relative)) print( v\_relative)

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

Listing 8: CoSV.py

## CRUD File Operations

1

# import sqlite in ide

import sqlite3

# Create a database or connect to one conn= sqlite3 . connect(’ appdatabase.db’)

# Create a Cursor c= conn. cursor()

# Creat a table

c. execute(""" CREATE TABLE table\_name( column\_name datatype , column\_name datatype

)""")

# Insert data into table

x = [(’ column 1’, ’ column 2’)

(’ column 1’, ’ column 2’)

]

c. executemany( " INSERT INTO table\_name VALUES(?,?)",x)

# Retrieve or read data c. execute(" SELECT \* FROM table\_name") print(c. fetchall())

# commit data to the database conn. commit()

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

Listing 9: db.py

26

# A Database:

27

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Mercury** | **Venus** | **Earth** | **Moon** | **Mars** | **Jupiter** | **Saturn** | **Uranus** | **Neptune** | **Pluto** |
| **Mass(**1024*kg***)** | 0.330 | 4.87 | 5.97 | 0.073 | 0.642 | 1898 | 568 | 86.8 | 102 | 0.0146 |
| **Diameter(***km***)** | 4879 | 12,104 | 12,756 | 3475 | 6792 | 142,984 | 120,536 | 51,118 | 49,528 | 2370 |
| **Density(***kg/m*3**)** | 5427 | 5243 | 5514 | 3340 | 3933 | 1326 | 687 | 1271 | 1638 | 2095 |
| **Gravity(***m/s*2**)** | 3.7 | 8.9 | 9.8 | 1.6 | 3.7 | 23.1 | 9.0 | 8.7 | 11.0 | 0.7 |
| **Escape Velocity(***km/s***)** | 4.3 | 10.4 | 11.2 | 2.4 | 5.0 | 59.5 | 35.5 | 21.3 | 23.5 | 1.3 |
| **Rotation Period(***hours***)** | 1407.6 | -5832.5 | 23.9 | 655.7 | 24.6 | 9.9 | 10.7 | -17.2 | 16.1 | -153.3 |
| **Length of Day(***hours***)** | 4222.6 | 2802.0 | 24.0 | 708.7 | 24.7 | 9.9 | 10.7 | 17.2 | 16.1 | 153.3 |
| **Distance from Sun(**106*km***)** | 57.9 | 108.2 | 149.6 | 0.384 | 227.9 | 778.6 | 1433.5 | 2872.5 | 4495.1 | 5906.4 |
| **Perihelion(**106*km***)** | 46.0 | 107.5 | 147.1 | 0.363 | 206.6 | 740.5 | 1352.6 | 2741.3 | 4444.5 | 4436.8 |
| **Apohelion(**106*km***)** | 69.8 | 108.9 | 152.1 | 0.406 | 249.2 | 816.6 | 1514.5 | 3003.6 | 4545.7 | 7375.9 |
| **Orbital Period(***days***)** | 88.0 | 224.7 | 365.2 | 27.3 | 687.0 | 4331 | 10,747 | 30,589 | 59,800 | 90,560 |
| **Orbital Velocity(***km/s***)** | 47.4 | 35.0 | 29.8 | 1.0\* | 24.1 | 13.1 | 9.7 | 6.8 | 5.4 | 4.7 |
| **Orbital Inclination(***degrees***)** | 7.0 | 3.4 | 0.0 | 5.1 | 1.9 | 1.3 | 2.5 | 0.8 | 1.8 | 17.2 |
| **Orbital Eccentricity(***e***)** | 0.205 | 0.007 | 0.017 | 0.055 | 0.094 | 0.049 | 0.057 | 0.046 | 0.011 | 0.244 |
| **Obliquity to Orbit(***degrees***)** | 0.034 | 177.4 | 23.4 | 6.7 | 25.2 | 3.1 | 26.7 | 97.8 | 28.3 | 122.5 |
| **Mean Temperature(***oC***)** | 167 | 464 | 15 | -20 | -65 | -110 | -140 | -195 | -200 | -225 |
| **Surface Pressure(***bars***)** | 0 | 92 | 1 | 0 | 0.01 | Unknown | Unknown | Unknown | Unknown | 0.00001 |
| **Number of Moons** | 0 | 0 | 1 | 0 | 2 | 79 | 82 | 27 | 14 | 5 |
| **Global Magnetic Field** | Yes | No | Yes | No | No | Yes | Yes | Yes | Yes | Unknown |

Table 13: Planetary Details

# References

1. Howard D. Curtis ,*“Orbital Mechanics for Engineering Students”*, 3rd Edition, Elsevier Publications, 2014
2. Bate, Roger R., Donald D. Muller and Jerry E. White, *“Fundamentals Of Astrodynamics”*,New York, NY, Dover Publications,1971.
3. *“Space:* [*Investing in the F*](https://www.morganstanley.com/ideas/investing-in-space)*inal Frontier”*,Jul, 24*th*, 2020, Retrieved on 25*th* May 2021 from Morgan Stanley, MorganStanley.com
4. *“Numerical Solutions to Differential Equations in R”*, Nessy Tania, Erica Graham, Qubes.
5. *“Preparing to Scale New Heights: Privatisation of India’s commercial Space Sector”*, Retrieved on 25*th* May 2021 from PWC, pwc.in
6. *“Lagrange Point”*, June 23, 2020, Retrieved on May 30, 2021 from NASA’s Website, solarsys- [tem.nasa.gov](https://solarsystem.nasa.gov/resources/754/what-is-a-lagrange-point/).in
7. GNU General Public License version-3, Open Source Initiative, [OpenSourceInitiative.org](https://opensource.org/licenses/gpl-3.0.html)
8. *“Mechanics of Orbit Using Python(MOPy)”*, [GitHub Repository.](https://github.com/iamlrk/MOPy)
9. *Python Documentation*[, python.or](https://docs.python.org/)g, v3.9.5
10. PyQt5 Documentation, [riverbankcomputing.com,](https://www.riverbankcomputing.com/static/Docs/PyQt5/) v5.15.4
11. *PySide2 Documentation*[, srinikom.github.io,](https://srinikom.github.io/pyside-docs/pysideapi2.html) v5.15.2
12. *Qt Designer Documentation*, doc.qt.io, v5.13
13. *NumPy Documentation*, NumPy.org, v1.20.3
14. *SciPy Documentation*, SciPy.org, v1.6.3
15. *Panda3D Documentation*, Panda3D.org, v1.10.9
16. *MatPlotLib Documentation*, matplotlib.org, May 08, 2021, v3.4.2
17. Wanderson M. Pimenta, GitHub Repository

28