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.
17030141AE007
lramkiranBTECH17@ced.alliance.edu.in

Manjunath
17030141AE009
manjunathBTECH17@ced.alliance.edu.in

Monisha Patel A.
17030141AE013
pamonishaBTECH17@ced.alliance.edu.in

Thoshitha R. Kumar
17030141AE027
kuthoshithaBTECH17@ced.alliance.edu.in

Under the guidance of

Prof. Gisa G.S.
Assistant Professor
Department of Aerospace Engineering,
Alliance College of Engineering and Design,
Alliance University, Bengaluru.

Prof. Yadu Krishnan
Assistant Professor
Department of Aerospace Engineering,
Alliance College of Engineering and Design,
Alliance University, Bengaluru.

Department of Aerospace Engineering
Alliance College of Engineering and Design
Alliance University, Bengaluru - 562106

Batch - 2017-21
Year - 2021

ALLIANCE UNIVERSITY

Private University Estd. in Karnataka State by Act No . 34 of year 2010
Recognized by the University Grants Commission (UGC), New Delhi

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

Internal Guide
Department of Aerospace Engineering
ACED, Alliance University
Bengaluru

Prof. Yadu Krishnan

Internal Guide
Department of Aerospace Engineering
ACED, Alliance University
Bengaluru

Prof. Velmurugarajan K.

Head of the Department
Department of Aerospace Engineering
ACED, Alliance University
Bengaluru

Dr. Reeba Korah

Interim Dean
Department of Aerospace Engineering
ACED, Alliance University
Bengaluru

External Viva

	Name of Examiners	Signature with date
1.		
2.		

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
17030141AE007

Manjunath
17030141AE009

Monisha Patel A.
17030141AE012

Thoshitha R. Kumar
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 encouragement without which this project would not have been possible.

Contents

List of Figures

List of Tables

1 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.

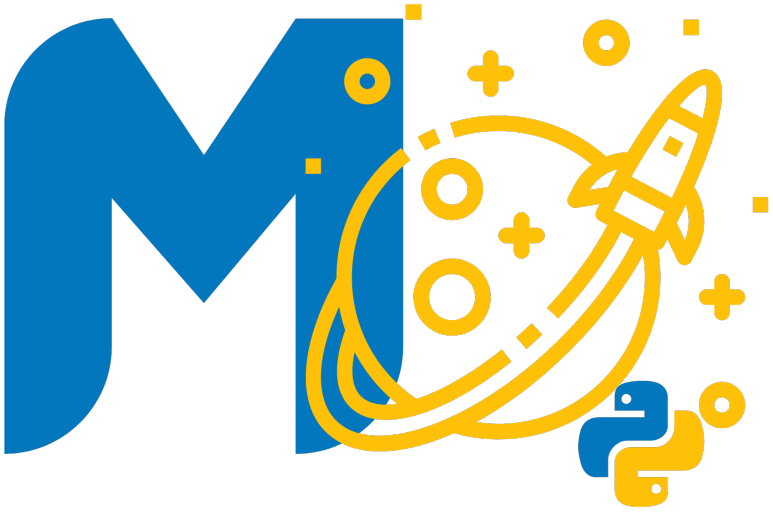

Figure 1: MOPy

1.1 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

1.2 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)

1.3 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.

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 easily 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.

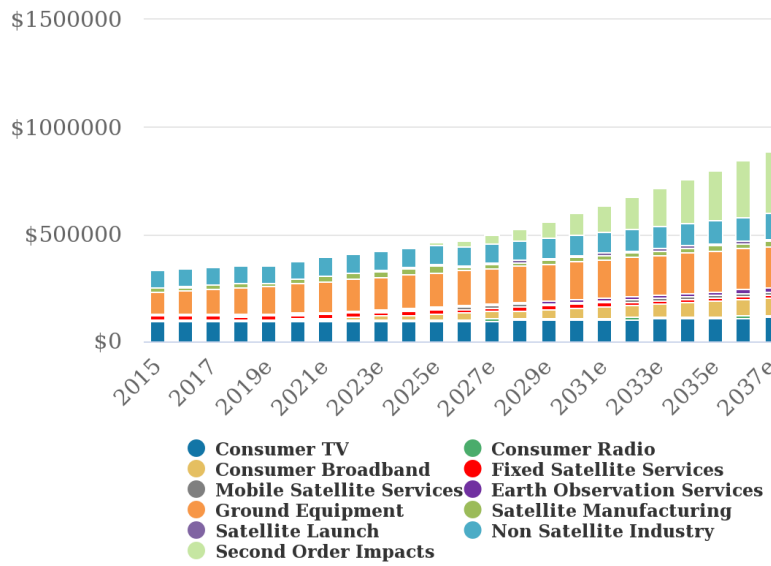

Figure 2: Global Market Trend according to Morgan Stanley.

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.

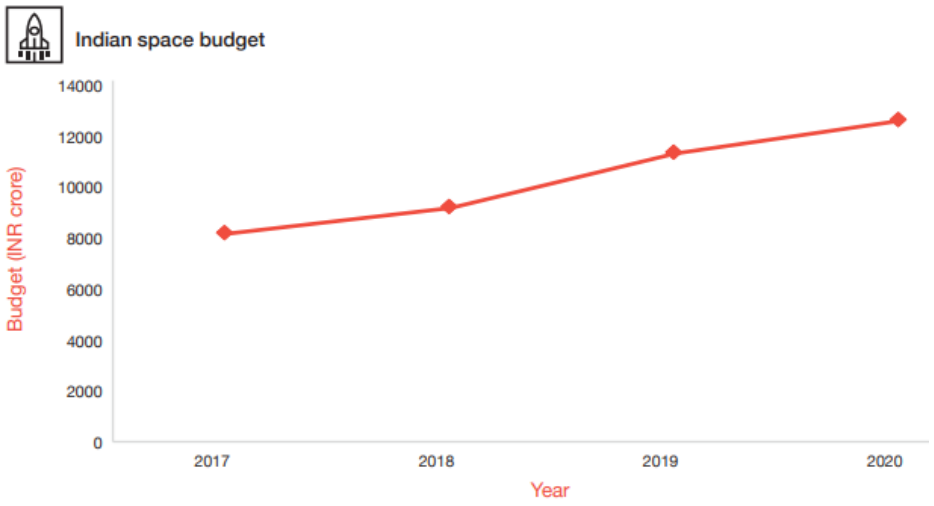

Figure 3: Indian Space Budget over the years.

1.4 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.

1.5 Front End Development

1.5.1 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.

1.5.2 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.

Figure 4: Home Page of MOPy

2 Details of the Features

2.1 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

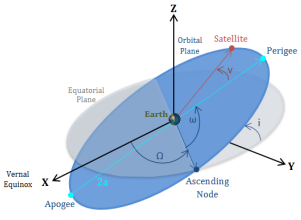

Figure 5: Classical Orbital Elements

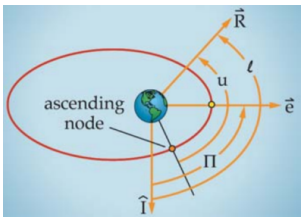

Figure 6: Alternate Orbital Elements

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.

$$a = \frac{r_a + r_p}{2} \tag{1}$$

$$a = \frac{h^2/\mu}{1 - e^2} \tag{2}$$

$where,$

$r_a = Radius\ of\ Apogee(F_1B)$

$r_p = Radius\ of\ Perigee(F_1A)$

$\mu = Standard\ Gravitational\ Parameter,$

2. **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.

$$e = \frac{c}{a} \tag{3}$$

$$e = \frac{1}{\mu}\left[\left(\frac{v^2 - \mu}{r}\right) \times \overrightarrow{r} - (\overrightarrow{r}.\overrightarrow{v}) \times \overrightarrow{v}\right] \tag{4}$$

$where,$

$r_a\ \&\ r_p = Radius\ of\ Apogee\ \&\ Perigee\ respectively,$

$\mu = Standard\ Gravitational\ Parameter,$

$\overrightarrow{v}\ \&\ v = Velocity\ vector\ \&\ Magnitude\ of\ Velocity\ Vector\ at\ r,$

$\overrightarrow{r}\ \&\ r = Radius\ Vector\ \&\ Magnitude\ of\ Radius\ Vector.$

3. **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 $\hat{K}$ unit vector to the specific angular momentum vector $\vec{h}$.

$$i = \cos^{-1}\left(\frac{\overrightarrow{h}.\overrightarrow{K}}{|h|}\right) \tag{5}$$

$where,$

$\overrightarrow{h} = Specific\ Angular\ Momentum,$

$\overrightarrow{K} = Unit\ Vector\ in\ Z\ direction,$

$|h| = Magnitude\ of\ Specific\ Angular\ Momentum.$

4. **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.

$$\Omega = \cos^{-1}\left(\frac{\overrightarrow{I}.\overrightarrow{n}}{|n|}\right) \tag{6}$$

$where,$

$\overrightarrow{n} = Nodal\ vector$

$It's\ the\ vector\ that\ joins\ the\ ascending\ node\ and\ the\ descending\ node.$

5. **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.

$$\omega = \cos^{-1}\left(\frac{\overrightarrow{n}.\overrightarrow{e}}{|n||e|}\right) \tag{7}$$

$$where,$$
$$\overrightarrow{e} = EccentricityVector.$$

6. **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.

$$\nu = \cos^{-1}\left(\frac{\overrightarrow{e}.\overrightarrow{r}}{|e||r|}\right) \tag{8}$$

$$where,$$
$$\overrightarrow{r} = RadiusVector$$

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 0^0 to 360^0. In the figure(6) Longitude of perigee is represented as "Π".

$$\Pi = \cos^{-1}\left(\frac{\overrightarrow{I}.\overrightarrow{e}}{|e|}\right) \tag{9}$$

2. **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 0^0 or 180^0. It's lies between 0^0 to 360^0. 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".

$$l = \cos^{-1}\left(\frac{\overrightarrow{I}.\overrightarrow{r}}{|r|}\right) \tag{10}$$

3. **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 0^0 to 360^0.
In the figure(6) Argument of latitude is represented as "u".

$$u = \cos^{-1}\left(\frac{\overrightarrow{n}.\overrightarrow{r}}{|n||r|}\right) \tag{11}$$

2.2 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, $r_1,\ v_1,\ \gamma_1$ 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 - $a, e/r_a, r_p/r_1, v_1, \gamma_1$
	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.

$$r = \frac{h^2/\mu}{1 - e\cos(\theta)}$$

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

4. **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:

$$Entertheaccelerationvector$$

2.3 Euler Angle

The inputs for this are as in table(4). 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

2.4 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 F_g drops off rapidly with distance r from the center of attraction. If

F_{g0} 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:

$$r_{SOI} = R\left(\frac{MiB\_mass}{MaB\_mass}\right)^{2/5}$$

2.5 Orbital Transfer

This feature simulates the orbital transfer using various numerical 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

2.6 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

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:

$$JDN = \frac{1461\left(Y + 4800 + \frac{M-14}{12}\right)}{4} + \frac{367\left(M - 2 - 12\frac{M-14}{12}\right)}{12} - \frac{3\left(\frac{Y + 4900 + \frac{M-14}{12}}{100}\right))}{4} + D - 32075$$

To convert from Julian Calendar:

$$JDN = 367 \times Y - \frac{7\left(Y + 5001 + \frac{M-9}{7}\right)}{4} + \frac{275M}{9} + D + 1729777$$

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

$$JD = JDN + \frac{HH - 12}{24} + \frac{MM}{1440} + \frac{SS}{86400}$$

2.7 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	$a, e/r_a, r_p/r_1,\ v_1,\ \gamma_1$/ Orbital Elements/State and velocity vectors
Outputs	$\mu,\ , h\ , \epsilon$, 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:

Circular and Elliptical	Parabolic	Hyperbolic
$\mu = G \times M_{Major}$		
$r_p = a \times (1-e)$	$l = 2 \times r_p$	$\theta_\infty = \cos^{-1}(\frac{1}{e}) = \beta$
$r_a = a \times (1+e)$		$\delta = 2\sin(\frac{1}{e})$
$\epsilon = \frac{-\mu}{2a}$	0	$h = \sqrt{\mu a(e^2-1)}$
$h = \sqrt{r_p \times (1+e)\mu}$	$h = \sqrt{2\mu r_p}$	$b = ae\sin(\theta_\infty) = \Delta$
$F_g = \frac{GMm}{r^2}$		
$V = \sqrt{\frac{2\mu}{r} - \frac{\mu}{a}}$	$V_r = \sqrt{\frac{2\mu}{r}}$	$V_\infty = e\mu\sin(\theta_\infty)/h$
$T = \sqrt{\frac{4\pi^2}{\mu} \times a^3}$		$\epsilon = \frac{\mu}{2a}$
$n = \frac{2\pi}{T}$		$l = \frac{-b^2}{a}$
$l = a \times (1-e^2)$		$r_p, r_a = \frac{h^2/\mu}{1 \pm e}$

Table 9: Formulae for Various parameters

2.8 Sensitivity Analysis

User has to select the type of calendar and the in the input section they have to select date and time and accuracy of digits and then by clicking on calculate button we will get Julian days in the output section. This takes in the required values an outputs how much of an error will that small change in the velocity or radius in the 3D environment.

Input	State Vector, Velocity Vector and two delta-v with slight difference
Output	Percentage difference caused to the final orbit parameters due to slight difference in delta-v

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.

2.9 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.

2.10 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

3 Database Management System

3.1 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.

4 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

5 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.

A Code for the Features

A.1 Calculation Of Orbital Elements

```
from numpy.linalg import norm
from numpy import dot, pi, cross, multiply as multi
from math import acos

class Calculate:
    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 muvalue(cls, major_body):
        major_body_list = pandas.read_csv("Major_and_Minor_Bodies.csv")
        major_bodies = list(major_body_list.Major_Body)
        major_bodies_mass = list(major_body_list.Mass)
        major_bodies_radius = list(major_body_list.Radius)
        sel_major_body = major_bodies.index(major_body)
        major_body_mass = major_bodies_mass[sel_major_body]
        major_body_radius = major_bodies_radius[sel_major_body]
        mu = cls.G * major_body_mass
        return [mu, major_body_radius]

    @classmethod
    def correct_ohm(cls, ohm, n_vec):
        if n_vec[0] > 0 and n_vec[1] > 0:
            print("This is a Prograde Elliptical Orbit and is in first Quadrant.")
            if ohm >90:
                ohm -= 360
        elif n_vec[0] < 0 and n_vec[1] > 0:
            print("This is a Retrograde Elliptical Orbit and is in second Quadrant."
    )
            if ohm > 180:
                ohm -= 360
        elif n_vec[0] < 0 and n_vec[1] < 0:
            print("This is a Retrograde Elliptical Orbit and is in Third Quadrant.")
            if ohm < 180:
                ohm -= 360
        elif n_vec[0] > 0 and n_vec[1] < 0:
            print("This is a Prograde Elliptical Orbit and is in fourth Quadrant.")
            if ohm < 90:
                ohm -= 360
        return ohm

    @classmethod
    def other_var(cls, pos_vec, vel_vec):
        h_vec = cross(pos_vec, vel_vec)
        n_vec = cross(cls.K,h_vec)
        return [h_vec, n_vec]

    @classmethod
    def OE(cls, pos_vec, vel_vec, mu):
```

```
        [h_vec, n_vec] = Calculate.other_var(pos_vec, vel_vec)
        e_vec = (multi((norm(vel_vec)*norm(vel_vec)-(mu/norm(pos_vec))),pos_vec)-
    multi(dot(pos_vec,vel_vec),vel_vec))/(mu)
        call.ecc_o.setText(str(norm(e_vec)))
        inc = (acos((dot(h_vec, cls.K))/norm(h_vec))) * 180/pi
        call.inc_o.setText(str(inc))
        sma = 1/((2/norm(pos_vec))-((norm(vel_vec)*norm(vel_vec))/mu))
        call.sma_o.setText(str(sma))
        return [sma, inc, e_vec]

    @classmethod
    def ACOE(cls, pos_vec, vel_vec, e_vec, inc):
        [h_vec, n_vec] = Calculate.other_var(pos_vec, vel_vec)
        if inc != 0 or 180 and norm(e_vec) > 0: #Nothing is Zero/180
            ohm = (acos((dot(cls.I,n_vec))/norm(n_vec)))
            ohm = Calculate.correct_ohm(ohm, n_vec)
            call.ohm_o.setText(str(ohm))
            nu = (acos((dot(e_vec,pos_vec))/(norm(e_vec)*norm(pos_vec))))
            call.nu_u_o.setText(str(nu))
            omega = (acos((dot(n_vec,e_vec))/multi(norm(n_vec),norm(e_vec))))
            call.omega_l_o.setText(str(omega))
            return [ohm, omega, nu]
        elif inc == 0 or 180: #Inclination is Zero
            if norm(e_vec) > 0: #Elliptical Orbit
                nu = (acos((dot(e_vec,pos_vec))/(norm(e_vec)*norm(pos_vec))))
                call.nu_u_o.setText(str(nu))
                Long_of_peri_pi = acos(dot(cls.I,e_vec)/(norm(cls.I)*norm(e_vec)))
                return [Long_of_peri_pi, nu]
            elif norm(e_vec) == 0: #Circular Orbit
                Tr_long_l = acos(dot(cls.I,pos_vec)/(norm(pos_vec)*norm(cls.I)))
                return [Tr_long_l]
        elif norm(e_vec) == 0: #Circular orbit with inclination non-zero/pi
            Arg_of_lattitude_u = acos(dot(n_vec,pos_vec)/(norm(n_vec)*norm(pos_vec))
    )
            return [Arg_of_lattitude_u]

    @classmethod
    def possibility(cls, major_body, pos_vec, vel_vec):
        [mu, major_body_radius] = Calculate.muvalue(major_body)
        pos = norm(pos_vec)
        vel = norm(vel_vec)
        sma = 1/((2/pos)-(vel*vel/mu))
        e_vec = (multi((norm(vel_vec)*norm(vel_vec)-(mu/norm(pos_vec))),pos_vec)-
    multi(dot(pos_vec,vel_vec),vel_vec))/(mu)
        r_peri = sma*(1-norm(e_vec))
        if r_peri <= major_body_radius:
            print("The Orbit is not possible as radius of perigee is less than
    radius of the major Body")
            return False
        return True
```

Listing 1: CoOE.py

A.2 Euler Angles

```
from math import *
from numpy import *

class EA():

    def RxRyRz(pitch_theta, roll_phi, yaw_si):
        Rx = matrix([[ 1, 0 , 0 ],
                     [ 0, cos(roll_phi),sin(roll_phi)],
                     [ 0,-sin(roll_phi), cos(roll_phi)]])

        Ry = matrix([[ cos(pitch_theta), 0,-sin(pitch_theta)],
                     [ 0 ,1, 0],
                     [sin(pitch_theta), 0, cos(pitch_theta)]])

        Rz = matrix([[ cos(yaw_si), sin(yaw_si), 0 ],
                     [-sin(yaw_si), cos(yaw_si) , 0 ],
                     [ 0, 0, 1]])
        return [Rx, Ry, Rz]

    def DCMtoEA(DCM, order):
        if order=='321':
            theta=-asin(DCM[0][2])
            phi=atan(DCM[1][2]/DCM[2][2])
            si=atan(DCM[0][1]/DCM[0][0])
        elif order=='123':
            theta=asin(DCM[2][0])
            phi=atan(DCM[2][1]/DCM[2][2])
            si=atan(DCM[1][0]/DCM[0][0])
        elif order=='132':
            theta=atan(DCM[2][0]/DCM[0][0])
            phi=atan(DCM[1][2]/DCM[1][1])
            si=-asin(DCM[1][0])
        elif order=='312':
            theta=atan(DCM[0][2]/DCM[2][2])
            phi=asin(DCM[1][2])
            si=atan(DCM[1][0]/DCM[1][1])
        elif order=='213':
            theta=atan(DCM[2][0]/DCM[2][2])
            phi=-asin(DCM[2][1])
            si=atan(DCM[0][1]/DCM[1][1])
        elif order=='231':
            theta=atan(DCM[0][2]/DCM[0][0])
            phi=atan(DCM[2][1]/DCM[1][1])
            si=asin(DCM[0][1])
        return [theta, si, phi]

    def EAtoDCM(theta, si, phi, order):
        [Rx, Ry, Rz] = EA.RxRyRz(theta, si, phi)
        if order=='321':
            RxRy = dot(Rx,Ry)
            return (dot(RxRy,Rz))
        elif order == '123':
            x = Rx(phi*pi/180)
            y = Ry(theta*pi/180)
            z = Rz(si*pi/180)
```

```
            zy = dot(z,y)
            return (dot(zy,x))
        elif order == '213':
            x = Rx(phi*pi/180)
            y = Ry(theta*pi/180)
            z = Rz(si*pi/180)
            zx = dot(z,x)
            return (dot(zx,y))
        elif order == '312':
            x = Rx(phi*pi/180)
            y = Ry(theta*pi/180)
            z = Rz(si*pi/180)
            yx = dot(y,x)
            return (dot(yx,z))
        elif order == '132':
            x = Rx(phi*pi/180)
            y = Ry(theta*pi/180)
            z = Rz(si*pi/180)
            yz = dot(y,z)
            return (dot(yz,x))
        elif order == '231':
            x = Rx(phi*pi/180)
            y = Ry(theta*pi/180)
            z = Rz(si*pi/180)
            xz = dot(x,z)
            return (dot(xz,y))
        else:
            return ('No transformation matrix')

order = '321' #input("Enter the order:")
DCM = [[0.6405, 0.75319, -0.15038],[0.76736, -0.63531, 0.086824],[-0.030154,
    -0.17101, -0.98481]]
print(EA.DCMtoEA(DCM,order))

print(EA.EAtoDCM(8.6489  * 180/pi, 49.619 * 180/pi, -5.039 * 180/pi, order))
```

Listing 2: EA.py

A.3 Sphere of Influence

```
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 = 1.989e30
    rSOI = (r_maj_to_min*(MiB_mass/MaB_mass)**(2/5))
    return [rSOI, rSOI/MiB_radius]
```

Listing 3: SoI

A.4 Calculation of Julian Day

```
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)]
```

Listing 4: CJSD.py

A.5 Calculation of Various Parameters of the Orbit

```
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


```

```
# all the units should be km
class CalculateCircularElliptical:
    I = [1, 0, 0]
    J = [0, 1, 0]
    K = [0, 0, 1]
    G = 6.67e-20 #units are in km3 kg-1 s-2
    def __init__(self, major_body):

        self.major_body = major_body
        return

    @classmethod
    def semiecc(cls, sma, mag_e, major_body):
        r_per = sma * (1 - mag_e)
        r_apo = sma * (1 + mag_e)
        [mean_motion, T_period, mag_h, sme, slr] = CalculateCircularElliptical.
    orb_const(r_per, sma, mag_e, major_body)
        return [r_per, r_apo, mean_motion, T_period, mag_h, sme, slr]

    @classmethod
    def perapo(cls, r_per, r_apo, major_body):
        sma = (r_per + r_apo)/2
        mag_e = (r_apo - r_per)/(r_apo + r_per)
        [mean_motion, T_period, mag_h, sme, slr] = CalculateCircularElliptical.
    orb_const(r_per, sma, mag_e, major_body)
        return [mag_e, sma, mean_motion, T_period, mag_h, sme, slr]

    @classmethod
    def orb_const(cls, r_per, sma, mag_e, major_body):
        major_body_mass = 5.972e24 #self.major_body * 1
        mu = cls.G * major_body_mass
        slr = sma*(1-mag_e**2)
        mag_h = (r_per*(1+e)*mu)**0.5
        T_period = sqrt(((4*pi**2)/mu)*sma**3)
        mean_motion = T_period/(2*pi)
        sme = -mu/(2* sma)
        return [mean_motion, T_period, mag_h, sme, slr]

    @classmethod
    def time_since_periapsis(cls, mag_e, theta, T_period):
        E_anomly = 2 * atan((sqrt((1 - mag_e)/(1 + mag_e)) * tan(theta/2)))
        M_anomly = E_anomly - mag_e * sin(E_anomly)
        t_since_perigee = T_period * M_anomly/(2*pi)
        return t_since_perigee

    @classmethod
    def velocity_at_any_point(cls, sma, r, major_body):
        major_body_mass = major_body * 1
        mu = cls.G * major_body_mass
        v_point = sqrt(((2 * mu)/r) - (mu/sma))
        return v_point

class CalculateParabola():
    I = [1, 0, 0]
    J = [0, 1, 0]
    K = [0, 0, 1]
    G = 6.67e-20 #units are in km3 kg-1 s-2
```

```
71
72      @classmethod
73      def const_values(cls, r_per, major_body):
74          major_body_mass = major_body * 1
75          mu = major_body_mass * cls.G
76          slr = 2 * r_per
77          v_per = sqrt((2*mu)/r_per)
78          mag_h = sqrt(mu*r_per * 2)
79          return [slr, v_per, mag_h]
80
81      @classmethod
82      def velocity_at_any_point(cls, r, major_body):
83          major_body_mass = major_body * 1
84          mu = major_body_mass * cls.G
85          v = sqrt((2*mu)/r)
86          return v
87
88  class CalculateHyperBola():
89      I = [1, 0, 0]
90      J = [0, 1, 0]
91      K = [0, 0, 1]
92      G = 6.67e-20 #units are in km3 kg-1 s-2
93
94      @classmethod
95      def const_values(cls,major_body,mag_e, sma):
96          mu = cls.G * major_body
97          theta_inf = acos(1/mag_e) #angle between asympotes
98          beta = theta_inf
99          delta = 2 * sin(1/mag_e) #turn angle
100         mag_h = sqrt(sma * mu * (mag_e**2-1)) #specific angular momentum
101         b = sma * mag_e * sin(theta_inf) #semi minor axis
102         Delta = b #aiming radius
103         v_inf = mu * mag_e*sin(theta_inf)/mag_h #velocity at r = infinity
104         sme = mu/(2 * sma) #specific mechanical energy
105         slr = -b**2/sma #semi latus rectum
106         return [theta_inf, beta, delta, mag_h, b, Delta, v_inf, sme, slr]
107
108     @classmethod
109     def semiecc(cls,major_body,sma,mag_e):
110         mu = cls.G * major_body
111         [theta_inf, beta, delta, mag_h, b, Delta, v_inf, sme, slr] =
    CalculateHyperBola.const_values(major_body, sma, mag_e)
112         r_apo = (mag_h**2/mu)/(1 - mag_e)
113         r_per = (mag_h**2/mu)/(1 + mag_e)
114         return [theta_inf, beta, delta, mag_h, b, Delta, v_inf, sme, slr, r_apo,
    r_per]
```

Listing 5: VPCO

A.6 Sensitivity Analysis

```
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]
```

Listing 6: SA.py

A.7 Lagrangian Points

```
from math import pi
from numpy import arange
import matplotlib.pyplot as plt
from scipy.optimize import root_scalar

G = 6.67408e-11
m1 = 5.972e24
m2 = 0.07346e24
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=[2e8, 3.5e8])
print(L1.root+x1)
L2 = root_scalar(x_eqn, bracket=[3.9e8, 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()
```

Listing 7: LP.py

A.8 CRUD File Operations

```
# 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()
```

Listing 8: db.py

A Database:

	Mercury	Venus	Earth	Moon	Mars	Jupiter	Saturn	Uranus	Neptune	Pluto
Mass($10^{24}kg$)	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(10^6km)	57.9	108.2	149.6	0.384	227.9	778.6	1433.5	2872.5	4495.1	5906.4
Perihelion(10^6km)	46.0	107.5	147.1	0.363	206.6	740.5	1352.6	2741.3	4444.5	4436.8
Apohelion(10^6km)	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 Final 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] GNU General Public License version-3, Open Source Initiative, OpenSourceInitiative.org

[7] *"Mechanics of Orbit Using Python(MOPy)"*, GitHub Repository.

[8] *Python Documentation*, python.org, v3.9.5

[9] PyQt5 Documentation, riverbankcomputing.com, v5.15.4

[10] *PySide2 Documentation*, srinikom.github.io, v5.15.2

[11] *Qt Designer Documentation*, doc.qt.io, v5.13

[12] *NumPy Documentation*, NumPy.org, v1.20.3

[13] *SciPy Documentation*, SciPy.org, v1.6.3

[14] *Panda3D Documentation*, Panda3D.org, v1.10.9

[15] *MatPlotLib Documentation*, matplotlib.org, May 08, 2021, v3.4.2