STUDYING THE PLANETARY MOTION USING PYTHON

Abstract

In the present work, we have studied the motion of planet and the dwarf planet and estimated the value of Mass of the Sun and value of Gravitational Constant (in case of dwarf planet). We also have studied the non-relativistic contribution of Mercury’s perihelion precession. For the purpose of the same, we have considered the paper by Price and Rush, the freely available NASA data from NASA website as well as freely available data in Wikipedia website. For computational modeling purpose, we have used the Python programming language.

Keywords: Python, NASA, Programming language

Authors

Jayashree Kundu
Assistant Teacher
Shyam Sukhi Balika Shiksha Niketan
West Bengal, India

Dr. Rakesh Kumar Mandal
Associate Professor
Department of Computer Science & Technology
University of North Bengal
P.O North Bengal University
West Bengal, India

Dr. Tamal Sarkar
Scientific Research Officer-II
High Energy & Cosmic Ray Research Centre
University of North Bengal
P.O North Bengal University
West Bengal, India
ts.phys.edu2020@gmail.com

I. METHODOLOGY

We have taken the recent parameters /data of the planets and dwarf planets from the NASA website for all the Planets and Dwarf planets, and to study the precession of Mercury, we consider the paper by Price and Rush (Price & Rush, 1979) for calculating the non-relativistic contribution of Mercury's perihelion precession. In our work, we have created a computational model which takes the data in Excel format as input and produces an output in Excel format after processing the data about the required parameters. We have used Python open-source software for writing the computational model. The organization of the chapter is as follows: First, we have discussed our methodology and why Python is the preferred software. In the second part and third of the given chapter, we discuss how to find out Sun's mass and Gravitational Constant (i.e., Universal Gravitation) using the planetary motion data of planets and dwarf planets. The chapter's fourth part deals with how to code the computational model to study the non-relativistic contribution of Mercury's perihelion precession.

II. PYTHON AND ITS PREFERENCE FOR ASTRONOMY

Python has become one of the most popular programming languages in astronomy due to its versatility, ease of use, and rich ecosystem of scientific libraries. Several factors contribute to Python's preference for astronomy:

1. Python offers many scientific libraries essential for astronomical research (Helfrich, 2019). Some prominent ones include NumPy for numerical computations, SciPy for scientific computing, matplotlib for data visualization, and Astropy for astronomical data analysis. Astropy, in particular, provides core functionality and data structure for handling astronomical data, including units, coordinates, and time.
2. Python's clear and readable syntax makes learning easy, even for those with little programming experience. Astronomers can quickly adapt to Python and start writing productive code without spending much time on the language's intricacies.
3. Python has a vast and active community of astronomers and developers contributing to various projects and libraries. This support network ensures astronomers can seek help, share knowledge, and collaborate on common challenges.
4. Python easily integrates with other programming languages like C, C++, and Fortran, which are often used for computationally intensive tasks in astronomy. This allows astronomers to harness the power of optimized code where needed while still utilizing Python for higher-level jobs.
5. Python's Matplotlib and other libraries enable astronomers to create publication-quality plots and visualize complex data effectively. Visualization is essential for understanding data and presenting research findings.
6. Python's data manipulation libraries, such as Pandas, make processing and analyzing large astronomical datasets straightforward.
7. With the rise of machine learning and artificial intelligence in various scientific fields, Python's extensive libraries (e.g., TensorFlow, PyTorch, sci-kit-learn) also make it a viable choice for implementing these techniques in astronomy.
8. Many astronomy-focused Python packages are developed in collaboration with researchers and are tailored to specific astronomical needs, making them reliable and relevant to the field.

9. Python is an open-source language which aligns well with the principles of openness and transparency in scientific research. This allows researchers to inspect, modify, and share the code freely.

   Due to these advantages, Python has gained widespread acceptance in the astronomy community, from data analysis and simulations to data visualization and scientific computing. Its flexibility, combined with the availability of astronomy-specific libraries, makes it a powerful tool for astronomers to conduct their research efficiently.

III. EIGHT PLANETS OF SOLAR SYSTEM AND DETERMINATION OF MASS OF THE SUN

Now, to have a better understanding of how to extract valuable information from a given dataset, we have created one Excel file having planet name, it's mass, orbital distance, time period etc.

Table 1: Eight Planets of Solar System

Sl_no.	Planet_nm	Planet_mass	Orbital_Radius	Spin_wrt_Axis (in Earth day)	Time_Period
1	Mercury	0.33011	57.9	58.646	87.969
2	Venus	4.8675	108.2	-243.025	224.701
3	Earth	1	149.6	0.997	365.24
4	Mars	0.64171	228	1.026	686.565
5	Jupiter	1898.19	778.5	0.414	4328.9
6	Saturn	568.34	1432	0.426	10752.9
7	Uranus	86.813	2867	0.718	30667.3
8	Neptune	102.413	4515	0.671	60152

Source: NASA and Wikipedia

1. **Reading a File:** By using the following Python code, we will be able to read the previously created Excel file of Astronomical data by converting it into DataFrame.

```
import pandas as pd
df=pd.read_excel("Eightplanets_data.xlsx")
```

   The above code creates a DataFrame in the Python environment.

2. **Calculating Orbital Speed and Mass of Sun:** We all know that orbits of our planets are Elliptical in nature, but for mathematical simplicity, lets take that orbit is circular. Thus, we have got the followings.

$$\boldsymbol{Orbital\ Speed(v) = \frac{2\pi r}{T}} \qquad \text{....(1)}$$

And we have got this Eq.(2) by equating the Gravitational Force and Centripetal Force.

$$\boldsymbol{Mass\ of\ Sun(M_s) = \frac{4\pi^2 r^3}{GT^2}} \qquad \textbf{....}(2)$$

Where r= Orbital radius of planets, and T= Time period of planets, G is the gravitational constant.

Now, after getting the DataFrame(i.e.,df), we will deduce orbital speed of every planet with the help of the above Eq.(1),and mass of the sun with the the help of the following Eq.(2), by the following Python coding:-

```
import numpy as np
import pandas as pd
a_o_r=np.array(df.orbital_radius) #Creates an array of orbital radius from DataFrame.
a_t_p=np.array(df.time_period) #Creates an array of time period from DataFrame.
length=len(a_o_r) #Making the Loop range.
velo_list=list() #Creating an empty List to hold orbital velocity.
velo_2_r_list=list() #Creating an empty List to hold v2r values.
m_sun_list=list() #Creating an empty List to hold different values for Sun’s mass.
for i in range(length):
   temp_speed=((2*np.pi)*a_o_r[i])/(a_t_p[i]*24*60*60)
   speed=temp_speed*(10**6)
   sp_2_r=((temp_speed**2)*a_o_r[i])*(10**18)
   m_sun=(4*(np.pi**2)*((a_o_r[i]*(10**9))**3)/
                ((6.6743*(10**-11))*((a_t_p[i]*24*60*60)**2)))
   velo_list.append(speed) #Velocity of each planet is being appended in the existing list.
   velo_2_r_list.append(sp_2_r)
   m_sun_list.append(m_sun)
df['velocity']=velo_list #List of velocity is being appended to existing DataFrame.
df['v^2*r']=velo_2_r_list
df['sun_mass']=m_sun_list
df #Gives output of modified dataframe.
```

If we notice the above DataFrame(i.e.,df), then we can have an interesting fact that the value in (v^2r) column are almost equal which exhibits the constant property and can be visually shown by Python as follows.

```
From matplotlib import pyplot as plt
plt.title("  Property of V^2*R")
plt.xlabel("Name Of Planets")
plt.ylabel("V^2*R value")
plt.scatter(df.planet_nm,df['v^2*r'])
plt.show()
```

We can also notice that the Mass of Sun is slightly different for each planet. Thus, we took Average of that mass, although we usually use Earth’s orbital radius and time period to calculate the mass of Sun.

3. **Calculating the Mean, Median and Standard Deviation:** Through the following coding, we are able to calculate mean, median and standard deviation of (v^2r) using Numpy.

```
mean_v2r=np.mean(velo_2_r_list)
median_v2r=np.median(velo_2_r_list)
std_v2r=np.std(velo_2_r_list)
```

   And in the same way, we calculate the mean median and standard deviation of Mass of sun which exhibit different value for different planet.

4. **Writing to Excel:** In order to insert all those calculations, we will have to write the Python code as follows:

```
with pd.ExcelWriter('C:/New_ Eightplanets_data.xlsx') as writer:
    df.to_excel(writer,sheet_name="DataBase_with_speed",index=False)
    Mean_value_list=[{'Mean of V^2r':np.mean(velo_2_r_list),'Mean Mass(Sun)':
                    [np.mean(m_sun_list),"KG"],'Median Mass(Sun)':
                    [np.median(m_sun_list),"KG"],'Standard Deviation of
                    mass (Sun)':
                    [np.std(m_sun_list),"KG"]}]
    df1=pd.DataFrame(Mean_value_list)
    df1.to_excel(writer,sheet_name="Mean_value_Data",index=False)
```

   The above code creates one new Excel file named New_Eightplanets_data in the specified path with two sheets named as DataBase_with_speed and Mean_value_Data having index as False just because of discarding the extra column that DataFrame always generates in extreme left.

5. **Checking of Excel Creation:** We can check our newly created Excel file in two ways. One is, we can open the file directly from the system drive by clicking on it. And another way to check the content is to have an access over that Excel file from the Python

6. **Environment through the following coding:-**

```
import pandas as pd
df=pd.read_excel("New_Eightplanets_data.xlsx")
df.tail(8)
```

   The last line of the above code displays the last eight records of excel file. Although in this case, we have only eight rows. Instead of writing df.tail(8), we may code as df.head() which displays first five records by default.

IV. NINE DWARF PLANETS OF SOLAR SYSTEM AND DETERMINATION OF THE GRAVITATIONAL CONSTANT

In our Solar system there are several celestial bodies that orbits Sun, but not in that way that our known eight planets do. That means to be in the category of planet, any celestial object should obey all the three rules which tells that if it, is of spherical shape (which ensures that it has its own gravity), revolves around the sun and doesn't create any disturbance for others to move around sun during each trip in its orbit or makes clear its own orbit as well, it is a planet. But there are some celestial objects in our Solar system, which don't obey the third rule although they obey the first two rules. Thus, these are no longer in the category of planets, and hence are called Dwarf Planet. Here, we have aimed to explore their property of obeying *Newton's* Law of Gravitation using Python.

Table 2: Dwarf Planets Dataset

Sl_no.	dwarf_nm	Mean_distance (from Sun)	Orbital_period
1	Ceres	2.77	4.6
2	Orcus	39.4	247.3
3	Pluto	39.48	247.9
4	Haumea	43.22	248.1
5	Quaoar	43.69	288.8
6	Makemake	45.56	307.5
7	Gonggong	67.38	553.1
8	Eris	67.38	558
9	Sedna	506.8	11400

Source: NASA & Wikipedia

Now, to have a better understanding of how to extract valuable information from a given dataset, we have created one Excel file with the sheet name as Dwarf_DataBase having dwarf planet's name, orbital distance, time period etc.

1. **Reading a File:** By using the following Python code, we will be able to read the previously created Excel file of Astronomical data by converting it into DataFrame.

```
import pandas as pd
df=pd.read_excel("Dwarfplanets_data.xlsx")
```

   The above code creates a DataFrame in the Python environment.

2. **Calculating Orbital Speed and Gravitational Constant:** We all know that orbits of our dwarf planets are Elliptical in nature, but for mathematical simplicity, lets take that orbit is circular. Thus, we have got the followings.

$$\boldsymbol{Orbital\ Speed(v) = \frac{2\pi r}{T}} \qquad \textbf{....}(1)$$

   And we have got this Eq.(3) by equating the Gravitational Force and Centripetal Force.

$$\boldsymbol{Gravitational\ Constant(G) = \frac{4\pi^2 r^3}{M_s T^2}} \qquad .....(3)$$

Where r= Mean distance from Sun(in AU), and T= Orbital time period of dwarf planet(in Earth years), M_s =Mass of Sun(in Kg).

Now, after getting the DataFrame(i.e.,df), we will deduce orbital speed of every dwarf planet with the help of the above Eq.(1),and the value of Gravitational Constant(G) with the help of the following Eq.(3), by the following Python coding:-

```
from matplotlib import pyplot as plt
plt import numpy as np
import pandas as pd
a_o_r=np.array(df['Mean_distance(from Sun)']) #Creates an array of orbital radius
                                                    from DataFrame.
a_t_p=np.array(df.Orbital_period) #Creates an array of time period from DataFrame.
length=len(a_o_r) #Making the Loop range.
velo_list=list() #Creating an empty List to hold orbital velocity.
velo_2_r_list=list()#Creating an empty List to hold v2r values.
G_value_list=list() #Creating an empty List to hold values for Universal Gravitation.
for i in range(length):
        temp_speed=((2*np.pi)*(a_o_r[i]*149.6)/(a_t_p[i]*365*24*60*60))
        speed=temp_speed*(10**6)
        sp_2_r=((temp_speed**2)*a_o_r[i])*(10**18)
        G_value=(4*(np.pi**2)*((a_o_r[i]*149.6*(10**9))**3))/
                                  (Mass_sun*((a_t_p[i]*365*24*60*60)**2))
        velo_list.append(speed) #Velocity of each dwarf planet is being appended in the
                                  existing list.
        velo_2_r_list.append(sp_2_r)
        G_value_list.append(G_value)
df['velocity']=velo_list #List of velocity is being appended to existing DataFrame.
df['v^2*r']=velo_2_r_list
df['G_value']=G_value_list
df  #Gives output of modified dataframe.
```

If we notice the above DataFrame(i.e.,df), then we can have an interesting fact that the value in (v^2r) column are almost equal which exhibits the constant property and can be visually shown by Python as follows.

```
plt.title(" Property of V^2*R")
plt.xlabel("\nName Of Dwarf-Planets")
plt.ylabel("V^2*R value")
plt.scatter(df.dwarf_nm,velo_2_r_list)
plt.show()
```

We can also notice that the Gravitational constant is almost same for each planet except Haumea. And this is clearly visible through Python coding as follows.

```
from matplotlib import pyplot as plt
plt.title(" Property of G")
plt.xlabel("\nName Of Dwarf-Planets")
plt.ylabel("G_value")
plt.scatter(df.dwarf_nm,G_value_list)
plt.show()
```

Although we took the Average of that G value, and to do this we usually use Earth's orbital radius and time period.

3. **Calculating the Mean, Median And Standard Deviation:** Through the following coding, we are able to calculate mean, median and standard deviation of G value using Numpy array.

```
Mean_G=np.mean(G_value_list)
Median_G=np.median(G_value_list)
Std_G=np.std(G_value_list)
```

And in the same way, we calculate the mean median and standard deviation of v^2r which also exhibit almost same value for different dwarf planets.

4. **Writing to Excel:** In order to insert all those calculations, we will have to write the Python code as follows:

```
with pd.ExcelWriter('C:/New_Dwarfplanets_data.xlsx') as writer:
        df.to_excel(writer,sheet_name="DwarfDataBase_with_speed",index=False)
        Mean_value_list=[{'Mean of V^2r':[np.mean(velo_2_r_list),"N*m^2/Kg"],
                          'Mean G_value':[Mean_G,"N*m^2/Kg^2"],
                          'Median G_value':[Median_G,"N*m^2/Kg^2"],
                          'Standard Deviation of G_value':Std_G}]
        df1=pd.DataFrame(Mean_value_list)
        df1.to_excel(writer,sheet_name="Mean_value_Data",index=False)
```

The above code creates one new Excel file named New_Dwarfplanets_data in the specified path with two sheets named as DwarfDataBase_with_speed and Mean_value_Data having index as False just because of discarding the extra column that DataFrame always generates in extreme left.

5. **Checking of Excel Creation:** We can check our newly created Excel file in two ways. One is ,we can open the file directly from the system drive by clicking on it. And another way to check the content is to have an access over that Excel file from the Python Environment through the following coding:-

```
import pandas as pd
df=pd.read_excel("New_Dwarfplanets_data.xlsx")
df.tail(8)
```

The last line of the above code displays the last eight records of excel file. Although in this case, we have only eight rows. Instead of writing df.tail(8), we may code as df.head() which displays first five records by default.

V. NON-RELATIVISTIC CONTRIBUTION OF MERCURY'S PERIHELION PRECESSION

Mercury, the nearest planet to the Sun, differs in several ways. One is the Precession of the Orbit. Its orbit is very unique and mysterious. Mercury takes approximately 88 days to make one trip around the Sun, and after each complete trip, its orbit gets changed a bit, and this amount of shift causes the change in the position of Mercury's perihelion which is a little bit fast than it was supposed to be. The reason is that Mercury is pulled not only by the Sun but also by the other planets of the solar system. Thus the attraction of the planets on Mercury becomes more critical when it is at the furthest distance from the Sun. As a result, the centripetal force (F_C) experienced by Mercury (due to Sun), gets weakened. And when it is closest to Sun, the F_C experienced by Mercury due to Sun is far much greater than the perturbing force, which causes Mercury to follow a new path instead of its normal one. We can understand this concept by the following explanation and coding, which shows the strengths of Python Libraries (such as Numpy, Panda, etc.).

To have a better understanding of how to extract valuable information from a given dataset, we have created one Excel file having planet name, it's mass, orbital distance, orbital time period etc.

1. **Reading a File:** By using the following Python code, we will be able to read the previously created Excel file of Astronomical data by converting it into DataFrame.

```
import pandas as pd
df=pd.read_excel("Eightplanets_data.xlsx")
df.head(8)
```

The above code creates a DataFrame (i.e.,df) in the Python environment as follows (i.e.,Table-1).

2. **Concept of Uniform Ring:** The way the small shift can be approximated easily is to assume all the planets as a uniform circular ring, centered on the Sun and the object under experiment (i.e.,Mercury) as a point object. And, as their exerted force matters a lot on Mercury(for its perihelion precession) and the planets may be in their orbit, hence it is important to calculate the Linear Mass Density(λ_i) for all the planets by the following equation.

$$\boldsymbol{Linear\ Mass\ Density(\lambda_i) = \frac{M_i}{2\pi R_i}} \qquad \text{....Eq.(4)}$$

Where M_i=Mass of every planet, R_i=Planet's Orbital Radius, i =2nd to 8th planets.

The above Eq.(4) can be achieved by the following coding.

```
import pandas as pd
import numpy as np
a_pl_mass=np.array(df.planet_mass)
a_o_r=np.array(df.orbital_radius)
length=len(a_pl_mass)
```

```
lamda_list=['...',]
for i in range(1,length):
    lamda_i=(a_pl_mass[i]*(10**24))/(2*np.pi*a_o_r[i]*(10**9))
    lamda_list.append(lamda_i)
df['Li']=lamda_list
```

The last line of above code adds one column to the existing DataFrame.

3. **Calculating the Forces acting on Mercury:** In order to compute the forces acting on Mercury due to Sun (which is actually a centripetal force towards sun, denoted by $F_0(a)$ and named by force_a0 in program segment) and other planets (which is an outward radial force, denoted by $F_1(a)$ and named by force_a1 in program segment), we will apply the following equations.

$$\boldsymbol{F_0(a) = \frac{-GM_0m_c}{a^2}} \qquad \text{....(5)}$$

$$\boldsymbol{F_1(a) = G\pi m_c \sum_{i=2}^{8} \lambda_i \frac{a}{R_i^2 - a^2}} \qquad \text{...(6)}$$

where G=Gravitational Constant, M_0=Mass of Sun, m_c =Mass of Mercury, a=Distance from Sun to Mercury and R_i is the orbital radius of ith planet.

The above Eq.(5) and Eq.(6) can be achieved by the following coding.

```
mass_mercury=a_pl_mass[0]*(10**24)
radius_mercury=a_o_r[0]*(10**9)
length=len(a_pl_mass)
for i in range(1,length):
    r2_a2=((a_o_r[i]*(10**9))**2)-(radius_mercury**2)
    lamda_i_a_r2_a2=(lamda_i*radius_mercury)/(r2_a2)
    force_a1_sum=force_a1_sum+lamda_i_a_r2_a2
force_a0= (-(G*mass_sun*mass_mercury)/(radius_mercury**2))
force_a1=G*np.pi*mass_mercury*force_a1_sum
```

4. **Calculating $\boldsymbol{F_1^{/}(a)}$:** Derivatives are actually useful when we want to find out the rate of changes of a quantity w.r.t. the other quantity. And as the planets (except Mercury) are in different position in their orbit at different time, hence we should take derivative of $F_1(a)$ as follows:--

$$\boldsymbol{F_1^{/}(a) = G\pi m_c \sum_{i=2}^{8} \lambda_i \frac{R_i^2 + a^2}{(R_i^2 - a^2)^2}} \qquad \text{...(7)}$$
$$= G\pi m_c S$$

where G, m_c, R_i ,i ,a have their usual meaning and S is the summation of derivate of $F_1(a)$.

The above Eq.(7) can be achieved by the following coding.

```
for i in range(1,length):
        force_a1_sum_derivative=force_a1_sum_derivative+
                (lamda_i*(((a_o_r[i]*(10**9))**2)+(radius_mercury**2)))/(r2_a2**2)
force_a1_derivative=G*np.pi*mass_mercury*force_a1_sum_derivative
```

In the above code, value of S(named by force_a1_sum_derivative in the segment) is calculated inside the For Loop and then multiplied outside to get the value of $F_1^{/}(a)$.

5. **Calculating the Apsidal angle(φ):** We are pretty sure about the fact that any central force produces a circular orbit and thus Mercury has a stable orbit which exhibits the property of Simple harmonic motion. Hence, apsidal angle(φ), is the angle between two apsides, is measured with the help of the following equation(neglecting the second-order term).

$$\boldsymbol{Apsidal\ angle(\varphi) = \pi\left[1 - \frac{F_1(a) + {}^{a}/_{2}F_1^{/}(a)}{F_0(a)}\right]} \qquad \textbf{...}(8)$$

The above Eq.(8) can be achieved by the following coding.

```
small_shift=round(((force_a1+((radius_mercury*force_a1_derivative)/2))/
                                        force_a0),10)
apsidal_angle=np.pi*(1-small_shift)
```

As the value of $F_0(a)$ is negative, variable small_shift holds negative value accordingly (i.e., -9.872×10^{-7}) which is some multiple of half of total Angular Distance(i.e., -9.872×10^{-7}) Which in turn increases the apsidal angle.

6. **Calculating the Rate of Precession($\boldsymbol{\omega}$):** We all know that normally in a circular motion, any rotating or orbiting body covers an Angular Distance of 2π. Thus 2π/T (where T is the Orbital time period of Mercury) is used to measure the Angular velocity which is expressed in Radians per unit time OR Degrees per unit time. Here, in this case, the total Angular Distance covered by Mercury under the perturbing force is 2φ (where is the apsidal angle between two apsides). Therefore, the Rate of Precession is calculated by the following equation.

$$\boldsymbol{Precession\ Rate(\omega) = \frac{2\varphi - 2\pi}{T}} \qquad \textbf{....}(9)$$

Now, Eq.(9) can be evaluated by the following coding, provided that the value of the apsidal angle is calculated previously.

```
o_t_p_mercury=df.loc[0,'time_period']#Picking Mercury's Time period from DataFrame.
Precession_Rate=((((2*apsidal_angle)-(2*np.pi))*(180/np.pi)*3600*365.25*100)/
                        o_t_p_mercury)
print("\nRate of Precession(calculated) is= ",Precession_Rate,"arcsec/century")
```

In the above coding, after subtracting 2π from 2φ, we have got an extra precession expressed in Radian which is converted in a suitable unit of arc sec by multiplying Radian with $180/\pi$ (to convert in Degree) ,then by 3600(to convert in DMS(i.e.,degree, minute, second)) and lastly by converting time period in century(i.e., dividing day by 365.25(making it year(which is the actual length of a year)) and then by 100(making it century)). Here the output is +531.2 arc sec/century.

7. **Writing to Excel:** In order to insert all those calculations, we will have to write the Python code as follows:

```
with pd.ExcelWriter('C:/New_Mercury_peri_precession.xlsx') as writer:
        df.to_excel(writer,sheet_name="DataBase_modified",index=False)
        force_precession_list=[{'Force(due to Sun)':[force_a0,"Newton"],
                                'Force(due to OtherPlanets)':[force_a1,"Newton"],
                                'Apsidal Angle':[apsidal_angle,"Radian"],
                        'PrecessionRate(Calculated)':[Precession_Rate,"arcsec/century"]}]
        df1=pd.DataFrame(force_precession_list)
        df1.to_excel(writer,sheet_name="Force_Precession_Data",index=False)
```

The above code creates one new Excel file named New_Mercury_peri_precession in the specified path with two sheets named as DataBase_modified and Force_Precession_Data having index as False just because of discarding the extra column that DataFrame always generates in extreme left.

8. **Checking of Excel Creation:** We can check our newly created Excel file in two ways. One is ,we can open the file directly from the system drive by clicking on it. And another way to check the content is to have an access over that Excel file from the Python Environment through the following coding:-

```
import pandas as pd
df=pd.read_excel("New_Mercury_peri_precession.xlsx")
df.tail(8)
```

The last line of the above code displays the last eight records of excel file. Although in this case, we have only eight rows. Instead of writing df.tail(8), we may code as df.head() which displays first five records by default.

VI. DISCUSSION

We have taken the recent parameters /data of the planets and dwarf planets from the NASA website. The methodology developed in the paper may help study more astrophysical phenomena using the data available from different sources. It will help in coding and creating a computational model for people who like to use Python for Astronomy. In the chapter, we have considered a straightforward case of one parameter from studies of the observational parameter.

REFERENCES

[1] Helfrich, G. (2019, 10 21). *4 Python tools for getting started with astronomy*. Retrieved from opensource.com: https://opensource.com/article/19/10/python-astronomy-open-data
[2] Price, M. P., & Rush, W. F. (1979). Nonrelativistic contribution to Mercury's perihelion precession. *American Journal of Physics, 47*(6). doi:10.1119/1.11779