Esitimating the orbital periods of Jupiter's Gallelian moons and verifying Kepler's third law

Aparajita Banerjee, Pratik Kalel, Samarth M Nanjannavar
National Institute Of Science Education and Research, Bhubaneswar

Abstract
The experiment aims to measure the orbital periods of Jupiter's four largest moons: Io, Europa, Ganymede, and Callisto; the verification of Kepler's Third Law happens in parallel. For the experiment, a 6-inch telescope and a smartphone camera are used to collect the observational data over the course of multiple nights. Precise timestamps have been recorded for each image. A custom Python program was developed to process these images and isolate the positions of Jupiter and its moons within certain pixel ranges to obtain their positions relative to the planet. From this positional information, one could monitor the movement of the moons, model their orbits, and compute the orbital period of each of them.The experimental results of the orbital periods of Jupiter's four largest moons are : Io (1.7697 ± 0.0017 days), Europa (3.5806 ± 0.0206 days), Ganymede (7.2172 ± 0.0343 days), and Callisto (16.6813 ± 0.0893 days), using a 6-inch telescope and smartphone camera. The relative errors between the calculated and actual orbital periods were 0.039%, 0.833%, 0.869%, and 0.047%, respectively. The consistency of Kepler's Third Law can be shown by analyzing the relationship of the orbital periods of the moons to their average distances from Jupiter. Moreover we also verified laplace resonance ratio for Io,Europa and Ganymede. This is an example of how quite inexpensive tools and computing can be used to perform meaningful astronomical analyses.

Contents

1 Introduction

Jupiter, the biggest of all the planets in our solar system, has astonishing 95 confirmed moons and each has different orbital properties. Among them the four largest, 'Galilean' moons, consist of Io, Europa, Ganymede, and Callisto that were first viewed by Galileo Galilei in 1610. They have different periods of orbital movements, the shortest period is that of Io that is 1.77 days and longest is Callisto's of 16.69 whereas for Europa is 3.55 and Ganymede is 7.16 days. Interestingly, Io, Europa, and Ganymede are locked in a 1:2:4

orbital resonance, a dynamic interaction that causes these moons to slowly migrate outward due to tidal interactions with Jupiter. This is an example of Laplace resonance(a three-body resonance with a 1:2:4 orbital period ratio)
how is it relevant for amateur astronomers? Tracking the positions of Jupiter's moons over multiple nights improves proficiency in telescope use, star chart navigation, and data collection. Amateurs take notes on how to conduct observations, timing events, and analyzing trends. Amateur astronomers equipped with limited resources will find this experiment accessible and meaningful without having to resort to serious technology. This experiment involves calculating the orbital periods of Jupiter's moons with a 6-inch telescope and a smartphone camera. The aim is to observe the positions of these moons against Jupiter over several days (preferably 17 days or more) to track their motion and thereby calculate their orbital periods. Such observations not only teach us about the mechanics of celestial motion but also connect us with the early methods of astronomical discovery.

1.1 Background Theory

Kepler's Third Law states that the square of the period (T) of a planet (or moon) is directly proportional to the cube of the semi-major axis (r) of its orbit. This is expressed mathematically by:

$$T^2 \propto r^3.$$

Thus, objects farther from the central body will take longer to complete an orbit compared to those nearer to it. Therefore, this relationship is expected to be specific and quantifiable.

This relationship can be verified in the context of Jupiter's moons. By determining the orbital periods(the time required for a moon to complete a revolution around Jupiter)and by comparing the respective average distances from Jupiter, the validity of Kepler's Third Law can be assessed. The hypothesis is that if the orbital period squared (T^2) is proportional to the orbital distance cubed (r^3), then Kepler's Law is valid for Jupiter's moons.

For the purpose of this experiment, the moons are assumed to be in circular orbits around Jupiter and the periodic function obtained by plotting distance from jupiter and time as a sine function. These assumption were made to facilitate simple calculations, as the distance from Jupiter to the moon would vary in an elliptical orbit but remain constant in a circular orbit. Furthermore, treating the orbit as a circle makes the application of Kepler's Third Law easier, since the distance remains constant and is not subject to the variability of elliptical orbits.

In reality, the orbits of the moons are not perfectly circular at all times. However, since the variations in their orbits are relatively small, it is a reasonable approximation to treat the moons as moving in circular orbits. This allows for a sufficiently accurate estimation of the moons' orbital periods and provides a means for qualitatively verifying Kepler's Third Law. By focusing solely on the distance and time in orbit, this simplification avoids the complications of elliptical motion.

2 Description of the experiment

2.1 Experimental setup

The experimental setup is simple: all you need is a 6-inch telescope, a 10mm eyepiece ,a smartphone for data collection, and a commitment to observing the night sky during a total period of about 17 days.

- 6-Inch Telescope:
  The 6-inch telescope with a 150 mm aperture and a focal length of 750mm is a suitable
- size for viewing Jupiter and its moons. Because of its larger aperture, clear details of Jupiter's atmosphere in addition to the positional information of its Galilean moons might be resolved with more light. It typically configures itself as a Newtonian reflector, having a parabolic main mirror and flat secondary mirror in such a way that chromatic aberration is reduced and images are clearer.
- 10mm eyepiece :
  A 10mm eyepiece provides sufficient magnification i.e 75x which allows us to clearly resolve Jupiter and its 4 moons.
- mobile holder :
  A mobile holder is used for stable imaging ,consistent alignment and hand free operation necessary to obtain images without trails

2.2 Experimental procedure and data acquisition

Instructions for Setting Up a 6-inch Telescope

1. Take the tripod of the 6-inch telescope and position one of its legs toward the south.(make sure the legs are at the maximum separation).
2. Attach the mount to the tripod, ensuring it is aligned to face the polar star (north).
3. Secure the counterweights on the mount.
4. Carefully attach the telescope to the mount, making sure the screws are tightened properly.
5. Install accessories such as the eyepiece, mobile holder,mobile and adjustment bolts.
6. Balance the telescope for smooth operation and stability.
7. Align viewfinder and eyepiece using a distinct stable red light source.
8. Point telescope towards Jupiter using a viewfinder and tight the screws.
9. Use fine adjusters to take Jupiter in the centre.
10. Change focus of the eyepiece accord- ingly.
11. Attach the mobile with the eyepiece with the help of the mobile holder.
12. Make sure Jupiter and its moon should be visible properly.
13. Change the settings in the pro mode of camera to get the best image possible.
14. Adjust your camera settings to reduce blur and pixel scattering (for this you can increase shutter speed reduce exposure time, adjust white balance).
15. Set a timer for 3 to 5 seconds and let it capture the image without disturbing the setup.
16. It is advisable to capture a short video

sequence in adverse weather conditions, such as when Jupiter experiences drift or blurring. Subsequently, individual frames can be extracted from the video.

17. repeat these steps for approx 17 days or more.

After we are done with collecting enough images for each day we will start our data modelling:

- The first step involves selecting the best three images per day from the stack of images. The criterion for a good image is:

    1. The pixels representing Jupiter and its moons should be sharply concentrated, with no visible trailing or blurring.

    2. All images should maintain a consistent orientation, ensuring they are aligned in the same orientation throughout the observations.

Figure 1: Example of a good image

Figure 2: Example of a bad image. you can see trails and blur here

Figure 3: Images at the bottom have consistent orientation whereas the one above have a different orientation, the above image needs to be discarded.

- From bottom to up, label the moons as a, b, c, and d, and Jupiter as J (in every image, irrespective of what they actually are. There might be different labeling of the images. This is fine because it will identify the correct identity during the later analysis.)

2.3 Determining the coordinates

The coordinates were determined using the code that has been explained thoroughly in the appendix section (appendix 6.1). The coordinates were determined according to the labeling that was done previously and the coordinates were entered in Table 1.

Table 1: Coordinates of Jupiter(J) and Moons(M) (a, b, c, d) at Different Times(T)

T (hrs)	J (x)	J (y)	Ma (x)	Ma (y)	Mb (x)	Mb (y)	Mc (x)	Mc (y)	Md (x)	Md (y)
0.00	1,541.63	1,260.79	1,451.89	1,641.29	1,470.08	1,460.52	1,506.82	1,228.51	1,541.63	615.22
23.47	1,761.27	1,242.60	1,882.12	1,506.93	1,805.47	1,321.76	1,785.20	1,317.67	1,535.59	770.20
50.35	1,879.61	1,632.74	1,896.31	1,595.75	1,871.37	1,482.83	1,862.24	1,390.59	1,832.72	1,330.10
74.78	1,697.13	1,424.65	1,698.43	1,568.71	1,687.06	1,473.74	1,666.11	1,360.68	1,705.80	1,107.24
96.78	2,218.54	1,256.84	2,226.30	1,480.64	2,190.52	1,421.08	2,212.27	1,123.01	2,213.50	889.92
121.60	2,221.68	1,551.15	2,427.15	1,932.29	2,269.63	1,618.94	2,154.14	1,427.71	2,111.80	1,412.20
145.18	1,843.00	1,412.24	1,920.31	2,009.33	1,856.89	1,570.84	1,839.35	1,377.45	1,813.41	1,236.89
192.28	2,299.78	1,014.78	2,226.57	1,661.32	2,270.66	1,325.34	2,296.68	1,125.75	2,286.24	1,073.29
215.40	2,549.01	819.33	2,669.57	1,386.26	2,539.61	724.13	2,519.08	593.74	-	-
238.58	2,128.41	829.71	2,052.80	1,214.12	2,093.35	949.36	2,132.96	765.73	2,207.44	591.79
262.42	1,589.51	1,015.39	1,497.88	1,239.77	1,553.61	1,182.71	1,646.69	875.14	1,730.59	652.87
288.53	1,626.28	1,160.13	1,558.93	1,273.35	1,676.44	1,100.88	1,710.52	1,084.44	1,728.24	963.55
313.02	2,520.70	1,399.58	2,480.41	1,505.19	2,524.79	1,316.92	2,546.56	1,186.02	2,608.48	1,051.81
337.33	2,042.02	1,579.50	1,894.19	1,930.50	1,971.51	1,732.55	2,266.85	1,087.49	-	-
360.02	2,367.47	1,303.43	2,416.95	1,673.06	2,395.35	1,470.46	2,320.10	670.26	-	-
384.18	1,052.84	1,552.44	1,043.71	1,653.13	1,088.84	1,464.27	1,133.13	1,321.71	1,262.79	848.51
406.80	1,827.58	1,516.91	1,785.21	1,602.28	1,854.64	1,404.18	1,920.59	1,304.01	2,030.95	918.14
433.07	2,393.99	1,435.03	2,473.35	1,696.69	2,370.83	1,301.64	2,312.74	1,073.88	2,274.49	1,013.88

3 Analyses

3.1 Data Analysis

The distance table is created using the coordinates from Table 1. We used the Euclidean distance formula to calculate it:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

where x_2 and y_2 represent the coordinates of Jupiter at any time t, and x_1 and y_1 represent the coordinates of a moon at the corresponding time. This allows us to calculate the distances from Jupiter, treating it as the origin. Additionally, we applied a convention to aid our analysis: the moon located below Jupiter is assigned a negative sign for its distance. (For example, if you take the data, it may happen that the moon to the right of Jupiter is assigned a negative sign, etc.) This is simply a convention.

Table 2: Distance Table Showing Distances of Moons (a, b, c, d) from Jupiter at Different Times

Time (hrs)	D_ja	D_jb	D_jc	D_jd
0.000	-390.935	-212.155	47.484	645.578
23.467	-290.643	-90.660	-78.794	523.538
50.350	40.588	150.137	242.779	306.253
74.783	-144.065	-50.116	71.089	317.524
96.783	-223.929	-166.612	133.981	366.952
121.600	-432.992	-83.027	140.713	177.152
145.183	-602.074	-159.210	34.983	177.834
192.283	-650.667	-311.917	-111.016	-60.055
215.400	-579.614	95.656	227.559	0.000
238.583	-391.774	-124.677	64.146	250.702
262.416	-242.367	-171.132	151.453	388.997
288.533	-131.742	77.627	113.247	221.450
313.016	-113.034	82.764	215.117	358.677
337.333	-380.854	-168.513	540.942	0.000
360.016	-372.929	-169.337	634.938	0.000
384.183	-101.100	95.240	244.306	734.579
406.800	-95.311	115.929	232.328	632.358
433.066	-273.428	135.385	370.179	437.780

3.1.1 Data Modelling

After constructing the distance table, we began the process of identifying the moons. From the table, we identified the maximum modulus distance from Jupiter as 734.578. However, to account for the possibility that the largest separation of the farthest moon might have been missed, we set the maximum distance to 750(m1). All distances in the table were then divided by this value(table 3). Next, we plotted these normalized distances with respect to time (Fig:3). This will further aid our visualization of the data.

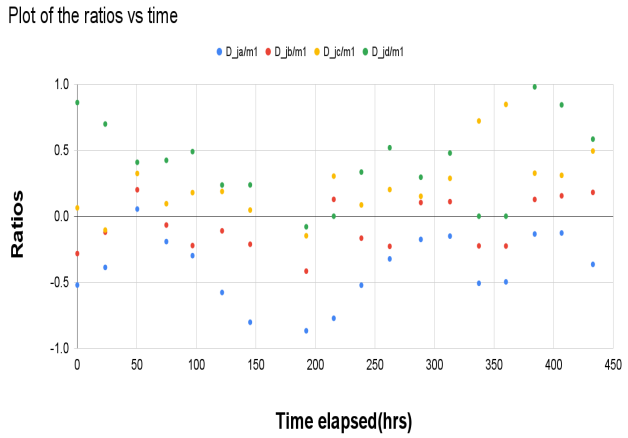


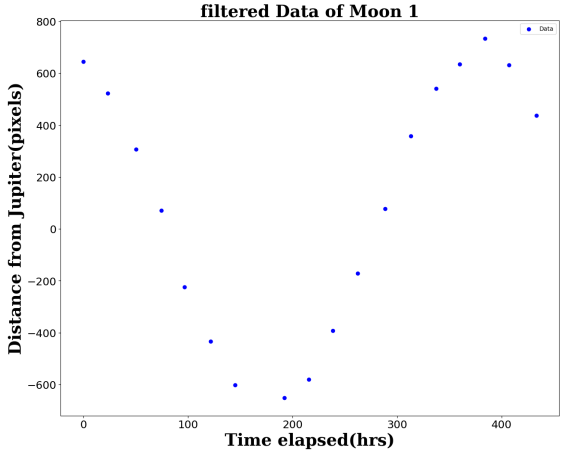
Figure 4: The plots from data in table 3

To identify the moons, we located the maximum and minimum points on the plot and visualize a sine curve that passes through these extrema. This is quiet easy for the farther away moons, Data points along this curve were attributed to the farthest moon. You can also see this in the above plot(Fig:3), the maximum is first green data point, then blue data point is the minimum and then the green data point at the end is another maximum point, we can also visualize a sine curve passing through these points. These points were then isolated, and the corresponding distances were removed from the distance table (table 2), and made a new distance table(table 4). This was identified as moon 1.

Table 4: Distance Table with Data Points Removed

Time (hrs)	**D_ja**	**D_jb**	**D_jc**	**D_jd**
0.000	-394.647	-213.492	47.484	-
23.467	-290.643	-90.660	-78.794	-
50.350	40.588	150.137	242.779	-
74.783	-144.065	-50.116	71.089	317.524
96.783	-	-166.612	133.981	366.952
121.600	-	-83.027	140.713	177.152
145.183	-	-159.210	34.983	177.834
192.283	-	-311.917	-111.016	-60.055
215.400	-	95.656	227.559	-
238.583	-	-124.677	64.146	250.702
262.416	-242.367	-	151.453	389.997
288.533	-131.742	-	113.247	221.450
313.016	-113.034	82.764	215.117	-
337.333	-380.854	-168.513	-	-
360.016	-372.929	-169.337	-	-
384.183	-101.100	95.240	244.306	-
406.800	-95.311	115.929	232.328	-
433.066	-273.428	135.385	370.179	-

We repeated the same procedure to isolate the data points corresponding to the second farthest, third farthest, and finally the closest moon to Jupiter (i.e., the fourth farthest). All this was done in Google sheets, as it is much easier to handle the data. We finally obtained the distance of the different moons of the Jupiter. we plotted them to get a better visualization.



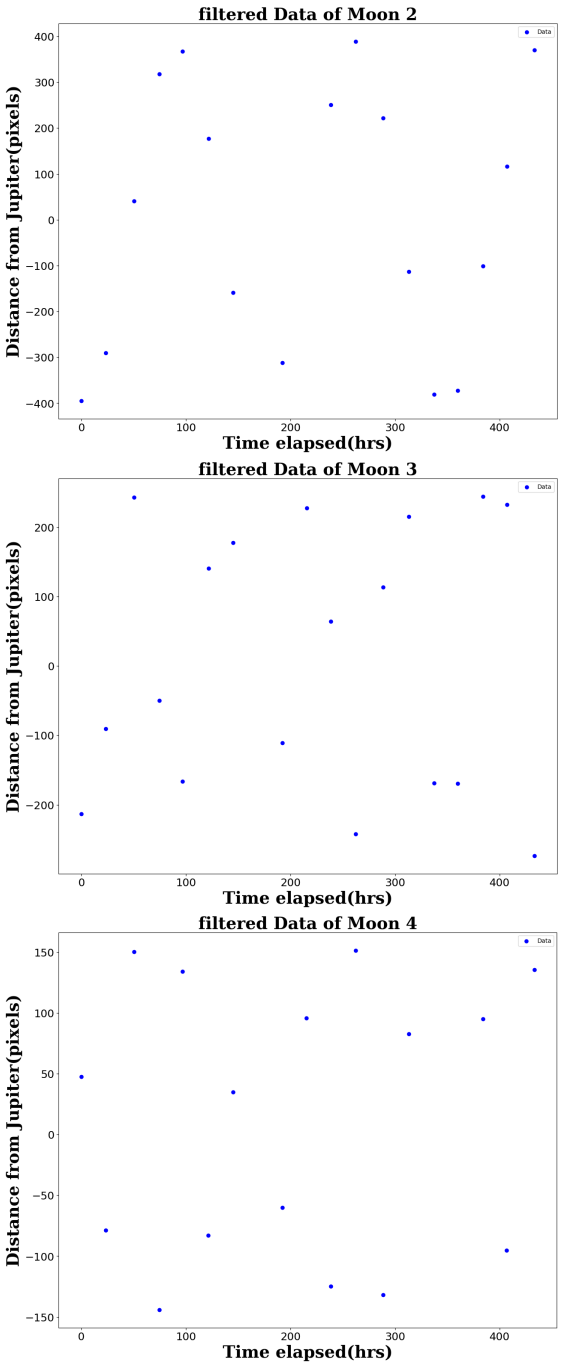


Figure 5: plots of Filtered data of moons

After isolating the data for each moon, we used Python to fit the points to a sine curve with minimal error. The model function used was $Asin(\omega t + c)$, where A, ω, and c are the fitting parameters. This model is consistent with our assumption that the orbits are approximately circular, and since we are observing Jupiter and its moons edge-on, the projection of a circular orbit appears as a harmonic function. After the initial fitting, we observed that some data points were better suited to other moons, so we refined the dataset by reassigning points based on their alignment with the fitted curves (Fig: 5).

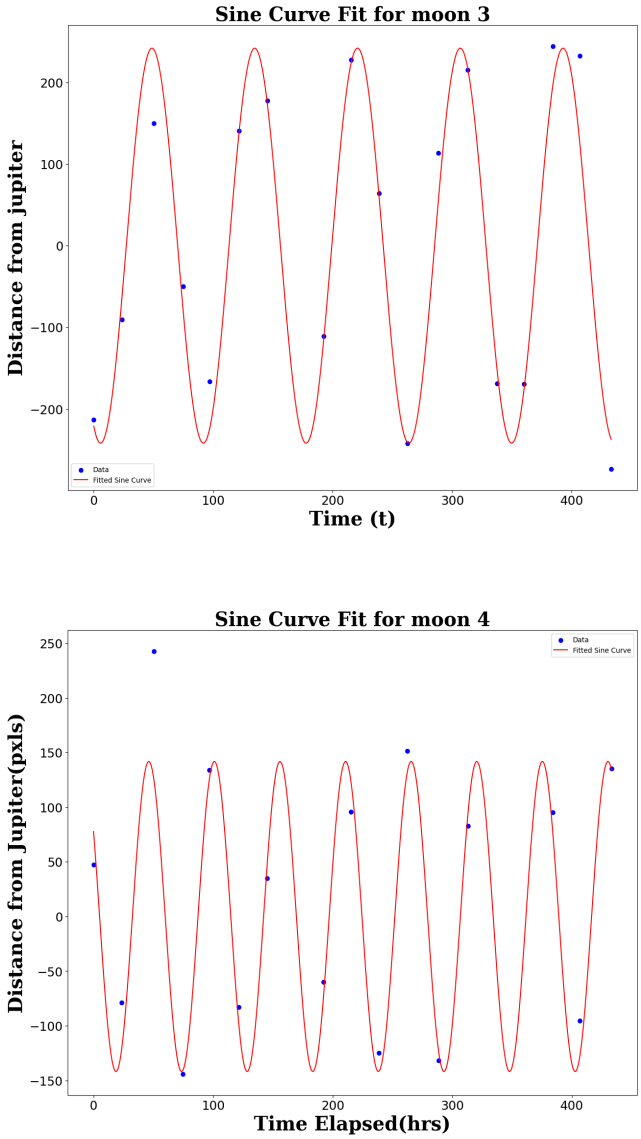


Figure 6: Plots that tell us more about which data point belongs to which moon, after the first attempt to fit the sine function

Refering to the above figure(figure 8). Look at the third data plots in both plots. It is way lower ($\approx$ 150pxls) and higher($\approx$ 250) than the predicted values by the respective sin curve fits($\approx$ 250 and $\approx$ 150 respectively) . So, its evident that these two data points must be

exchanged in order to get a better fit. This was further identified in other data points, and these data points were sorted accordingly.

Table 5: Fitered distances identifying the moon

Time (hrs)	**Moon 1**	**Moon 2**	**Moon 3**	**Moon 4**
0.00	645.578	-394.647	-213.492	47.484
23.46	523.538	-290.643	-90.660	-78.794
50.35	306.253	40.588	242.779	150.137
74.78	71.089	317.524	-50.116	-144.065
96.78	-223.929	366.952	-166.612	133.981
121.60	-432.992	177.152	140.713	-83.027
145.18	-602.074	-159.210	177.834	34.983
192.28	-650.667	-311.917	-111.016	-60.055
215.40	-579.614	-	227.559	95.656
238.58	-391.774	250.702	64.146	-124.677
262.41	-171.132	389.997	-242.367	151.453
288.53	77.627	221.450	113.247	-131.742
313.01	358.677	-113.034	215.117	82.764
337.33	540.942	-380.854	-168.513	-
360.01	634.938	-372.929	-169.337	-
384.18	734.579	-101.100	244.306	95.240
406.80	632.358	115.929	232.328	-95.311
433.06	437.780	370.179	-273.428	135.385

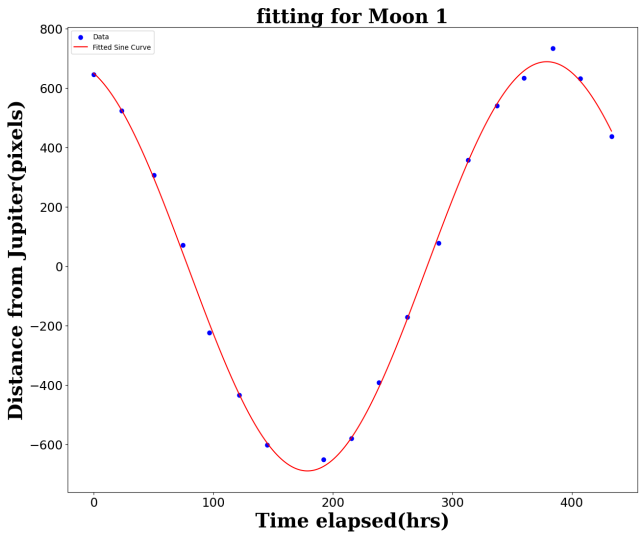


Figure 7: Fitted parameters:
A = 688.9858 ± 7.1254
w = 0.0157 ± 0.0001
c = 1.9060 ± 0.0197
time period in hrs: 400.3510 ± 2.1428
time period in days: 16.6813 ± 0.0893
the fit sin curve is y = 688.9858 sin(0.0157 t + 1.9060)

Once the final sorting was complete, we got the corresponding distance table with the moons identified(table 5) we fit the data points again and obtained the corresponding plots(Fig: 6 -10). the code and the explanation for the code that is used for fitting and the the reason for choosing the model function can be found in the appendix section(Appendix 6.2).
The sine function is defined by:

y= Asin(ωt +c)
y= Distance for Jupiter to Moon
A= Amplitude of the sine graph
ω= Angular frequency
t= time elapsed
c=Phase difference

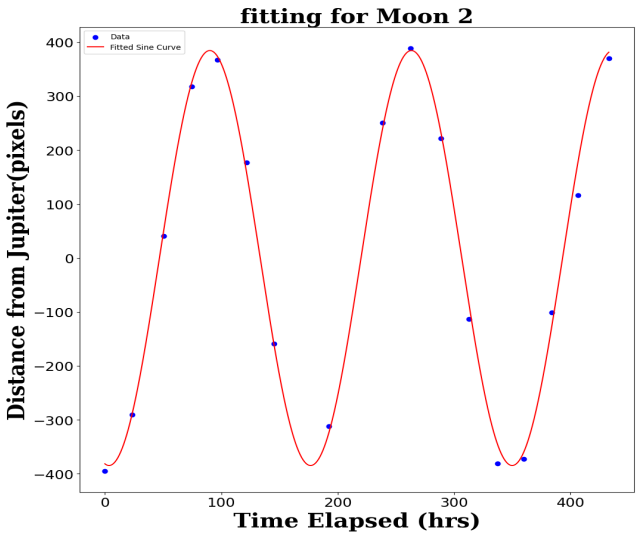


Figure 8: Fitted parameters:
A = -384.6231 ± 8.0633
w = 0.0363 ± 0.0002
c = 1.4422 ± 0.0445
time period in hrs: 173.2124 ± 0.8239
time period in days: 7.2172 ± 0.0343
the fit sin curve is y = -384.6231 sin(0.0363 t + 1.4422)

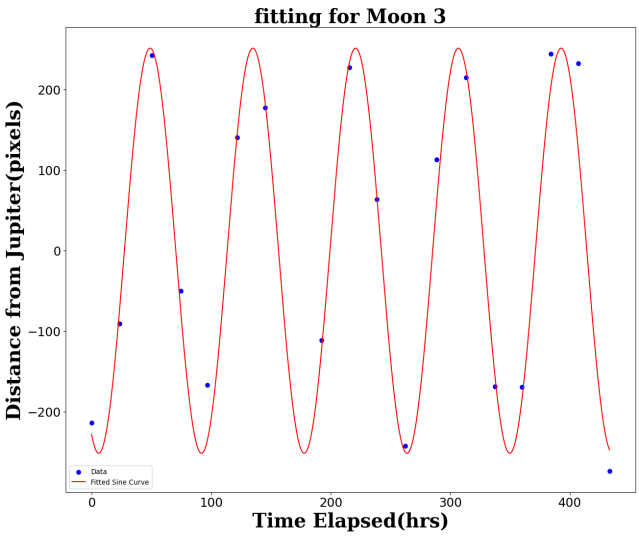


Figure 9: Fitted parameters:
A = 251.4782 ± 12.6323
w = 0.0731 ± 0.0004
c = -2.0053 ± 0.1047
time period in hrs: 85.9349 ± 0.4955
time period in days: 3.5806 ± 0.0206
the fit sin curve is y = 251.4782 sin(0.0731 t + -2.0053)

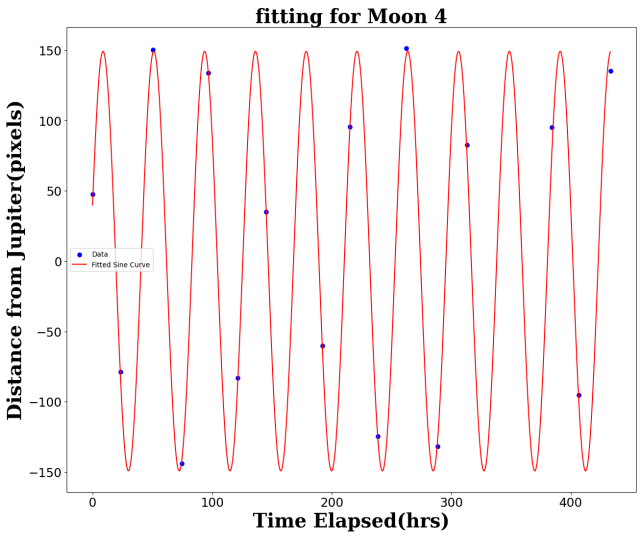


Figure 10: Fitted parameters:
A = 149.2201 ± 2.5679
w = 0.1479 ± 0.0001
c = 0.2702 ± 0.0316
time period in hrs: 42.4737 ± 0.0398
time period in days: 1.7697 ± 0.0017
the fit sin curve is y = 149.2201 sin(0.1479 t + 0.2702)

We went ahead and calculated the ratio of their time periods, the ratio of time periods are, M2/M1 is 2.023, M3/M1 is 4.078 and M4/M1 is 9.425. So. based on these periods, we identified moon 1 as Callisto, moon 2 as Ganymede, moon 3 as Europa, and moon 4 as Io.

3.1.2 Verifying Kepler's third law and finding the ratios of the time period

From the determined time periods and the amplitudes of the orbits of the moons we moved on to verify Kepler's 3rd law, consider the data in the following table:

	Amp. (A)	**Period (T)**
Callisto	688.985	400.351
Ganymede	384.623	173.212
Europa	251.478	85.9349
Io	149.220	42.473

Table 6: Amplitude and Period of the moons

	$\mathbf{A}^3(10^8)$	$\mathbf{T}^2$
Callisto	3.270	160280.946
Ganymede	0.569	30002.541
Europa	0.159	7384.803
Io	0.033	1804.011

Table 7: Amplitude cubed and Period squared of the moons

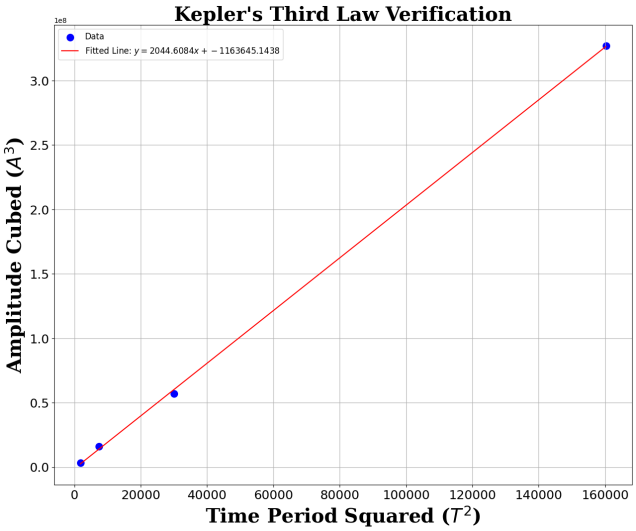


Figure 11:

3.2 Error Analysis

- Errors in the fitted parameter:

  The uncertainties in the fitted parameters are derived from the covariance matrix calculated during the optimization process in `scipy.optimize.curve_fit`. The covariance matrix, $\mathbf{V}$, is related to the inverse of the Hessian matrix of the least-squares cost function, $\mathbf{V} = (\mathbf{J}^T\mathbf{J})^{-1}\sigma^2$, where $\mathbf{J}$ is the Jacobian

matrix of the residuals and σ^2 is the variance of the data points. The diagonal elements of the covariance matrix represent the variances of the parameters, and their square roots provide the standard errors (uncertainties). This process assumes that the residuals are normally distributed, the model accurately represents the data, and the data points have uncorrelated errors. If these assumptions hold, the uncertainties offer a reliable measure of the confidence in the fitted parameters. In Python, the uncertainties are computed as `np.sqrt(np.diag(pcov))`, where `pcov` is the covariance matrix output by the fitting process.

- Error in time period: Time period is calculated using the formula:

$$\omega = \frac{2\pi}{T}$$

ω as well as error in $\Delta\omega$ is already calculated by fitting the data into a sine curve.

Relative error:

$$\frac{\Delta T}{T} = \frac{\Delta\omega}{\omega}$$

By rearranging this we get

$$\Delta T = T \cdot \frac{\Delta\omega}{\omega}$$

- How accurate are we?:

  the literature value of the orbital periods,[1]

  1. The orbital period of Io =1.769 days
  2. The orbital period of Europa =3.551 days
  3. The orbital period of Ganymede =7.155 days
  4. The orbital period of Callisto =16.689 days

Moons	Determined value	Relative error
Io	1.7697	0.039 %
Europa	3.5806	0.833 %
Ganymede	7.2172	0.869%
Callisto	16.6813	0.047 %

Table 8: Accuracy of obtained results

4 Results and inferences

- The results obtained were:

  1. The orbital period of Io =(1.7697± 0.0017) days
  2. The orbital period of Europa =(3.5806 ± 0.0206) days
  3. The orbital period of Ganymede =(7.2172 ± 0.0343) days
  4. The orbital period of Callisto =(16.6813 ± 0.0893) days

The relative errors between the actual and calculated value of time period are 0.039% ,0.833%,0.869% and 0.047% respectively, as calculated previously(table 8). Which tells us that the the values we have obtained are very accurate.

- From the graph obtained between A^3 and T^2 (Figure 10)

  $\frac{T^2}{A^3} = Constant$

  Hence verifying Kepler's third law.

- We went ahead and also verified the Laplace orbital resonance ratio between Io, Europa, and Ganymede i.e 1:2:4
  We calculated the ratio to be 1:2.02:4.08

5 Conclusions

This experiment successfully calculates the orbital period of Jupiter's moons which turned out to be closely aligned with the actual value. By assuming the orbits to be circular we also verified Kepler's third law which establishes a relation between time period and the semi-major axis of the orbit. Moreover we also verified laplace resonance ratio.
There errors can be justified by the following sources:

1. The coordinates were calculated by Gaussian fitting and finding the mean of the Gaussian. But the plot obtain was not exactly a Gaussian curve but a much more complex curve. Assuming it to be Gaussian reduced the complexity of the problem and gave results with reasonable accuracy.
2. While modelling we simplified our data into a sine function by normalizing it . But in reality the function obtained is a periodic function and not necessarily a sine function ,rather we can say it is a Fourier series. But to reduce complexity the periodic function was assumed to be a sine function which helped us easily calculate ω and hence the periods with reasonable accuracy.
3. Other source can be the alignment of our telescope. The telescope's optics may diverge from the intended target due to misalignment, which might result in aberrations or distortions in the picture that is viewed. Measurements might become more difficult because to overlapping or hazy views of celestial bodies caused by poor alignment.
4. One improvisation that can be made is taking hourly data for the initial 3 to 4 days this will help you obtain the curve for moons with short time period like io more accurately and easily. As in our data set we can see the values have alternating signs for io which made it difficult to plot a smooth graph

6 Appendix

6.1 Python script to calculate the coordinates

1: Script to calculate the cordinates

```
import cv2
import matplotlib.pyplot as plt
import numpy as np
from scipy.optimize import
    curve_fit

def gaussian(x, amp, mu, sigma):
    return amp * np.exp(-((x - mu
        ) ** 2) / (2 * sigma ** 2)
        )

image_path = r"C:\Users\91935\
    Downloads\20241113_230358.jpg"
image = cv2.imread(image_path)
```

```
gray_image = cv2.cvtColor(image,
    cv2.COLOR_BGR2GRAY)
gray_image1 = gray_image + 1e-5

c = 255 / np.log(1 + np.max(
    gray_image1))
log_image = c * (np.log(
    gray_image1))

log_image = np.clip(log_image, 0,
     255).astype(np.uint8)


plt.figure(figsize=(22, 15))
plt.imshow(log_image, cmap='gray'
    )
plt.title("Logarithmic
    Transformed Image")
plt.axis('on')

plt.grid(True, which='both',
    color='white', linestyle='-',
    linewidth=0.1)

x_step = int(input("Enter the x-
    gridline step size"))
y_step = int(input("Enter the y-
    gridline step size"))

plt.xticks(np.arange(0, log_image
    .shape[1], step=x_step))
plt.yticks(np.arange(0, log_image
    .shape[0], step=y_step))
plt.xticks(rotation=90)
plt.show()

A = input("Are you isolating
    jupiter? (y/n)")

if A == "y":
    #isolating jupiter
    x_cord = int(input("Enter the
        guess x coordinate of the
        jupiter"))
    y_cord = int(input("Enter the
        guess y coordinate of the
        jupiter"))

ROI = gray_image[y_cord-35:
    y_cord+35 , x_cord-35:
    x_cord+35]

plt.figure(figsize=(10, 10))
plt.imshow(ROI, cmap='gray')
plt.title("Region Of Interest
    ")
plt.axis('on')
plt.xticks(np.arange(0, ROI.
    shape[1], step=1))
plt.yticks(np.arange(0, ROI.
    shape[0], step=1))
plt.xticks(rotation=90)
plt.show()

#Fitting along x axis
y_con =  int(input("Enter the
    constant y for the
    jupiter"))
X_data = ROI[y_con,:]
x_values = np.arange(len(
    X_data))

valid_indices = X_data < 251
X_data_f = X_data[
    valid_indices]
x_values_f = x_values[
    valid_indices]
initial_guess_x = [np.max(
    X_data_f), x_values_f[np.
    argmax(X_data_f)], 1.0]
popt_x, pcov_x = curve_fit(
    gaussian, x_values_f,
    X_data_f, p0=
    initial_guess_x)
amp_opt_x, mu_opt_x,
    sigma_opt_x = popt_x
A_err_x, mu_err_x,
    sigma_err_x = np.sqrt(np.
    diag(pcov_x))
fitted_y_values = gaussian(
    x_values, *popt_x)
plt.scatter(x_values, X_data,
     label="Original Data",
    color="blue", s=20)
```

```
plt.plot(x_values,
    fitted_y_values, label="
    Fitted Gaussian", color="
    red", linewidth=2)

plt.xlabel("X values")
plt.ylabel("Amplitude")
plt.title("Gaussian Fit to
    Data X axis")
plt.xticks(np.arange(0, len(
    X_data), step=1))
plt.legend()
plt.show()

print(f"Optimized parameters:
    Amplitude_x = {amp_opt_x
    }, Mean (mu)_x = {mu_opt_x
    }, Sigma_x = {sigma_opt_x}
    ")

#Fitting along y axis
x_con =  int(input("Enter the
    constant x for the
    jupiter"))
Y_data = ROI[:,x_con]
y_values = np.arange(len(
    Y_data))

Y_data_f = Y_data[
    valid_indices]
y_values_f = y_values[
    valid_indices]
initial_guess_y = [np.max(
    Y_data_f), y_values_f[np.
    argmax(Y_data_f)], 1.0]
popt_y, pcov_y = curve_fit(
    gaussian, y_values_f,
    Y_data_f, p0=
    initial_guess_y)
amp_opt_y, mu_opt_y,
    sigma_opt_y = popt_y
A_err_y, mu_err_y,
    sigma_err_y = np.sqrt(np.
    diag(pcov_y))
fitted_x_values = gaussian(
    y_values, *popt_y)
plt.scatter(y_values, Y_data,
     label="Original Data",
    color="blue", s=20)
plt.plot(y_values,
    fitted_x_values, label="
    Fitted Gaussian", color="
    red", linewidth=2)

plt.xlabel("Y values")
plt.ylabel("Amplitude")
plt.title("Gaussian Fit to
    Data for Y axis")
plt.xticks(np.arange(0, len(
    Y_data), step=1))
plt.legend()
plt.show()

print(f"Optimized parameters:
    Amplitude_y = {amp_opt_y
    }, Mean (mu)_y = {mu_opt_y
    }, Sigma_y = {sigma_opt_y}
    ")

#Determining the center of
    the jupiter
x_cent = x_cord-35+mu_opt_x
y_cent = y_cord-35+mu_opt_y

#Sanity check
Sanity_ROI = gray_image[int(
    y_cent)-35:int(y_cent)+35
    , int(x_cent)-35:int(
    x_cent)+35]

plt.figure(figsize=(10, 10))
plt.imshow(Sanity_ROI, cmap='
    gray')
plt.title("Sanity Check")
plt.axis('on')
plt.xticks(np.arange(0,
    Sanity_ROI.shape[1], step
    =1))
plt.yticks(np.arange(0,
    Sanity_ROI.shape[0], step
    =1))
plt.show()
```

```
    print("the (x,y) central
        coordinate of the jupiter
        is(", x_cent,"  ",mu_err_x
        ,",", y_cent,"  ",mu_err_y
        ,")")
else:
    #isolating a moon
    x_cord1 = int(input("Enter
        the guess x coordinate of
        the object"))
    y_cord1 = int(input("Enter
        the guess y coordinate of
        the object"))

    ROI1 = gray_image[y_cord1-10:
        y_cord1+10 , x_cord1-10:
        x_cord1+10]

    plt.figure(figsize=(10, 10))
    plt.imshow(ROI1, cmap='gray')
    plt.title("Region Of Interest
        ")
    plt.axis('on')
    plt.xticks(np.arange(0, ROI1.
        shape[1], step=1))
    plt.yticks(np.arange(0, ROI1.
        shape[0], step=1))
    plt.show()

    #Fitting along x axis
    y_con1 =  int(input("Enter
        the constant y for the
        object "))
    X_data1 = ROI1[y_con1,:]
    x_values1 = np.arange(len(
        X_data1))

    initial_guess_x1 = [np.max(
        X_data1), np.argmax(
        X_data1), 1.0]
    popt_x1, pcov_x1 = curve_fit(
        gaussian, x_values1,
        X_data1, p0=
        initial_guess_x1)
    amp_opt_x1, mu_opt_x1,
        sigma_opt_x1 = popt_x1
    A_err_x1, mu_err_x1,
        sigma_err_x1 = np.sqrt(np.
        diag(pcov_x1))
    fitted_y_values1 = gaussian(
        x_values1, *popt_x1)
    plt.scatter(x_values1,
        X_data1, label="Original
        Data", color="blue", s=20)
    plt.plot(x_values1,
        fitted_y_values1, label="
        Fitted Gaussian", color="
        red", linewidth=2)

    plt.xlabel("X values")
    plt.ylabel("Amplitude")
    plt.title("Gaussian Fit to
        Data X axis")
    plt.xticks(np.arange(0, len(
        X_data1), step=1))
    plt.legend()
    plt.show()

    print(f"Optimized parameters:
        Amplitude_x = {amp_opt_x1
        }, Mean (mu)_x = {
        mu_opt_x1}, Sigma_x = {
        sigma_opt_x1}")

    #Fitting along y axis
    x_con1 =  int(input("Enter
        the constant x for the
        object"))
    Y_data1 = ROI1[:,x_con1]
    y_values1 = np.arange(len(
        Y_data1))

    initial_guess_y1 = [np.max(
        Y_data1), np.argmax(
        Y_data1), 1.0]
    popt_y1, pcov_y1 = curve_fit(
        gaussian, y_values1,
        Y_data1, p0=
        initial_guess_y1)
    amp_opt_y1, mu_opt_y1,
        sigma_opt_y1 = popt_y1
    A_err_y1, mu_err_y1,
        sigma_err_y1 = np.sqrt(np.
        diag(pcov_y1))
```

```
fitted_x_values1 = gaussian(
    y_values1, *popt_y1)
plt.scatter(y_values1,
    Y_data1, label="Original
    Data", color="blue", s=20)
plt.plot(y_values1,
    fitted_x_values1, label="
    Fitted Gaussian", color="
    red", linewidth=2)

plt.xlabel("Y values")
plt.ylabel("Amplitude")
plt.title("Gaussian Fit to
    Data for Y axis")
plt.xticks(np.arange(0, len(
    Y_data1), step=1))
plt.legend()
plt.show()

print(f"Optimized parameters:
    Amplitude_y = {amp_opt_y1
    }, Mean (mu) _y = {
    mu_opt_y1}, Sigma_y = {
    sigma_opt_y1}")

#Determining the center of
    the object
x_cent1 = x_cord1-10+
    mu_opt_x1
y_cent1 = y_cord1-10+
    mu_opt_y1

#Sanity check
Sanity_ROI1 = gray_image[int(
    y_cent1)-10:int(y_cent1)
    +10 , int(x_cent1)-10:int(
    x_cent1)+10]

plt.figure(figsize=(10, 10))
plt.imshow(Sanity_ROI1, cmap=
    'gray')
plt.title("Sanity Check")
plt.axis('on')
plt.xticks(np.arange(0,
    Sanity_ROI1.shape[1], step
    =1))
plt.yticks(np.arange(0,
    Sanity_ROI1.shape[0], step
    =1))
plt.show()

print("the (x,y) central
    coordinate of the object
    is(",x_cent1,"  ",
    mu_err_x1,",", y_cent1,"
    ",mu_err_y1,")")
```

1. The image is first converted to grayscale and then logarithmic transformation is applied to each pixel; this is done to enhance the details in darker regions while compressing the dynamic range of brighter regions.

2. Then gridlines are inserted in the image (you can select the size of grids depending on your image.) this helps us to isolate the region of interest.

3. For isolating Jupiter, we will consider a range of ±35 pixels, whereas for the moons, we will consider a range of ±10 pixels.

4. We have assumed the spread of light to be Gaussian, so a Gaussian fitting is performed on x and y axis and the center is determined where the optimized mean of the gaussian is from both the X-axis and Y-axis fits.

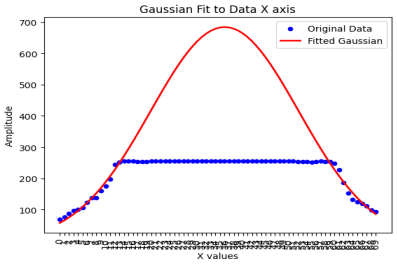


Figure 12: an example plot of the gaussian fit

5. Finally a sanity check is done to verify the determined center is correct. We get a small (20*20 grayscale image)region of interest centered around the calculated coordinates. This ensured the calcu-

lated coordinates match the visual center of the image.

6.2 Python script to fit the moons data with a sine function

2: Script to fit the moons data

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import
    curve_fit


def sine_function(t, A, w, c): #
    t-time, A-amplitude, w-angular
     frequency, c-phase difference
     return A * np.sin(w * t + c)

# data
t_data = np.array
    ([0,23.4666,50.35,74.783,
96.783,121.6,145.183,192.283,
215.4,238.583,262.416,288.533,
313.016,337.333,360.016,384.183,
406.8,433.066])
y_data = np.array
    ([645.5781415,523.5381766,
306.2528327,71.08862097,
-223.9288075,-432.9923056,
-602.0743625,-650.6665467,
-579.6136031,-391.7743009,
-171.1319462,77.62693671,
358.6773043,540.9424275,
634.9381008,734.5785418,
632.3584927,437.7801396])


# Initial guesses for parameters
    (A, w, c)
A_guess = (max(y_data) - min(
    y_data)) / 2
w_guess = 2 * np.pi / (t_data[16]
     - t_data[0])#guessing the
    period from the data
c_guess = 0
p0 = [A_guess, w_guess, c_guess]

popt, pcov = curve_fit(
    sine_function, t_data, y_data,
     p0=p0)

# parameters and errors
A, w, c = popt
A_err, w_err, c_err = np.sqrt(np.
    diag(pcov))

print(f"Fitted parameters:\nA = {
    A:.4f} {A_err:.4f}\nw = {w
    :.4f} {w_err:.4f}\nc = {c
    :.4f} {c_err:.4f}")
print(f"time period in hrs: {2 *
    np.pi / w:.4f} {2 * np.pi *
    w_err / (w * w):.4f}")
print(f"time period in days: {2 *
    np.pi / (24 * w):.4f} {2 *
    np.pi * w_err / (24 * w * w)
    :.4f}")
print(f"the fit sin curve is y =
    {A:.4f} sin({w:.4f} t + {c:.4f
    })")


t_fit = np.linspace(min(t_data),
    max(t_data), 1000)
y_fit = sine_function(t_fit, *
    popt)

plt.figure(figsize=(15, 12))
plt.scatter(t_data, y_data, label
    ="Data", color="blue")
plt.title("filtered Data of Moon
    1")
plt.xlabel("Time elapsed(hrs)")
plt.ylabel("Distance from Jupiter
    (pixels)")
plt.legend()
plt.show()

plt.figure(figsize=(15, 12))
plt.scatter(t_data, y_data, label
    ="Data", color="blue")
plt.plot(t_fit, y_fit, label="
    Fitted Sine Curve", color="red
```

```
    ")
plt.title("fitting for Moon 1")
plt.xlabel("Time elapsed (hrs)")
plt.ylabel("Distance from Jupiter
    (pixels)")
plt.legend()
plt.show()
```

This code performs a curve fitting of a sinusoidal model to a set of data. The data represents the distance of a moon from Jupiter over time, and a sinusoidal function is chosen because the motion of celestial bodies often follows periodic patterns, such as orbits, which can be modeled using sinusoidal functions.

The goal of the code is to determine the amplitude, angular frequency, and phase difference of the sine curve that best fits the provided data, and subsequently calculate the period of the oscillation.

1. **Code Description** The code imports three libraries:

   - `numpy`: For numerical operations and array handling.
   - `matplotlib.pyplot`: For plotting the data and the fitted curve.
   - `scipy.optimize.curve_fit`: For performing the curve fitting.

   The main steps of the code are as follows:

   (a) *Defining the Sine Function:* The function `sine_function(t, A, w, c)` models the periodic data with a sinusoidal function:

   $$y(t) = A\sin(wt + c)$$

   where:

   - A is the amplitude,
   - w is the angular frequency (related to the period),
   - c is the phase difference.

   (b) *Preparing Data:* The time data (`t_data`) and corresponding distance values (`y_data`) of the moon are provided.

   (c) *Initial Guesses:* Initial guesses for the amplitude (A), angular frequency (w), and phase difference (c) are made. The amplitude is set to half the difference between the maximum and minimum of the data, the frequency is estimated based on the time interval between data points, and the phase difference is initially set to zero.

   (d) *Curve Fitting:* The `curve_fit` function is used to fit the sine function to the data. It returns the optimal values of the parameters A, w, and c, along with their uncertainties.

   (e) *Period Calculation:* The period T of the oscillation is calculated from the angular frequency w using the formula:

   $$T = \frac{2\pi}{w}$$

   The uncertainties of the period are also computed using the propagation of error.

   (f) *Plotting:* Two plots are generated:

   - A scatter plot of the original data.
   - A plot of the data points along with the fitted sine curve.

2. **Results** The fitted sine curve is of the form:

   $$y(t) = A\sin(wt + c)$$

   where the parameters A, w, and c are determined by the curve fitting process. The period of the oscillation is derived from the angular frequency w, and its uncertainty is calculated.

References

- https://web.pa.msu.edu/people/horvatin/Astronomy_Facts/planet_pages/Jupiters_moons.htm[link to verify the actual time period of jupiter's gallilean moons]
- https://tejraj.com/product/startracker-150750-eq-reflector-telescope[Link to buy 6-inch telescope]
- https://www.amazon.in/CALANDIS-Eyepiece-Astronomical-Telescope-Accessories/dp/B0BQVN57VX?gQT=1[link to buy 10mm eyepiece]
- https://www.amazon.in/ELEPHANTBOAT%C2%AEMetal-Telescope-Spotting-Camera-Bracket/dp/B082B4DKMD?source=ps-sl-shoppingads-lpcontext&ref_=fplfs&psc=1&smid=A3LL2A01BH2M gQT=1[link to buy mobile holder]