ISSN No:-2456-2165

Mathematical Modeling for Planetary Motion using Python's PyGame Module, Matplotlib and Linux Shell Scripting

Aashni Joshi[1]
12th Grade - ISC Board Science Student from
Jamnabai Narsee School Mumbai
Maharashtra India

Mohan Kshirsagar[2]
Completed Bachelors in Electronics and
Telecommunication Engineering from
Sinhgad institute of Technology Lonavala,
Maharashtra, India.

Abstract:- In this research paper a model of the solar system for computing the orbits of all 8 planets true to its astronomical value and scale using python programming language is developed. The motion is visualised using Pygame and Matplotlib library. Iteration is utilized to depict and update the position of the planets during the process, and also develop a model to generate the date, time, and place of all solar and lunar eclipses from 2001 to 2100 by importing python modules related to astronomy. The model's validity and accuracy are demonstrated by the study and application of Kepler's Laws Of Planetary Motion throughout the procedure and modelling.

Keywords:- *Planetary Motion, Mathematical Modeling, Python Modules for Astronomy, Modelling of The Solar & Lunar Eclipses, Simulation of the Solar System, 3D plot for the projectile on a planet, Solar and Lunar eclipses from 2001 to 2100.*

I. INTRODUCTION

Astronomers are keen to follow the motions of the planets as they revolve around the Sun since the discovery of modern science. People could only see the star formations with their naked eyes before the invention of telescopes, so they can only see stars close to the Earth's orbit. Following that, the invention of telescopes made it possible for mankind to see distant planets clearly. These studies led to the development of Newton's law of universal gravitation and Kepler's laws, which offer mathematical explanations of planets' orbits.

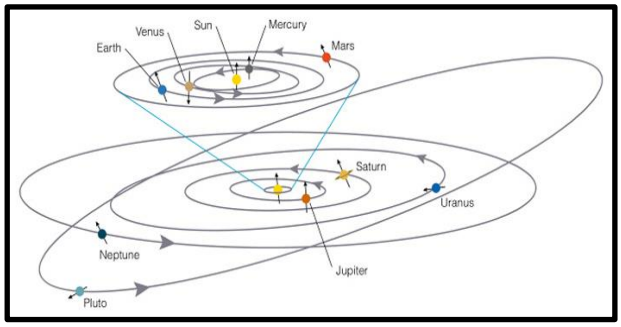

Fig 1:- Solar System

However, it is not practical to determine the movements of the majority of planets in the cosmos through analytical computation or direct observation. Observation through telescopes necessitates precise and expensive equipment as well as long periods of observation, and the complexity of the realistic system means that this problem has no analytical solutions. With the advancement of computational power over the last few decades, an increasing number of physicists have begun to use computational tools to solve equations, perform integrals, invert matrices, simulate various physical processes, and so on. This research paper describes how to simulate the motion of the Sun and eight planets using a computational method. In addition it also explains how to generate the date, time, and place of all solar and lunar eclipses from 2001 to 2100 using Python.

II. PROBLEM STATEMENT

During my research at IIT Bombay, NCAIR I realized there are various software packages that are built to aid physics learning and teaching and they have specific modules and solve a number of problems. There are specialized physics packages which consists of a training simulation software and instructional materials that help students understand and explore the fundamental laws of physics and mathematical methods of investigation of the movement of celestial bodies. In contrast, the mathematical modelling software allow users to build mathematical models of physical problems and obtain corresponding solutions. I also initially tried modelling the planetary motion using Maple Software. However, I later learnt the Python Computing language from scratch and decided to model the various motions on Python itself.

III. FINDINGS AND ANALYSIS

A. *Use of Python Programming:*

- Constructed a function that determines the next prime number after a number 'n' wherein the value of n is inputted by the user
- Calculated the root of a function from bisection method
- Researched and understood the working behind Newton Raphson Method of calculating the root and also implemented that in a python program

ISSN No:-2456-2165

- Researched about the time module in python and imported it for various programs to calculate the time of execution of the programs
- Created a virtual rock papers and scissors game by importing the random function from the math module.

B. Use of Linux Shell Scripting:

- The syntax of Linux and can now implement functions like pwd, cd, ls, cp, rm, echo, cat and the vi editor
- The basics of shell scripting and how it works. Eventually with the help of above computing languages, tools and software I was able to easily, quickly and with sufficient accuracy, model and animate the various cases of motion of material points and bodies under the action of gravity, inertia, engine thrust, and environmental resistance as follows.

C. Research, Analysis for Modeling of Solar and Lunar Eclipse:

- Analysed and understood the motion of the moon around the earth and the conditions necessary for solar and lunar eclipses to occur and thereby coded two programs that generate the date, time and place of all solar and lunar eclipses from 2001 to 2100 by importing python modules connected to astronomy.
- Created a program that will generate the length of the shadow of any object given the date, time, city and height of the object. The program generates the latitude and longitude simply by inputting the city by the user.
- In addition to the length of the shadow, the program also calculates and prints the angle the shadow makes with the geographic north i.e. the direction of the shadow.

D. Sundial Creation:

- Understood the working of the different types of sundials and created a sundial for Mumbai in accordance with its latitude. Manually constructed the dial as well as the pointer.
- Calculated and implemented the difference, in degrees, between every hour line and thereby designed the dial
- The dial is now put up at the terrace of the NCAIR-National Center for Aerospace Innovation and Research of IIT Bombay for viewers to observe.

E. Modeling and Animation of the solar system.

- Using the Pygame Module, created an animated program that depicts the motion of the first four planets around the sun.
- Not only was the mass of the sun considered in modelling the orbit, but also the average mass of other planets was factored in..
- Moreover, the orbits were modelled as elliptical and not simply circular. Distances of the planets from the sun were exactly to scale in the program.

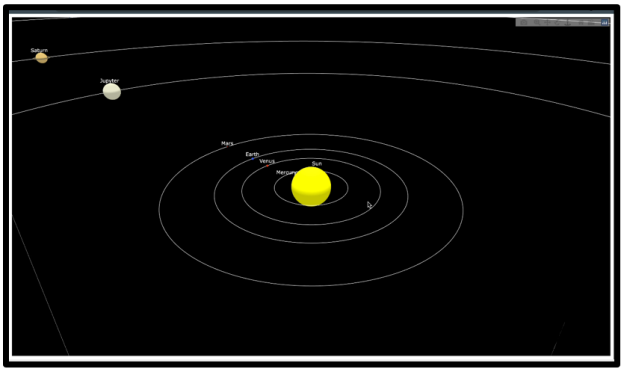

Fig 2:- Modeling and Animation of Solar System

- Successfully created a 3D model on Python of the Solar System with all the Planets around the Sun using Matplotlib. The size of the planets relative to each other was to scale with the exception of the sun. Distances of the planets from the sun were exactly to scale in the program.

F. Research and Analysis of Planets (Revolutions and Rotations):

- Described why the plane of revolution of all planets as well as the direction of revolution is the same
- Explained and Analysed three theories for the unusual rotation direction of Venus
- Identified the angle between earth's magnetic pole and spinning pole, between earth's rotation axis to revolution plane and between plane of revolution of earth and plane of revolution of moon
- Interpreted and explained the motion of moon around the earth regarding time of revolution as well as time of rotation and through a simulation, illustrated why we always face the same side of the moon

G. Recursion:

Understood the process of recursion through online content and constructed several programs to develop a greater understanding of it:

- Checking if a number is prime or not
- The power function — returns a^b for given values of a and b
- Generating the Fibonacci series
- Calculating factorial of any number
- Calculating the sum of all odd and even digits between two numbers
- Generating prime triplets
- Solving the Tower of Hanoi problem

H. Projectile Motion:

Derived equations relating to projectile motion on basis of its initial velocity, angle, time period, maximum height, and angle of projectile:

- Designed a program that plots the path of a projectile given its initial velocity and angle with the horizontal using matplotlib.

ISSN No:-2456-2165

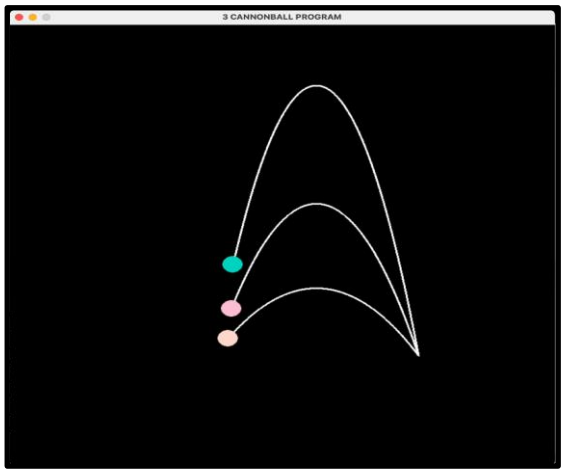

Fig 3:- Projectile Motion

- In addition, constructed two 3D plots of time vs velocity vs angle and range vs velocity vs theta for a given projectile.
- Also included an option for the user to select any planet in the solar system and thereby get the 3D plot for the projectile on that planet.
- Structured a program such that upon entering a Range, this program generates 3 different combinations of initial velocity and angle with the horizontal that give the same range.
- However, we think of it as a cannonball shooting shots. A total of 3 shots are fired and the time interval between these shots is inputted from the user. The time period for the first cannonball is also inputted from the user. Not only does this program accurately calculate values that deliver the same range, but it is also error corrected. If the time interval between two cannonballs is more than the time period of the first, the program will generate an error message and call the input function again.
- As a continuation of the program above, using the Pygame module, also created a program that traces the path of these three projectiles in an animated fashion with the particular time intervals factored in.

IV. PYTHON MODULES FOR MATHEMATICAL ANALYSIS

During this project I have worked on so many models that could give me clarity on how the entire research would work collaboratively. In my research there are some modules we have customized and used to track planetary motion.

A. Predicting all Lunar Eclipses :

The Python program that calculates the date and time for all the lunar eclipses between 2001 and 2100.

To make this scenario work we have used ephem library which is freely available. Since its first release in 1998, PyEphem has given Python programmers the ability to compute planet, comet, asteroid, and Earth satellite positions. It wraps the “libastro” C library that powers the XEphem astronomy application for UNIX — whose author Elwood Downey generously gave permission for PyEphem to use his library with Python. PyEphem can also compute the angular separation between two objects in the sky, determine the constellation in which an object lies, and find the times an object rises, transits, and sets.

```python
# Importing the required libraries
import ephem
from datetime import datetime, timedelta

# Setting start and end time
curtime = datetime(2001, 1, 1, 0, 0, 0)
endtime = datetime(2100, 12, 31, 23, 59, 59)


moon = ephem.Moon()
sun = ephem.Sun()
observer = ephem.Observer()

# Setting the observer at the centre of the Earth
observer.elevation = -6371000

# Setting pressure as 0 to avoid any effects due to pressure
observer.pressure = 0


while curtime <= endtime:

    observer.date = curtime
    moon.compute(observer)
    sun.compute(observer)

    # A lunar eclipse occurs
    sep = abs((float(ephem.separation(moon, sun))
        / 0.01745329252) - 180)


    if sep < 0.9:
        print(curtime.strftime('%Y/%m/%d %H:%M:%S'), sep)
        curtime += timedelta(days = 1)
    else:
        curtime += timedelta(hours = 1)
```

Fig 4:- Predicting All Lunar Eclipse

B. Recursion to calculate Factorial:

Using the concept of recursion to calculate the factorial of a number. The goal of this program is to further understand how recursion works.

```python
#Using the concept of recursion to calculate the factorial of a number
#The goal of this program is to further understand how recursion works


#Defining the function
def factorial(n):
    #Base
    if n==0:
        return 1
    else:
        return n*factorial(n-1)   #Calling the function inside the function


#To input the number from the user
n=int(input(print("enter a number ")))
a=factorial(n)
print("factorial=",a)
```

Fig 5:- Recursion to Calculate Factorial

In computer science, recursion is a method of solving a computational problem where the solution depends on solutions to smaller instances of the same problem. Recursion solves such recursive problems by using functions that call themselves from within their own code. The approach can be applied to many types of problems, and recursion is one of the central ideas of computer science.

C. Recursion to generate Fibonacci Series:

Generating the first 'n' terms of the Fibonacci series using the process of Recursion. The goal of this program is to further comprehend the complex process of Recursion in python.

ISSN No:-2456-2165

```
#Generating the first 'n' terms of the Fibonacci series using the process of Recursion.
#The goal of this program is to further comprehend the complex process of Recursion in python.

print("Fibonacci series")
#Inputting value of 'n'
print("Please enter the number of terms you wish to generate. ")
n=input()
n=int(n)

#Defining a function to calculate the nth term of the Fibonacci series
def fiboterm(x):
    if x==1:
        return 0
    elif x==2:
        return 1
    else:
        #Applying recursion by calling the same function inside the function
        z= fiboterm(x-2)+fiboterm(x-1)
        return z

#Defining the function to generate the series uptil 'n' terms
def fibonacci(n):

    if n==3:
        a=0
        b=1
        c=a+b
        print(a)
        print(b)
        print(c)


    else:
        d=fiboterm(n-2)
        e=fiboterm(n-1)
        f=d+e
        #Recursion
        print(fibonacci(n-1),f)


fibonacci(n)
```

Fig 6:- Recursion to generate Fibonacci Series

In computer science, recursion is a method of solving a computational problem where the solution depends on solutions to smaller instances of the same problem. Recursion solves such recursive problems by using functions that call themselves from within their own code. The approach can be applied to many types of problems, and recursion is one of the central ideas of computer science

D. Recursion for Power Function:

Using recursion to calculate the value of a number 'a' raised to another number 'b'. This program was designed to understand Recursion better.

```
#Using recursion to calculate the value of a number 'a' raised to another number 'b'
#This program was designed to understand Recursion better

print("Please enter values for 'a' and 'b' to obtain a raised to the power of b")

#Inputting values of a and b
print("enter a ")
a=int(input())
b=print("enter b ")
b=int(input())

#Defining power function
def power(a,b):
    #Base
    if b==1:
        return a
    else:
        #Calling function again so that it recurses
        return a*power(a,b-1)


#Calling function for a and b
c=power(a,b)

#Printing final answer
print ("a^b is",c)
```

Fig 7:-Recursion for Power Function

In computer science, recursion is a method of solving a computational problem where the solution depends on solutions to smaller instances of the same problem. Recursion solves such recursive problems by using functions that call themselves from within their own code. The approach can be applied to many types of problems, and recursion is one of the central ideas of computer science

E. Checking for Prime Number using Recursion:

Using recursion to see if a given number is a prime number or not. The goal of this program is to further comprehend the complex process of Recursion in python.

```
#Using recursion to see if a given number is a prime number or not
#The goal of this program is to further comprehend the complex process of Recursion in python

#Inputting the number
print("Please enter the number you want to check")
n=int(input())

#'k' acts an indicator
k=0
i=2

#Defining the function
def isPrime(n,i):
   global k
   if i<n:
       if n%i==0:
           k+=1     #Incrementing 'k' to indicate that it's not a prime number
           return k
       else:
          #Calling the function inside the same function in order to recurse
           isPrime(n,i+1)

isPrime(n,i)
if k==0:
    print("prime!")
else:
    print("not prime:(")
```

Fig 8:- Checking for Primary Number using Recursion

In computer science, recursion is a method of solving a computational problem where the solution depends on solutions to smaller instances of the same problem. Recursion solves such recursive problems by using functions that call themselves from within their own code. The approach can be applied to many types of problems, and recursion is one of the central ideas of computer science.

F. Calculating-Root-by-Newton-Raphson Method:

In numerical analysis, Newton's method, also known as the Newton–Raphson method, named after Isaac Newton and Joseph Raphson, is a root-finding algorithm which produces successively better approximations to the roots (or zeroes) of a real-valued function. The most basic version starts with a single-variable function f defined for a real variable x, the function's derivative f', and an initial guess x0 for a root of f.

```
# Python3 code for implementation of Newton
# Raphson Method for solving equations

# An example function whose solution
# is determined using Bisection Method.
# The function is x^3 - x^2 + 2
def func( x ):
        return x * x * x - x * x + 2

# Derivative of the above function
# which is 3*x^x - 2*x
def derivFunc( x ):
        return 3 * x * x - 2 * x

# Function to find the root
def newtonRaphson( x ):
        h = func(x) / derivFunc(x)
        while abs(h) >= 0.0001:
                h = func(x)/derivFunc(x)

                # x(i+1) = x(i) - f(x) / f'(x)
                x = x - h

        print("The value of the root is : ",x)

# Driver program to test above
x0 = -20 # Initial values assumed
newtonRaphson(x0)
```

Fig 9:- Calculating root-by-newton’s 67aphson method

This program is designed to calculate the root of a given function using Newton Raphson method.

ISSN No:-2456-2165

G. Calculating time taken for a Newton Raphson Method program to execute:

This program calculates the time at the start of the program and at the end of it. It then subtracts both to return the time taken for the program to the run. The program taken is the program for calculating the root of a given function by Newton Raphson method.

```
#Calculating time for Newton Raphson Method


from sympy import *
import time

#Inputting a function from the user
print("Please enter the function ")
funcc=input()
t1=time.time()

#Evaluating the function
def f(x):
    return eval(funcc)

#Defining a function that returns the derivative of the function
def derivf(x):
     h=0.00000000001
     return (f(x+h)-f(x))/h


#Setting the guess value as 2
x=2

for i in range (1,1000):              #Letting the program run for not more than 1000 times
        xnew=x-f(x)/derivf(x)         #Newton Raphson formula
        x=xnew

#Printing the root
print("The root was found to be ",x)
t2=time.time()
print("Time taken= ",t2-t1)
```

Fig 10:- Newton Raphson Method

In numerical analysis, Newton's method, also known as the Newton–Raphson method, named after Isaac Newton and Joseph Raphson, is a root-finding algorithm which produces successively better approximations to the roots (or zeroes) of a real-valued function. The most basic version starts with a single-variable function f defined for a real variable x, the function's derivative f', and an initial guess x0 for a root of f.

H. 2D Plotting:

A program designed to plot the graph for three different functions on the same plot. The functions are: 1) sin(x) 2) sin(x)+cos(x) 3) sin(x)*cos(x).

```
# -*- coding: utf-8 -*-
"""Untitled2.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1mlFZQ-640v70OWctJYiaXMwLkf5bN6qP
"""

from matplotlib import pyplot as plt
import numpy as np
import math

x=np.arange(-2*np.pi,2*np.pi,0.01)
plt. plot(x,np.sin(x))
plt.plot(x,np.sin(x)*np.cos(x)*2)
plt.plot(x,np.sin(x)/math.sqrt(2)+np.cos(x)/math.sqrt(2))
plt.show()
```

Fig 11:- 2D data plotting using matplotlib python library

The three functions were plotted by importing the matplotlib library.

I. Calculating time taken for a Bisection Method program to execute:

This program calculates the time at the start of the program and at the end of it. It then subtracts both to return the time taken for the program to the run. The program taken is the program for calculating the root of a given function by the Bisection method.

In mathematics, the bisection method is a root-finding method that applies to any continuous function for which one knows two values with opposite signs. The method consists of repeatedly bisecting the interval defined by these values and then selecting the subinterval in which the function changes sign, and therefore must contain a root. It is a very simple and robust method, but it is also relatively slow. Because of this, it is often used to obtain a rough approximation to a solution which is then used as a starting point for more rapidly converging methods.

```
#Calculating time taken for a Bisection Method program

from matplotlib import pyplot as plt
import numpy as np
import math

import time

#Inputting
print("Enter function ")
funcc=input()

#Recording Time at the start of the program
t1=time.time()

def f(x):
    return eval(funcc)

def bisection(a,b,tol):
    global c
    x1 = a
    x2 = b
    i=0

    while (np.abs(f(x1)-f(x2))>=tol):
      i+=1

      #Setting a limit for the number of times the program runs
      if i==1000:
        print("roots not found")
        break

      c=(x1+x2)/2.0   #Making 'c' the average of two x values

      prod=f(x1)*f(c)

      if prod > tol:
         x1=c
      else:
         if prod < tol:
           x2=c
    return c

c=bisection(-100,100,0.00000000000001)  #Calling the function

print("The Root of the equation is ",c)
t2=time.time()
print("Time taken for execution is ",t2-t1)
```

Fig 12:- Bisection Method

J. Prime-Number-Calculator:

This program consists of two different programs that calculate the prime number that is closest to and is less than a given number ‘n’. ‘n’ is inputted from the user. The first program checks for non-prime numbers until the number n itself whereas the second one checks until the square root of the number. We then compare the time taken for both programs to execute.

ISSN No:-2456-2165

```
import math


#@title Prime Number calculator { run: "auto" }
n = 10000000 #@param {type:"slider", min:2, max:10000000, step:1}

import time
#n=input(print("please enter a number "))
#n=int(n)
t1 = time.time()

print("Prime number is:")

for num in range(n, 2,-1):

   if num > 1:
       for i in range(2, num):
           if (num % i) == 0:
               break
       else:
           print(num)
           break

t2= time.time()

print("time taken =" + str(t2-t1))
```

```
Prime numbers are:
9999991
time taken =1.240405559539795
```

```
#@title Prime Number calculator { run: "auto" }
n = 10000000 #@param {type:"slider", min:2, max:10000000, step:1}
import time
import math
#n=input(print("please enter a number "))
#n=int(n)
t1 = time.time()
print("Prime number is:")

for num in range(n,2,-1):

   if num > 1:
       for i in range(2, math.ceil(math.sqrt(num))):
           if (num % i) == 0:
               break
       else:
           print(num)
           break

t2=time.time()

print("time taken =" + str(t2-t1))
```

```
Prime numbers are:
9999991
time taken =0.0036499500274658203
```

Fig 13:- Prime Number calculator

V. RESULTS AND CONCLUSION

Simulated the motion of Planets of the Solar System: Using the Pygame Module, simulated the motion of the first four planets around the sun.

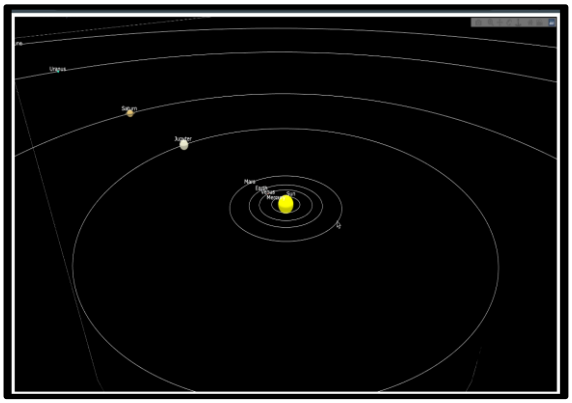

Fig 14:- 3D Modeling of Solar System

Constructed a 3D model of the Solar System that shows the orbits of all 8 planets true to its astronomical value scale using Python programming language.

Not only was the mass of the sun considered in modelling the orbit, but also the mass of other planets was factored in as an average.

Moreover, the orbits were modelled as elliptical and not simply circular. Distances of the planets from the sun were exactly to scale in the program.

In addition, created a program that traced Projectile Motion in an animated fashion. It also included an option for the user to select any planet in the solar system and thereby get the 3D plot for the projectile on that planet.

Simulation of The Solar & Lunar Eclipses: Developed two programs that generate the date, time, and place of all solar and lunar eclipses from 2001 to 2100 by importing python modules related to astronomy.

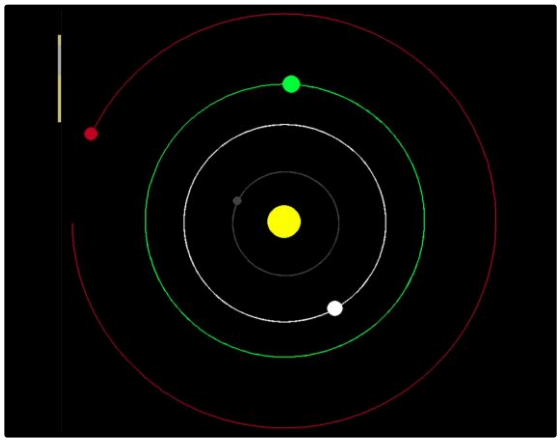

Fig 15:- Solar & Lunar Eclipse Projection

Created a program that will generate the length of the shadow of any object given the date, time, city and length of the object. The program generates the latitude and longitude simply by inputting the city from the user.

Sundial Creation: Manually created a sundial for Mumbai in accordance with its latitude. Manually constructed the sundial as well as the pointer.

Calculated and implemented the difference, in degrees, between every hour line and thereby designed the Sundial. The dial is now put up at the terrace of the NCAIR - National Center for Aerospace Innovation and Research of IIT Bombay for viewers to observe.

ACKNOWLEDGMENT

Firstly we would like to show my deep appreciation to my parents for their outstanding support and motivation who also helped me through the research work journey.

Also I am immensely grateful to Dr. Asim Tewari - Chair Professor Department of Mechanical Engineering of IIT Bombay for his mentorship and guidance including concept clarification, comments and reviews during the entire research and project development process.

REFERENCES

[1]. Krasinsky, G. A.; Pitjeva, E. V.; Vasilyev, M. V.; Yagudina, E. I. (July 2002). "Hidden Mass in the Asteroid Belt". Icarus. 158 (1): 98–105. Bibcode:2002Icar..158...98K.doi:10.1006/icar.2002.6837.
[2]. "The Sun's Vital Statistics". Stanford Solar Center. Archived from the original on 14 October 2012. Retrieved 29 July 2008., citing Eddy, J. (1979). A New Sun: The Solar Results From Skylab. NASA. p. 37. NASA SP-402.
[3]. Williams, David R. (7 September 2006). "Saturn Fact Sheet". NASA.
[4]. Williams, David R. (23 December 2021). "Jupiter Fact Sheet". NASA Goddard Space Flight Center.
[5]. Weissman, Paul Robert; Johnson, Torrence V. (2007). Encyclopedia of the solar system. Academic Press. p. 615. ISBN 978-0-12-088589-3.
[6]. Podolak, M.; Weizman, A.; Marley, M. (December 1995). "Comparative models of Uranus and Neptune". Planetary and Space Science. 43 (12): 1517–1522. Bibcode:1995P&SS...43.1517P.doi:10.1016/0032-0633(95)00061-5.
[7]. Podolak, M.; Podolak, J. I.; Marley, M. S. (February 2000). "Further investigations of random models of Uranus and Neptune". Planetary and Space Science. 48 (2–3): 143–151. Bibcode:2000P&SS...48..143P. doi:10.1016/S0032-0633(99)00088-4.
[8]. Zellik, Michael (2002). Astronomy: The Evolving Universe (9th ed.). Cambridge University Press. p. 240. ISBN 978-0-521-80090-7. OCLC 223304585.
[9]. Placxo, Kevin W.; Gross, Michael (2006). Astrobiology: a brief introduction. JHU Press. p. 66. ISBN 978-0-8018-8367-5.
[10]. Standish, E. M. (April 2005). Kurtz, D. W. (ed.). "The Astronomical Unit now". Transits of Venus: New Views of the Solar System and Galaxy, Proceedings of IAU Colloquium #196, Held 7–11 June 2004 in Preston, U.K. Cambridge: Cambridge University Press. 2004: 163–179. Bibcode:2005tvnv.conf..163S. doi:10.1017/S1743921305001365. S2CID 55944238.
[11]. Emilio, Marcelo; Kuhn, Jeff R.; Bush, Rock I.; Scholl, Isabelle F. (2012). "Measuring the Solar Radius from Space during the 2003 and 2006 Mercury Transits". The Astrophysical Journal. 750 (2): 135. arXiv:1203.4898. Bibcode:2012ApJ...750..135E. doi:10.1088/0004-637X/750/2/135. S2CID 119255559.
[12]. Williams, David R. (23 December 2021). "Neptune Fact Sheet". NASA Goddard Space Flight Center.
[13]. Jaki, Stanley L. (1 July 1972). "The Early History of the Titius-Bode Law". American Journal of Physics. 40 (7): 1014–1023. Bibcode:1972AmJPh..40.1014J. doi:10.1119/1.1986734. ISSN 0002-9505.
[14]. Phillips, J. P. (1965). "Kepler's Echinus". Isis. 56 (2): 196–200. doi:10.1086/349957. ISSN 0021-1753. JSTOR 227915. S2CID 145268784.
[15]. Boss, Alan (October 2006). "Is it a coincidence that most of the planets fall within the Titius-Bode law's boundaries?". Astronomy. Ask Astro. Vol. 30, no. 10. p. 70.