SHIV NADAR UNIVERSITY

Department of Physics

School of Natural Sciences

PHY 106 PROJECT

BINARY STAR SYSTEM AND ITS ORBITAL ELEMENTS

Aksaj Bharatwaj (2010110059)

Bhavika Yadav (2010110070)

Siddharth M (2010110914)

Sonal Garg (2010110641)

ABSTRACT

Our Python code calculates the orbital elements for a binary star system based on the observational data collected. The equations used are the result of combining Kepler's laws, the definitions of the orbital elements and the naturally occurring symmetries and rules of a stable binary system.

Introduction: Two stars that are gravitationally bound are said to orbit their common center of mass. In astronomy, a binary system is one that consists of two stars that are gravitationally bound.

When observing binary systems, we can only attain a few measurable values. Using techniques in astrometry and photometry, by observing the light from the individual objects, estimate values of their masses can be obtained. After locating the center of each object, the angular separation between them, the angle between the line connecting them and vertical from the center of the primary can be measured; these are the Separation and Position Angles of the system at the time of the observation, respectively.

THEORY

The binary star system consists of a lot of orbital elements which are absolutely necessary to describe the true motion and characteristics of the binary system.

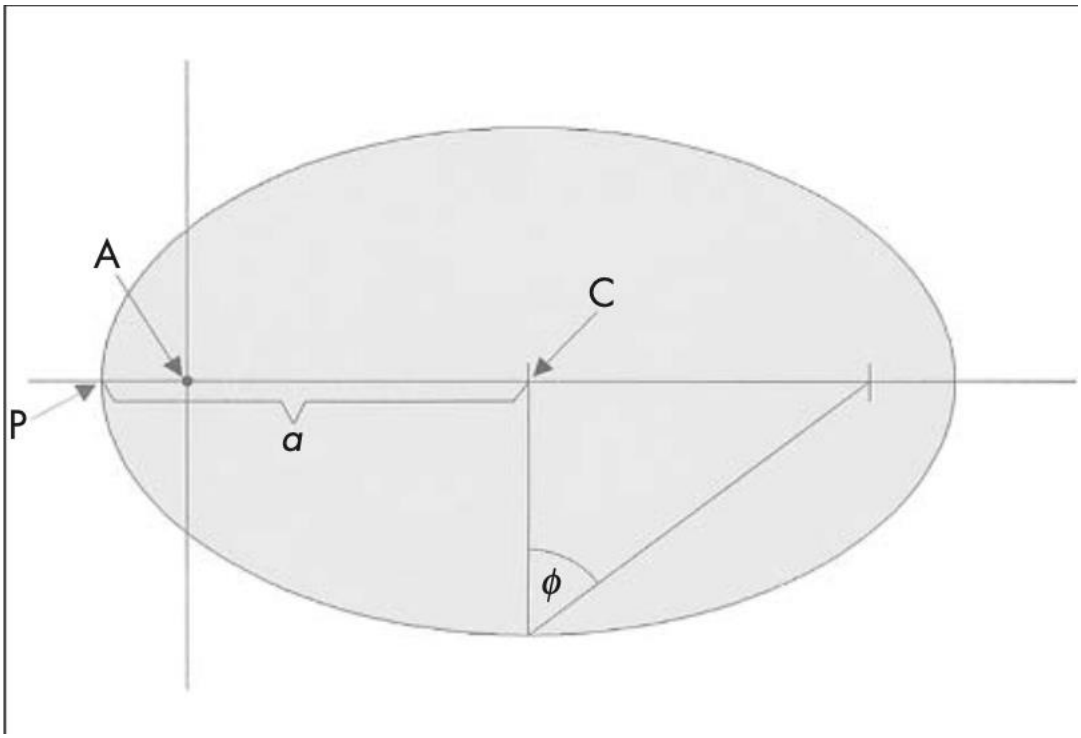
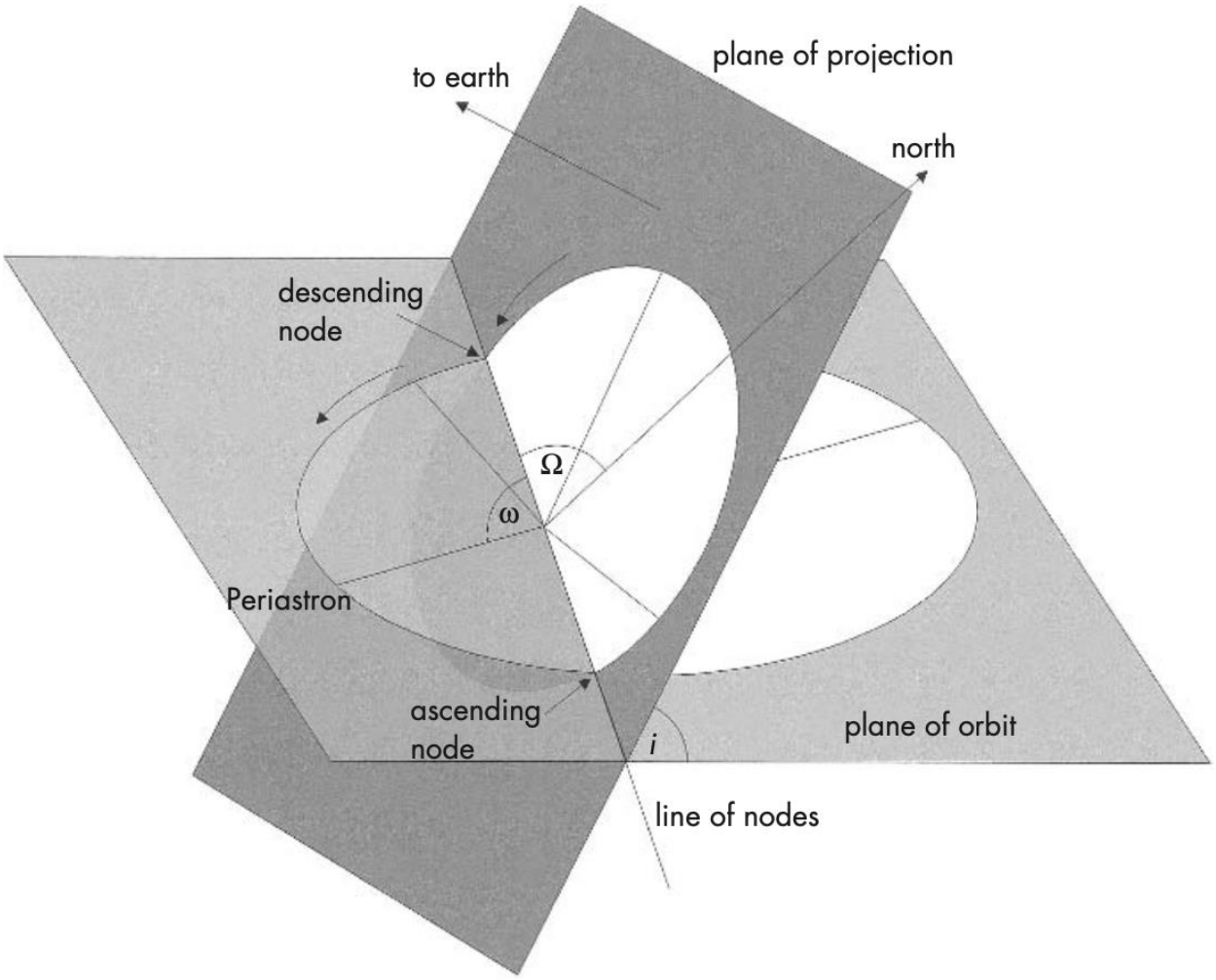


Fig: THE TRUE ELEMENTS OF A VISUAL BINARY STAR

Accordingly, the periastron P is the closest point of the ellipse to A and the elements of the real orbit are:

- P: The revolution period (in years), often described in terms of mean motion per year ($n = \frac{2\times\pi}{P}$)
- T: Time passage through Periastron
- e: The numerical eccentricity e of the orbit (the auxiliary angle e is given by $e = sin\text{ø}$).
- a: The semi-major axis of the binary system in arcseconds.

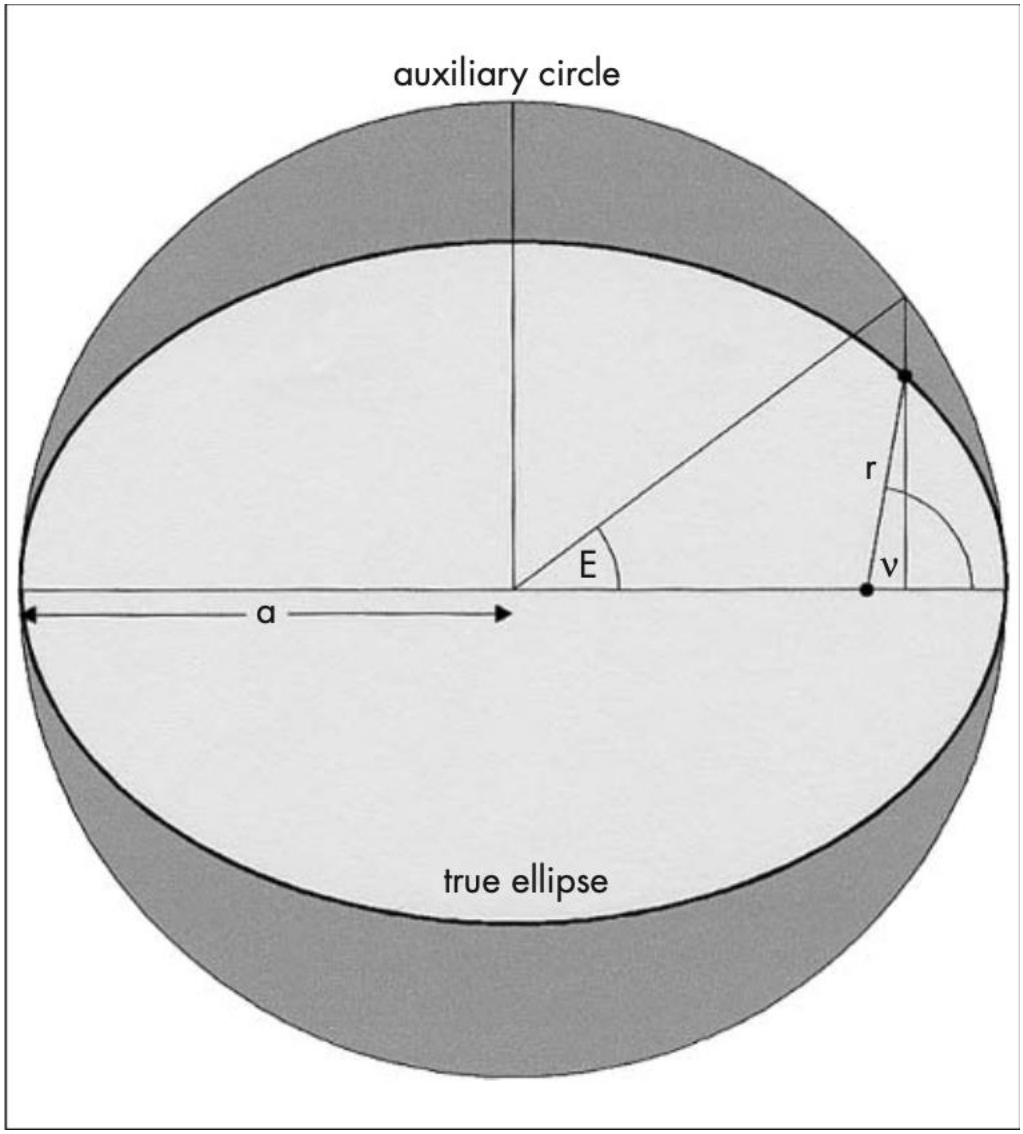
The apparent (observed) orbit results from a projection of the true orbit onto the celestial sphere (as shown below.)



This projection is determined by:

- **Projection Angle of the Ascending Node (Ω):** This is the position angle of the line of intersection between the plane of projection and the true orbital plane. The angle is counted from North to the line of nodes. The ascending node is the node where the motion of the companion is directed away from the Sun. It differs from the second node by 180° and can only be determined by the radial-velocity measurements.
- **Orbital Inclination (*i*):** This is the angle between the plane of projection and the true orbital plane. It ranges from $0° \leq i \leq 180°$. For $0° \leq i \leq 90°$, the motion is called direct. The companion then moves in the direction of increasing position angles (anticlockwise). For $90° \leq i \leq 180°$, the motion is called retrograde.

- **Argument of Periastron (ω):** This is the angle between the node and the periastron, measured in the plane of true orbit and in the direction of motion of the companion.



The angle E is called the eccentric anomaly and has to be determined from the mean anomaly, M:

$n(t - T) = M = E - \sin(E)$: Kepler's Equation

This equation is transcendental, i.e., it is not algebraic and has to be solved iteratively. The approximation, thus, is given by:

$$E_o = M + e\sin(M) + \frac{e^2}{M}\sin(2M)$$

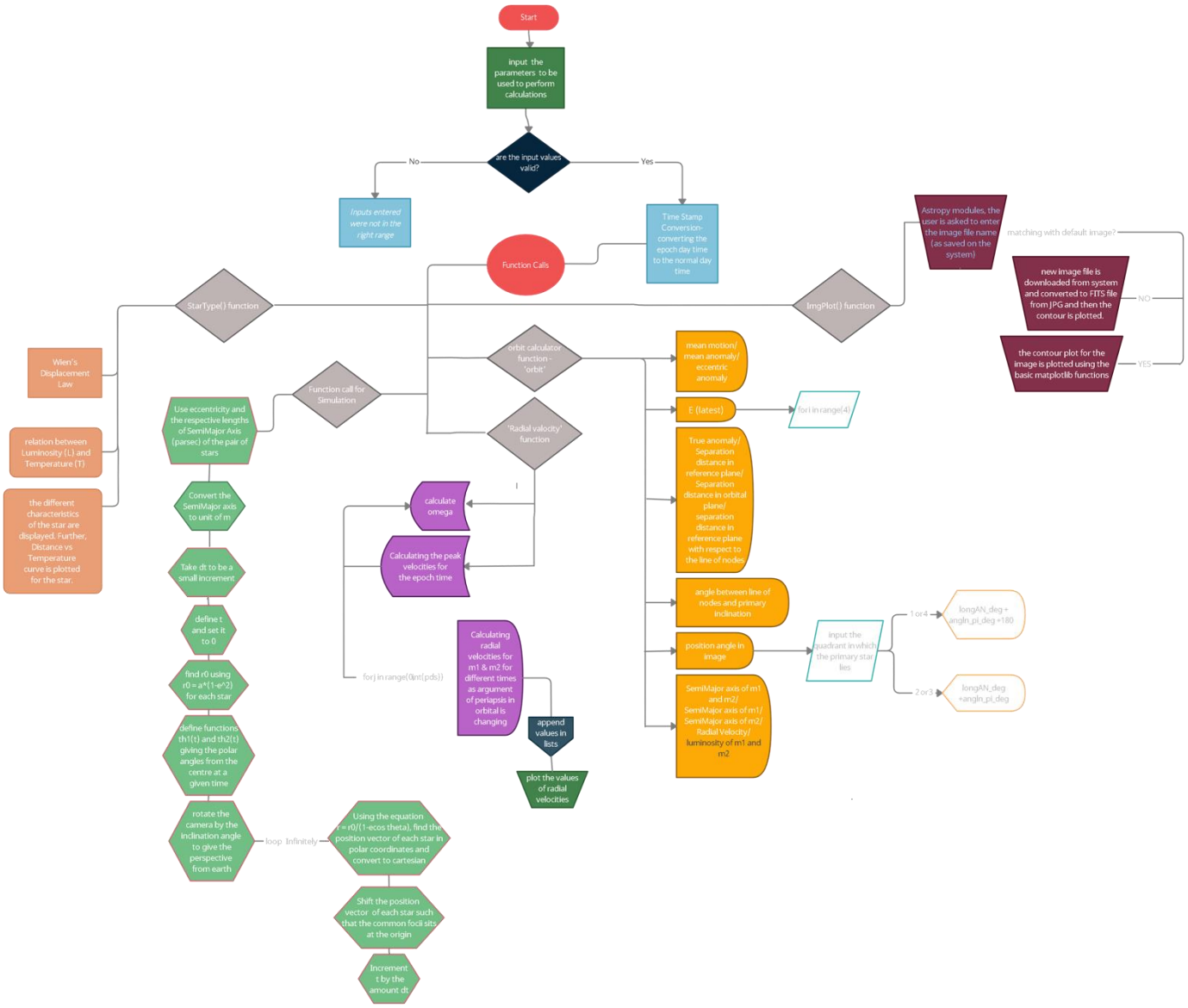
This new E_o is used to calculate new M_o:

$$M_o = E_o + \frac{M - M_o}{1 - e\ cosE_o}$$

These formulae are iterated to the desired accuracy. Four iterations are sufficient for $e \leq 0.95$.

The desired positions are further calculated using the above diagrams and formulating the equations as mentioned ahead.

FLOW CHART


Start
input the parameters to be used to perform calculations
are the input values valid?
No
Yes
Inputs entered were not in the right range
Time Stamp Conversion- converting the epoch day time to the normal day time
Function Calls
StarType( ) function
ImgPlot( ) function
Astropy modules, the user is asked to enter the image file name (as saved on the system)
matching with default image?
new image file is downloaded from system and converted to FITS file from JPG and then the contour is plotted.
NO
the contour plot for the image is plotted using the basic matplotlib functions
YES
Wien's Displacement Law
relation between Luminosity (L) and Temperature (T)
the different characteristics of the star are displayed. Further, Distance vs Temperature curve is plotted for the star.
Function call for Simulation
orbit calculator function - 'orbit'
'Radial valocity' function
Use eccentricity and the respective lengths of SemiMajor Axis (parsec) of the pair of stars
Convert the SemiMajor axis to unit of m
Take dt to be a small increment
define t and set it to 0
find r0 using r0 = a*(1-e^2) for each star
define functions th1(t) and th2(t) giving the polar angles from the centre at a given time
rotate the camera by the inclination angle to give the perspective from earth
loop Infinitely
Using the equation r = r0/(1-ecos theta), find the position vector of each star in polar coordinates and convert to cartesian
Shift the position vector of each star such that the common focii sits at the origin
Increment t by the amount dt
calculate omega
Calculating the peak velocities for the epoch time
for j in range(0int(pds))
Calculating radial velocities for m1 & m2 for different times as argument of periapsis in orbital is changing
append values in lists
plot the values of radial velocities
mean motion/ mean anomaly/ eccentric anomaly
E (latest)
for i in range(4)
True anomaly/ Separation distance in reference plane/ Separation distance in orbital plane/ separation distance in reference plane with respect to the line of nodes
angle between line of nodes and primary inclination
position angle in image
input the quadrant in which the primary star lies
1 or 4
longAN_deg + angIn_pi_deg +180
2 or 3
longAN_deg +angIn_pi_deg
SemiMajor axis of m1 and m2/ SemiMajor axis of m1/ SemiMajor axis of m2/ Radial Velocity/ luminosity of m1 and m2

CALCULATIONS

The following equations are the result of combining Kepler's laws, the definitions of the orbital elements and the naturally occurring symmetries and rules of a stable binary system.

First, calculate the Mean Motion (n) from the provided orbital period (P) of the binary:

$$n = \frac{2\pi}{P} \quad \text{----------(1)}$$

```
#mean motion
n=2*pi/Pds
```

Use the Mean Motion, time of current epoch (t) and time of the last periapsis (T) to obtain the Mean Anomaly (M):

$$M = n \times (t - T) \quad \text{----------(2)}$$

```
#mean anomaly
M_deg=n*(t-T)
```

For the elliptical orbits, the eccentricity (e) depends on the angular momentum, circular orbits have the maximum angular momentum for a given energy.

$$e = M + e \times sin(M) + \frac{e^2}{2M} \times \sin(2M) \quad \text{----------(3)}$$

```
#eccentric anomaly
e_last=M_deg+ e*sin(M_deg)+ (e**2/(2*M_deg))*sin(2*M_deg)
```

Applying repetitive Iteration to calculate Latest Eccentric Anomaly (Elatest):

$$Elatest = Elast - \frac{[M - Elast + e \times \sin(Elast)]}{[e \times \cos(Elast) - 1.0]} \quad \text{----------(4)}$$

```
e_latest_last=e_last +((M_deg-M0_deg)/(1- e*cos(e_last)))
e_last=e_latest_last
```

The loop completes when E_{latest} and E_{last} are the same to 10 decimal places. Use the resultant E_{latest} to calculate the True Anomaly (TA) of the binary system:

$$\mathrm{TA} = \frac{[\cos^{-1}(Elatest) - e]}{[1.0 - e \times \cos(Elatest)]} \quad \text{----------(5)}$$

```
#True Anomaly
TA=acos((cos(e_latest_last)-e)/(1-e*cos(e_latest_last)))
```

Calculate the Separation Distance in the reference plane from the provided Separation Angle and System Distance:

$$Seperation_{Distance_{ReferencePlane}} = SeperationAngle_{measured} \times SystemDistance \text{ ----------(6)}$$

```
#seperation distance in reference plane from provided seperation distance and seperation angle AU
sep_dis_rp=ang_arcsec*sys_dys
```

Take Inclination into account to find the Separation Distance in the orbital plane:

$$Seperation_{Distance_{OrbitalPlane}} = \frac{Seperation_{Distance_{ReferencePlane}}}{\{[sin(\omega + TA) \times cos(i)]^2 + [\cos(\omega + TA)]^2\}^{\frac{1}{2}}} \quad \text{----------(7)}$$

Where, ω = Argument of Periapsis (°)

```
#seperation distance in orbital plane
sep_dis_op=sep_dis_rp/(sqrt(((sin(argperi_deg+TA)*(((cos(complex(0,1)).real))**2)
                            +(cos((argperi_deg+TA))**2))))).real
```

Use the True Anomaly and Separation Distance in the orbital plane to find the X and Y components of the Separation Distance in the reference plane (WRT the Line of Nodes(Ω)):

$$Seperation_{Distance_Y} = Seperation_{Distance_{OrbitalPlane}} \times \sin(\omega + TA) \times \cos(i) \rightarrow (8a)$$

$$Seperation_{Distance_X} = Seperation_{Distance_{OrbitalPlane}} \times \cos(\omega + TA) \rightarrow (8b)$$

```
#seperation distance in reference plane with respect to the line of nodes
sep_dis_rp_y=sep_dis_op*sin(argperi_deg+TA)*(cos(complex(0,1)).real).real
sep_dis_rp_x=sep_dis_op*cos(argperi_deg+TA)
```

Now to find the angle between the Line of Nodes and the primary, correct for inclination:

$$Angle_{AN}toM1_{corr} = \tan^{-1}\left(\frac{Seperation_{Distance_Y}}{Seperation_{Distance_X}}\right) \quad \text{----------(9)}$$

```
#Angle between the line of nodes and primary inclination
angln_pi_deg=atan(sep_dis_rp_y/sep_dis_rp_x)
```

The final measured Position Angle depends on which quadrant M1 lies in on an X (Line of Nodes, positive to right in reference plane) Y (⊥ to Line of Nodes, positive in upwards direction in reference plane) grid. Four quadrant options result in two final equations:

If M_1 lies in 1st or 4th Quadrant:

$$Position_{Angle} = Long_{AN} + Angle_{AN}toM1_{corr} + 180° \quad \text{----------(9a)}$$

If M_2 lies in 2nd or 3rd Quadrant:

$$Position_{Angle} = Long_{AN} + Angle_{AN}toM1_{corr} \quad \text{----------(9b)}$$

```
#Position angle in image
quad=int(input("Enter the quadrant in which primary star(m1) Lies => "))
if quad==1 or quad==4:
    pos_angle=longAN_deg+angln_pi_deg +180
else:
    pos_angle=longAN_deg+angln_pi_deg
```

Using the Separation Distance in the orbital plane, Eccentricity and True Anomaly to calculate the total of the Semi-major axis (a1+a2):

$$a_{total} = Seperation_{Distance_{OrbitalPlane}} \times \frac{1+e\times\cos(TA)}{1-e^2} \quad \text{----------(10)}$$

```
#Sum of semimajor aces of m1 and m2
a_total=sep_dis_op*(1+e*cos(TA))/(1-e**2)
```

Finally, if the system is a binary star system (indicated by setting both M1 and M2 to something other than 1), the individual Semi-major axis can be found from the mass ratio and the total of semimajor axis:

$$a_1 = \frac{a_{total}}{(1+(\frac{m_2}{m_1}))} \quad \text{----------(11a)}$$

$$a_2 = \frac{a_1\times m_1}{m_2} \quad \text{----------(11b)}$$

```
#semimajor axis of m1
a1=a_total/(1+(m2/m1))
#semimajor axis of m2
a2=(a1/m2)*m1
```

The code is further used to calculate the peak flux-weighted radial velocity of the stars during an eclipse.

$$v_1 = \frac{[m_2 \times \sin^3(inc_{angle}) \times 2\pi G]}{[(m_1+m_2)^2 \times P]^{\frac{1}{3}}} \quad \text{----------(12a)}$$

$$v_2 = \frac{[m_1 \times \sin^3(inc_{angle}) \times 2\pi G]}{[(m_1+m_2)^2 \times P]^{\frac{1}{3}}} \quad \text{----------(12b)}$$

```
#Radial Velocity
v1r=((((m2*sin(inc_deg))**3)*2*pi*4.3009e-3)/(((m1+m2)**2)*Pds))**(1/3)
v2r=((((m1*sin(inc_deg)**3)*2*pi*4.3009e-3)/(((m1+m2)**2)*Pds)))**(1/3)
```

The most important physical property of stars that determines everything else is Mass-luminosity relation.

$$L_1 = (m_1)^{\alpha_1} \times 3.828 \times 10^{26} \text{ ----------(13a)}$$

$$L_2 = (m_2)^{\alpha_2} \times 3.828 \times 10^{26} \text{ ----------(13b)}$$

$L \sim M^4$ (A key relation for the understanding of stellar physics • Main Sequence stars follow this relation, but giants, supergiants, & white dwarfs do not)

```
#Luminosity
L1=((m1)**alpha)*3.828*1e26
L2=((m2)**alpha)*3.828*1e26
```

FIGURES:

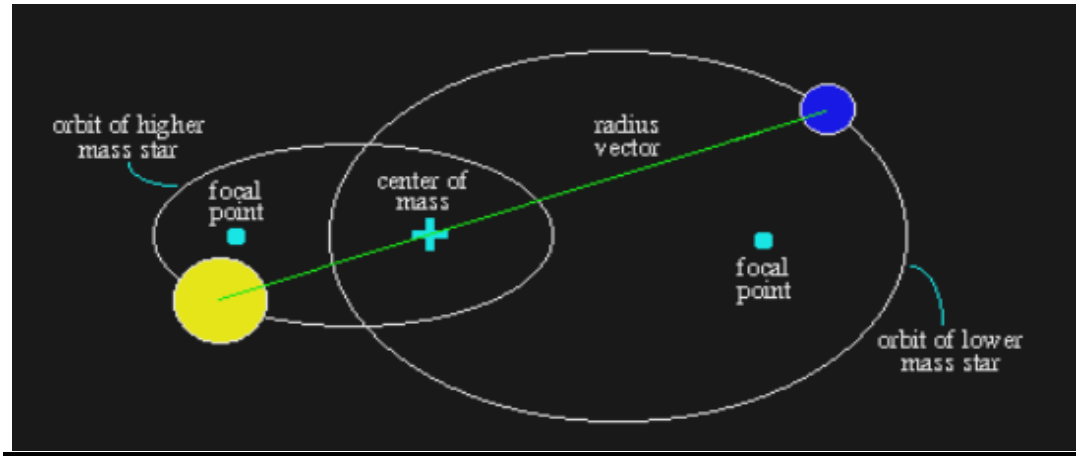


Fig. 1.- Orbits of binary systems.

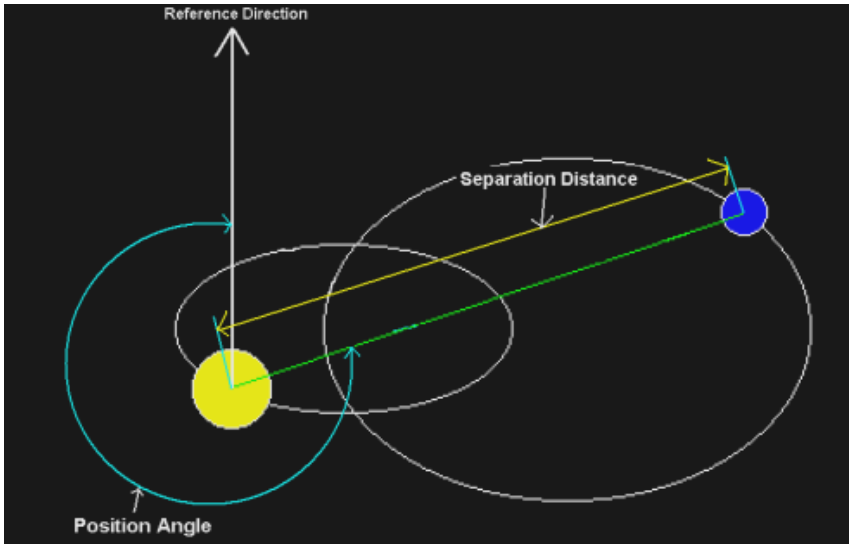


Fig. 2.- Diagram of Binary Star System and its separation and position angles.

PROGRAM CODE

1. **Modules imported**: The following libraries are imported with the functions that are further used in the code majorly in the orbit() function, ImgPlot() function [which shows the contour plot] and radial velocity() function.

```
#********************************** MODULES IMPORTED ***********************************

from numpy import arange,real
from cmath import pi,sin,cos,acos,sqrt,atan
from math import radians
from datetime import *
from time import *
import matplotlib.pyplot as pl
import math as m
```

```
import numpy as np
from PIL import Image
from astropy.io import fits
import matplotlib.pyplot as plt
from astropy.visualization import astropy_mpl_style
```

2. **Inputs:** The code asks for several inputs from the user to carry out the calculation and print the resultant values that the program returns.

- t: Time of epoch of observation
- sys_dys: System distance from earth
- inc_deg: Inclination angle
- longAN_deg: Longitude of Ascending Node
- e: Eccentricity of the binary system
- T: Time of last periapsis
- Pds: Orbital Period of the binary system

- argperi_deg: Argument of periapsis in orbital plane
- m1: Mass of Primary Star
- m2: Mass of Secondary Star
- alpha1: luminosity relation for primary star
- alpha2: luminosity relation of secondary star
- quad: Quadrant in which primary star lies.
- num: To check whether user has the value of observed peak wavelength or not.
- star_call: To display the Star type of Primary or Secondary Star based on user input.
- image_file: To obtain the name of the image file in order to display the contour plot.
- lambda_peak: To obtain the observed value of peak wavelength of the star.
- n: Maximum limit of Distance from star.

```
#************************************************************** INPUTS **************************
print('/n ********************** INPUTS ********************** /n')
print('/n ********************** INPUTS ********************** /n')
t=float(input("Enter epoch of observation (in epoch timestamp)= "))
sys_dys=float(input("Enter system distance from earth (in parsec)= "))
ang_arcsec=float(input("Enter seperation angle of stars (in degrees) = "))
inc_deg=float(input("Enter inclination angle (in degrees) = "))
longAN_deg=float(input("Enter longitude of ascending node (in degrees) = "))
e=float(input("Enter eccentricity of the orbit = "))
T=float(input("Enter time of last periapsis (in epoch timestamp) = "))
Pds=float(input("Enter period of orbits (in years) = "))
argperi_deg=float(input("Enter Argument of periapses in orbital plane (in degrees) = "))
m1=float(input("Enter mass of primary star (in terms of solar mass) = "))
m2=float(input("Enter mass of secondary star (in terms of solar mass) = "))
alpha1=float(input("Enter value of alpha in the luminosity relation for the primary star = "))
alpha2=float(input("Enter value of alpha in the luminosity relation for the secondary star = "))
```

```
quad=int(input("Enter the quadrant in which primary star(m1) Lies => "))
```

```
print('Do you have the peak wavelength of the star? ')
num=int(input('Enter 1 if YES. Enter 2 if NO.'))
```

```
image_file=str(input('Enter the name of image file (with .jpg)'))
```

```
lambda_peak=float(input("Enter the peak wavelength of the Star. "))
```

```
n=int(input('Enter the max limit for distance away from the star.'))
```

3. **<u>Time Stamp Conversion:</u>** The Unix epoch is the number of seconds that have elapsed since January 1, 1970 (midnight UTC/GMT), not counting leap

seconds. This part of the code is basically converting the epoch daytime to normal day time.

```
#*************************** TIME STAMP CONVERSION ****************************

date_time1=datetime.utcfromtimestamp(t)
time1=date_time1.strftime("%m/%d/%Y, %H:%M:%S")
#print(time1)
date_time2=datetime.utcfromtimestamp(T)
time2=date_time2.strftime("%m/%d/%Y, %H:%M:%S")
#print(time2)

Time_period_of_obs=T-t
date_time_final=datetime.utcfromtimestamp(Time_period_of_obs)
time_final=date_time_final.strftime("%X")
print('Time Period of Observation', time_final)
```

4. **<u>Function Calls:</u>** In this part of the code, we have specified the range of the input variables.

```
#************************************************ FUNCTION CALLS ************

statement=bool(sys_dys>=0.01 and sys_dys<=50 and ang_arcsec>=0.01 and
               ang_arcsec<=2.0 and inc_deg>=0.001 and inc_deg<=179.999
               and longAN_deg>=0.001 and longAN_deg<=179.999 and e>=0.001
               and e<=0.95 and m1>=0.1 and m1<=10 and m2>=0.1 and m2<=m1
               and Pds>=1 and Pds<=100)
```

The first function that we are calling is the Orbit Calculator which calculates the various orbital element of the binary system after which the Radial Velocity function is called, and the radial velocity vs time curve is plotted. Followed by the StarType() or ImgPlot() function, which is followed by the Simulation.

```
if statement==True:
        ang_arcsec=ang_arcsec*pi/100/3600
        inc_deg=radians(inc_deg)
        longAN_deg=radians(longAN_deg)
        argperi_deg=radians(argperi_deg)
        a1,a2,v1r,v2r,L1,L2,pos_angle,TA,e_latest_last,M_deg,n=orbit(t, sys_dys, ang_arcsec, inc_deg, longAN_deg, e, T, Pds, argperi_deg, m1, m2, alpha,alpha1)
        print('/n ********************** OUTPUTS ********************** /n')
        print('Semi-Major axis of Primary Star : ',a1.real,' (parsec)')
        print('Semi-Major axis  of Secondary Star : ',a2.real,' (parsec)')
        print('Peak Radial Velocity of Primary Star : ',v1r.real,' (km/s)')
        print('Peak Radial Velocity of Secondary Star : ',v2r.real,' (km/s)')
        print('Luminosity of Primary Star : ',L1,' (in terms of Solar Luminosity)')
        print('Luminosity of Secondary Star : ',L2,' (in terms of Solar Luminosity)')
        print('Position Angle of the Binary Star System in Image-plane : ',pos_angle,' (degrees)')
        print('True Anomaly of the Binary Star System : ',TA.real,' (degrees)')
        print('Eccentric Anomaly of the Binary Star System : ',e_latest_last.real,' (degrees)')
        print('Mean Anomaly of the Binary Star System : ',M_deg,' (degrees)')
        print('Mean Motion of the Binary Star System : ',n,' (radians per year)')


        V1,V2=radial_velocity(Pds, argperi_deg, m1, m2)
        pl.plot(arange(0,Pds-1),V1)
        pl.plot(arange(0,Pds-1),V2)
        pl.grid()
        pl.xlabel('Time (yrs)')
        pl.ylabel('Radial Velocity () ')
        pl.legend(['Primary Star','Secondary Star'],loc='upper right')
        pl.title('Radial Velocity Curve for Binary Star')
        pl.show()

        print('Do you have the peak wavelength of the star? ')
        num=int(input('Enter 1 if YES. Enter 2 if NO.'))

        if num==1:
            star_call=int(input('Enter 1 to display Star Type of Primary Star and Enter 2 to display Star Type of Secondary Star'))
            if star_call==1:
                D,Temp=StarType(L1)
                pl.plot(D,Temp)
                pl.grid()
                pl.xlabel('Distance from Star (AU)')
                pl.ylabel('Temperature of the Star (Solar Temperature)')
                pl.legend('Distance vs Temperature Curve')
                pl.show()
            elif star_call==2:
                D,Temp=StarType(L2)
                pl.plot(D,Temp)
                pl.grid()
                pl.xlabel('Distance from Star (AU)')
                pl.ylabel('Temperature of the Star (Solar Temperature)')
                pl.legend('Distance vs Temperature Curve')
                pl.show()
            else:
                print('Wrong value entered. Try again later.')
        elif num==2:
            image_file=str(input('Enter the name of image file (with .jpg)'))
            ImgPlot(image_file)
        else:
            print("Error Encountered!")


        Simulate(a1.real,a2.real,e,inc_deg,100)
```

Here, we have used the if/else statement because even if a single input value is invalid or out of range in the ‘if’ statement, none of the functions will run and the code will jump to the ‘else’ statement, thus printing the error message: “Inputs entered were not in the right range”

5. **<u>Orbit calculator:</u>** We have used modules and taken binary equations to help the function perform the tasks.

    a. **‘Orbit()’ function:** This function calculates and returns values like (the Semi-major axis of both primary and secondary stars, Peak radial velocities of both primary and secondary stars, Luminosity of both primary and secondary stars, position angle of the Binary Star system in Image Plane, True Anomaly of the system, Eccentric Anomaly of the system, Mean anomaly of the system and the Mean motion

exhibited by the Binary System after which all the calculated values are returned.

```
def orbit(t,sys_dys,ang_arcsec,inc_deg,longAN_deg,e,T,Pds,argperi_deg,m1,m2,alpha,alpha1):

    #MEAN MOTION
    n=2*pi/Pds

    M_deg=n*(Time_period_of_obs)
    #MEAN ANOMALY
    M_deg=radians(M_deg)

    #ECCENTRIC ANOMALY
    e_last=M_deg+ e*sin(M_deg)+ (e**2/(2*M_deg))*sin(2*M_deg)

    for i in range(4):
        M0_deg=e_last-e*sin(e_last)
        e_latest_last=e_last +((M_deg-M0_deg)/(1- e*cos(e_last)))
        e_last=e_latest_last

    #TRUE ANOMALY
    TA=acos((cos(e_latest_last)-e)/(1-e*cos(e_latest_last)))

    #Seperation Distance in Reference Plane from provided Seperation Distance and Seperation Angle AU
    sep_dis_rp=ang_arcsec*sys_dys

    #Seperation Distance in Orbital Plane
    sep_dis_op=sep_dis_rp/(sqrt(((sin(argperi_deg+TA)*((((cos(complex(0,1))).real))**2)+(cos((argperi_deg+TA))**2)))).real

    #Seperation Distance in Reference Plane with respect to the Line of Nodes
    sep_dis_rp_y=sep_dis_op*sin(argperi_deg+TA)*((cos(complex(0,1))).real)       #Y-component
    sep_dis_rp_x=sep_dis_op*cos(argperi_deg+TA)                                  #X-component

    #Angle between the Line of Nodes and Primary Inclination
    angln_pi_deg=atan(sep_dis_rp_y/sep_dis_rp_x)

    #Position Angle in Image
    quad=int(input("Enter the quadrant in which primary star(m1) Lies => "))
    if quad==1 or quad==4:
        pos_angle=longAN_deg+angln_pi_deg +180
    else:
        pos_angle=longAN_deg+angln_pi_deg

    #Sum of SemiMajor axes of m1 and m2
    a_total=sep_dis_op*(1+e*cos(TA))/(1-e**2)

    #SemiMajor Axis of m1
    a1=a_total/(1+(m2/m1))

    #SemiMajor Axis of m2
    a2=(a1/m2)*m1

    #Radial Velocity
    v1r=((((m2*sin(inc_deg))**3)*2*pi*6.67*1e-11)/(((m1+m2)**2)*Pds)**(1/3)
    v2r=((((m1*sin(inc_deg)**3)*2*pi*6.67*1e-11)/(((m1+m2)**2)*Pds)))**(1/3)

    #Luminosity of m1 and m2
    L1=((m1)**alpha)
    L2=((m2)**alpha1)

    return a1,a2,v1r,v2r,L1,L2,pos_angle,TA,e_latest_last,M_deg,n
```

b. **‘radial_velocity’ function:** This is the second function that is called. Here, we are calculating the peak velocities for the epoch time (k1 & k2). Then, for the whole orbital period, we are calculating v1 and v2 (radial velocities for m1 and m2) for different times as the Argument of Periapsis in orbital plane (argperi_deg) is changing. We then take those values and append them in lists (V1 & V2) which are used to plot the radial velocity vs time curve.

```
def radial_velocity(pds,argperi_deg,m1,m2):
    k1=((((m2*sin(argperi_deg))**3)*2*pi*4.3009e-3)/(((m1+m2)**2)*pds))**(1/3)
    k2=((((m1*sin(argperi_deg)**3)*2*pi*4.3009e-3)/(((m1+m2)**2)*pds)))**(1/3)
    omega=2*pi/pds
    V1=[]
    V2=[]
    for j in range(0,int(pds)):
        v1=k1*m.sin(omega*j+argperi_deg)
        v2=k2*m.sin(omega*j+argperi_deg)
        V1.append(v1)
        V2.append(v2)

    return V1,V2
```

c. **StarType() function:** Depending on the user input, this function is called. Here, we use Wien's Displacement Law $T = b/\lambda_{peak}$ and the relation between Luminosity (L) and Temperature (T) i.e., $T = \left(\frac{L}{16\pi\sigma i^2}\right)^{\frac{1}{4}}$. Then, according to the data given, different characteristics of the star are displayed. Further, Distance vs Temperature curve is plotted for the star.

```
def StarType(L):
    lambda_peak=float(input("Enter the peak wavelength of the Star. "))
    b=2.8981e-3
    T=b/(lambda_peak*5778)      #Peak Surface Temperature in terms of Solar Temperature
    print('Peak Surface Temperature of the star is : ',T,' (in terms of Solar Temperature)')
    R=sqrt(L)* (1/(T**2))       #Radius of the Star in terms of Solar Radius
    print('Radius of the star in terms of Solar Radius is: ',R.real)

    if R.real<0.1 and L<1:
        print('The star is a white dwarf.')
    elif R.real>=0.1 and R.real<1 and L>=1 and L<10**2:
        print('The star is a main sequence star.')
    elif R.real>=1 and R.real<10:
        print('Star is of similar magnitude as that of Sun.')
    elif R.real>=10 and R.real<=100 and L>=10**2 and L<10**3:
        print('The star is a Red Giant.')
    elif R.real>=10 and R.real<=1000 or R.real>1000 and L>=10**4 and L<=10**6:
        print('The star is a supergiant.')

    if T>=1.7307 and T<=6.9228:
        print('Star is Hot.')
    elif T>=0.4326 and T<=0.8653:
        print('Star is Cool.')
    if L>100:
        print('Star is Bright.')
    elif L>=0.01 and L<=100:
        print('Star is neither too bright not too dim.')
    elif L<0.01:
        print('Star is Dim.')

    n=int(input('Enter the max limit for distance away from the star.'))
    D=[]
    for i in range(1,n+1):
        D.append(i)
    sigma=float(input('Enter the value of Stefan-Boltzmann Constant for the star.'))
    Temp=[]
    for i in range(1,n+1):
       tem=(L/(16*pi*sigma*i**2))**(1/4)
       Temp.append(tem)
    print('TEMPERATURE = ', Temp)

    return D, Temp
```

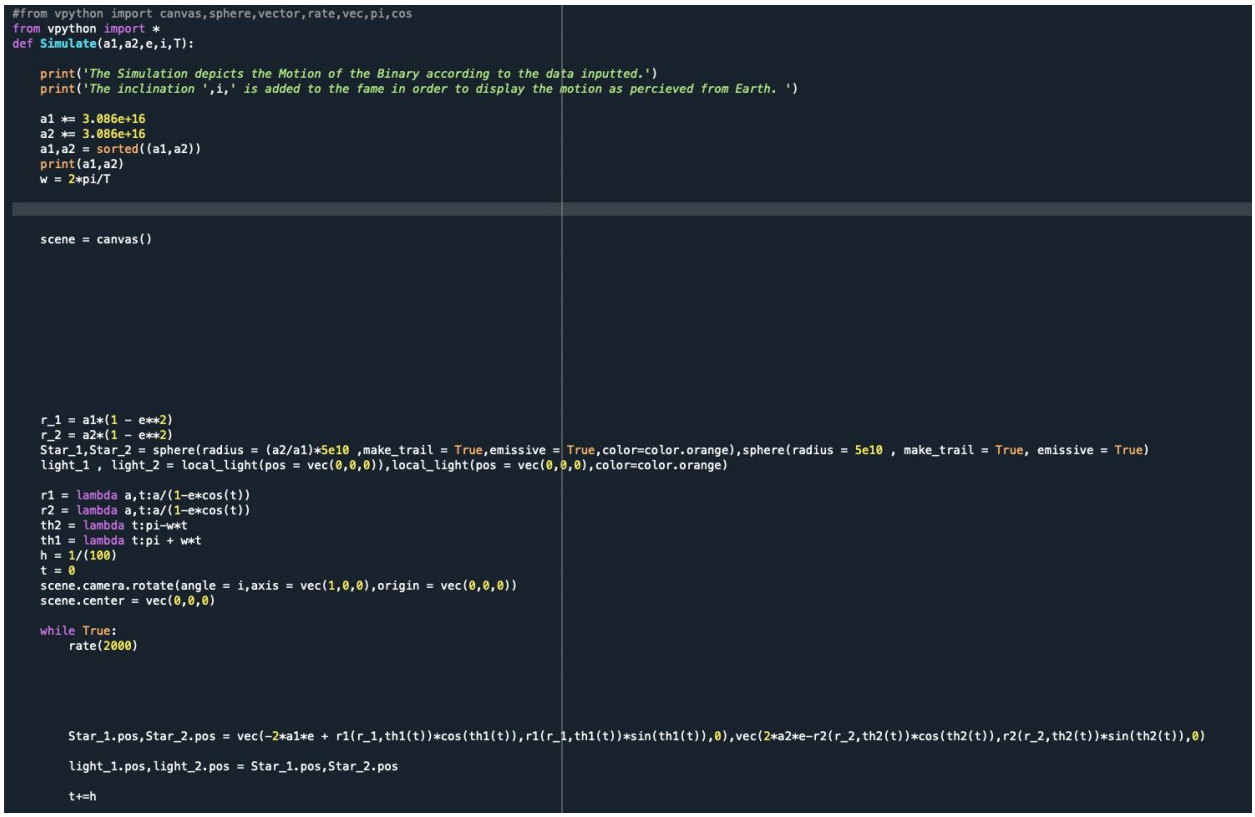
d. **ImgPlot() function:** Depending on the user input, this function is called. Here, using the Astropy modules, the user is asked to enter the image file name (as saved on the system) and then the code checks if it matched with the default image or not. If it matches with the default, then the contour plot for the image is plotted using the basic matplotlib functions. Otherwise, the new image file is downloaded from system and converted to FITS file from JPG and then the contour is plotted.

Note: The peak wavelength value (if the user doesn't know) can be determined from the contour. However, it doesn't work for x-ray images and obtaining the telescopic images is a bit difficult.

```
def ImgPlot(image_file):

    import numpy as np
    from PIL import Image
    from astropy.io import fits
    import matplotlib.pyplot as plt
    from astropy.visualization import astropy_mpl_style

    default='sirius.jpg'
    if image_file==default:
        ################################################################################
        # Use `astropy.io.fits.info()` to display the structure of the file:

            fits.info('red.fits')


        # Generally the image information is located in the Primary HDU, also known
        # as extension 0. Here, we use `astropy.io.fits.getdata()` to read the image
        # data from this first extension using the keyword argument ``ext=0``:

            image_data = fits.getdata('red.fits', ext=0)

            # The data is now stored as a 2D numpy array. Print the dimensions using the
            # shape attribute:

            print(image_data.shape)

        # Display the image data:

            plt.figure()
            plt.imshow(image_data, cmap='gnuplot')
            plt.colorbar()
    else:

        plt.style.use(astropy_mpl_style)

        # Load and display the original 3-color jpeg image:

        image = Image.open(image_file)
        xsize, ysize = image.size
        print(f"Image size: {ysize} x {xsize}")
        print(f"Image bands: {image.getbands()}")
        ax = plt.imshow(image)

        # Split the three channels (RGB) and get the data as Numpy arrays. The arrays
        # are flattened, so they are 1-dimensional:

        r, g, b = image.split()
        r_data = np.array(r.getdata()) # data is now an array of length ysize*xsize
        g_data = np.array(g.getdata())
        b_data = np.array(b.getdata())
        print(r_data.shape)

        # Reshape the image arrays to be 2-dimensional:

        r_data = r_data.reshape(ysize, xsize) # data is now a matrix (ysize, xsize)
        g_data = g_data.reshape(ysize, xsize)
        b_data = b_data.reshape(ysize, xsize)
        print(r_data.shape)

        # Write out the channels as separate FITS images.
        # Add and visualize header info

        red = fits.PrimaryHDU(data=r_data)
        red.header['LATOBS'] = "32:17:56" # add spurious header info
        red.header['LONGOBS'] = "110:56"
        red.writeto('newred.fits')

        green = fits.PrimaryHDU(data=g_data)
        green.header['LATOBS'] = "32:17:56"
        green.header['LONGOBS'] = "110:56"
        green.writeto('newgreen.fits')

        blue = fits.PrimaryHDU(data=b_data)
        blue.header['LATOBS'] = "32:17:56"
        blue.header['LONGOBS'] = "110:56"
        blue.writeto('newblue.fits')

        from pprint import pprint
        pprint(red.header)


        ################################################################################
        # Use `astropy.io.fits.info()` to display the structure of the file:

        fits.info('newred.fits')


        # Generally the image information is located in the Primary HDU, also known
        # as extension 0. Here, we use `astropy.io.fits.getdata()` to read the image
        # data from this first extension using the keyword argument ``ext=0``:

        image_data = fits.getdata('newred.fits', ext=0)

        # The data is now stored as a 2D numpy array. Print the dimensions using the
        # shape attribute:

        print(image_data.shape)

        # Display the image data:

        plt.figure()
        plt.imshow(image_data, cmap='gnuplot')
        plt.colorbar()
```

e. Simulate(): The Simulate() function is imported from the SimulatorFinale.py file. The function takes the precalculated values for the system like eccentricity, semi major axes of both primary and secondary star (in parsec). Then, semi major axes are converted to units of m. We take dt to be a small increment of time. Next, we defined time t and set it to 0. Then, we defined functions th1(t) and th2(t) which give the polar angles from the center at a given time. Then we are rotating the camera by the inclination angle to give the perspective of the binary system from Earth. Then an infinite loop is

run to find the position vector of each star in polar coordinates which is then converted to cartesian. Then the position vectors are shifted such that the common foci sits at the origin, all the while t is incremented by dt time.



```
#from vpython import canvas,sphere,vector,rate,vec,pi,cos
from vpython import *
def Simulate(a1,a2,e,i,T):

    print('The Simulation depicts the Motion of the Binary according to the data inputted.')
    print('The inclination ',i,' is added to the fame in order to display the motion as percieved from Earth. ')

    a1 *= 3.086e+16
    a2 *= 3.086e+16
    a1,a2 = sorted((a1,a2))
    print(a1,a2)
    w = 2*pi/T


    scene = canvas()









    r_1 = a1*(1 - e**2)
    r_2 = a2*(1 - e**2)
    Star_1,Star_2 = sphere(radius = (a2/a1)*5e10 ,make_trail = True,emissive = True,color=color.orange),sphere(radius = 5e10 , make_trail = True, emissive = True)
    light_1 , light_2 = local_light(pos = vec(0,0,0)),local_light(pos = vec(0,0,0),color=color.orange)

    r1 = lambda a,t:a/(1-e*cos(t))
    r2 = lambda a,t:a/(1-e*cos(t))
    th2 = lambda t:pi-w*t
    th1 = lambda t:pi + w*t
    h = 1/(100)
    t = 0
    scene.camera.rotate(angle = i,axis = vec(1,0,0),origin = vec(0,0,0))
    scene.center = vec(0,0,0)

    while True:
        rate(2000)




        Star_1.pos,Star_2.pos = vec(-2*a1*e + r1(r_1,th1(t))*cos(th1(t)),r1(r_1,th1(t))*sin(th1(t)),0),vec(2*a2*e-r2(r_2,th2(t))*cos(th2(t)),r2(r_2,th2(t))*sin(th2(t)),0)

        light_1.pos,light_2.pos = Star_1.pos,Star_2.pos

        t+=h
```

RESULT

Input values:

S.No.	Input Parameters	Values
1.	Epoch of Observation	1994.5715
2.	System Distance from Earth	2.64
3.	Separation Angle of Stars	0.3
4.	Inclination Angle	139.336
5.	Longitude of Ascending Node	45.400
6.	Eccentricity of the Orbit	0.59142
7.	Time of Last Periapsis	55367.839
8.	Orbital Period of the Binary	50.1
9.	Argument of Periapsis	30.839
10.	Mass of Primary Star	2.063
11.	Mass of Secondary Star	1.018
12.	Alpha of Primary Star	4.47
13.	Alpha of Secondary Star	-161
14.	Peak value of Wavelength of the Star	2.915e-7
15.	Maximum Distance from the Star (limiting value)	100

Output values:

```
/n ********************** OUTPUTS ********************** /n
Semi-Major axis of Primary Star :  9.697055475190194e-06  (parsec)
Semi-Major axis  of Secondary Star :  1.9651302009152626e-05  (parsec)
Peak Radial Velocity of Primary Star :  0.0008642979386391914  (km/s)
Peak Radial Velocity of Secondary Star :  7.9528652610553e-05  (km/s)
Luminosity of Primary Star :  25.273567937105277  (in terms of Solar Luminosity)
Luminosity of Secondary Star :  1.0230978118999112  (in terms of Solar Luminosity)
Position Angle of the Binary Star System in Image-plane :  (181.29203424185914+0j)  (degrees)
True Anomaly of the Binary Star System :  2.951053097366596  (degrees)
Eccentric Anomaly of the Binary Star System :  116.6117355581723  (degrees)
Mean Anomaly of the Binary Star System :  116.82714929058736  (degrees)
Mean Motion of the Binary Star System :  0.12541288038282608  (radians per year)
```

```
The Simulation depicts the Motion of the Binary according to the data inputted.
The inclination  2.4318719665588193  is added to the fame in order to display the motion as percieved from Earth.
```

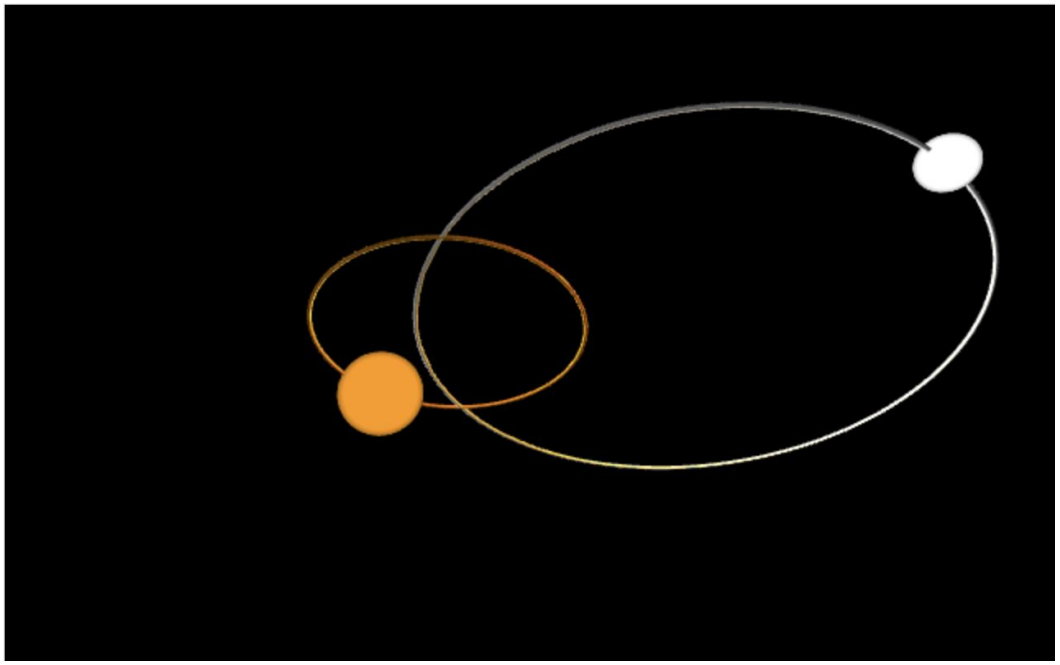


FIG: SIMULATION OF THE BINARY SYSTEM

```
Do you have the peak wavelength of the star?

Enter 1 if YES. Enter 2 if NO.2

Enter the name of image file (with .jpg)sirius.jpg
Filename: red.fits
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU       8   (400, 329)   int64
```

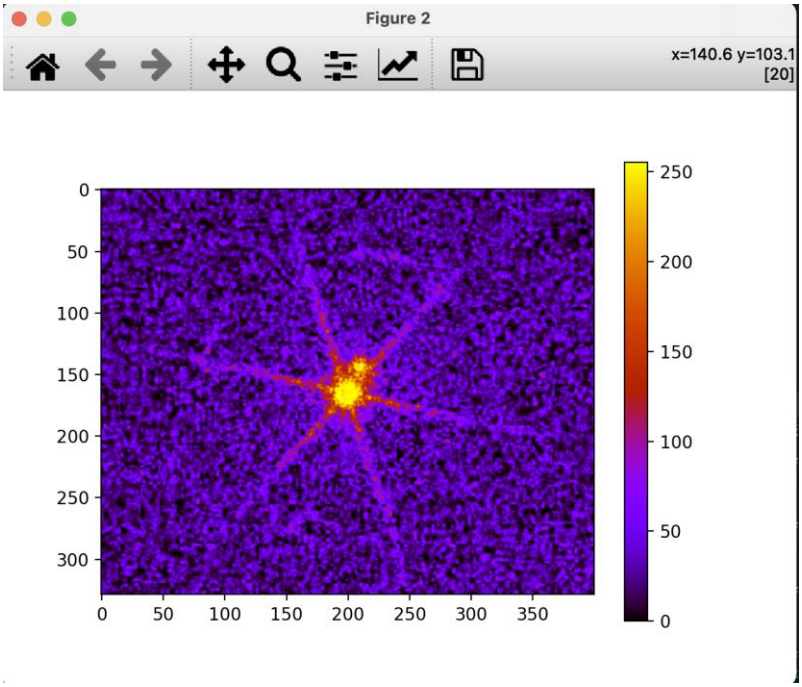


FIG: CONTOUR PLOT OF THE IMAGE

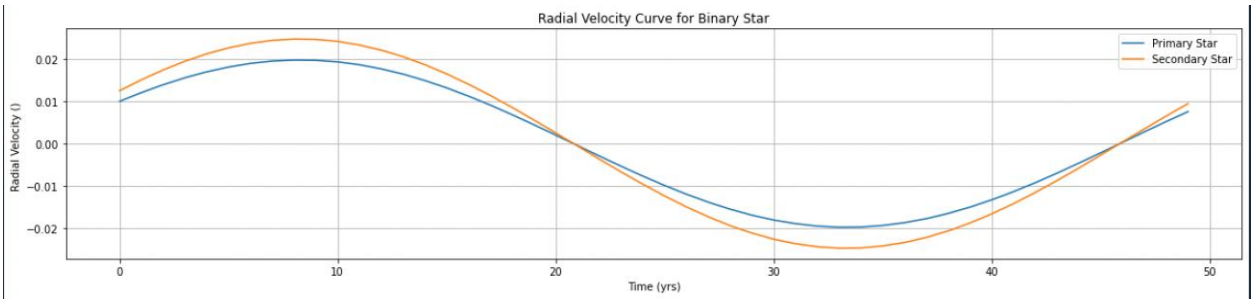


FIG: RADIAL VELOCITY CURVE FOR THE BINARY SYSTEM

```
Do you have the peak wavelength of the star?

Enter 1 if YES. Enter 2 if NO.1

Enter 1 to display Star Type of Primary Star and Enter 2 to display Star Type of Secondary Star1

Enter the peak wavelength of the Star. 2.915e-7
Peak Surface Temperature of the star is :  1.7206687458847572  (in terms of Solar Temperature)
Radius of the star in terms of Solar Radius is:  1.6980041127574053
Star is of similar magnitude as that of Sun.
Star is neither too bright not too dim.

Enter the max limit for distance away from the star.100

Enter the value of Stefan-Boltzmann Constant for the star.5.6703e-8
```

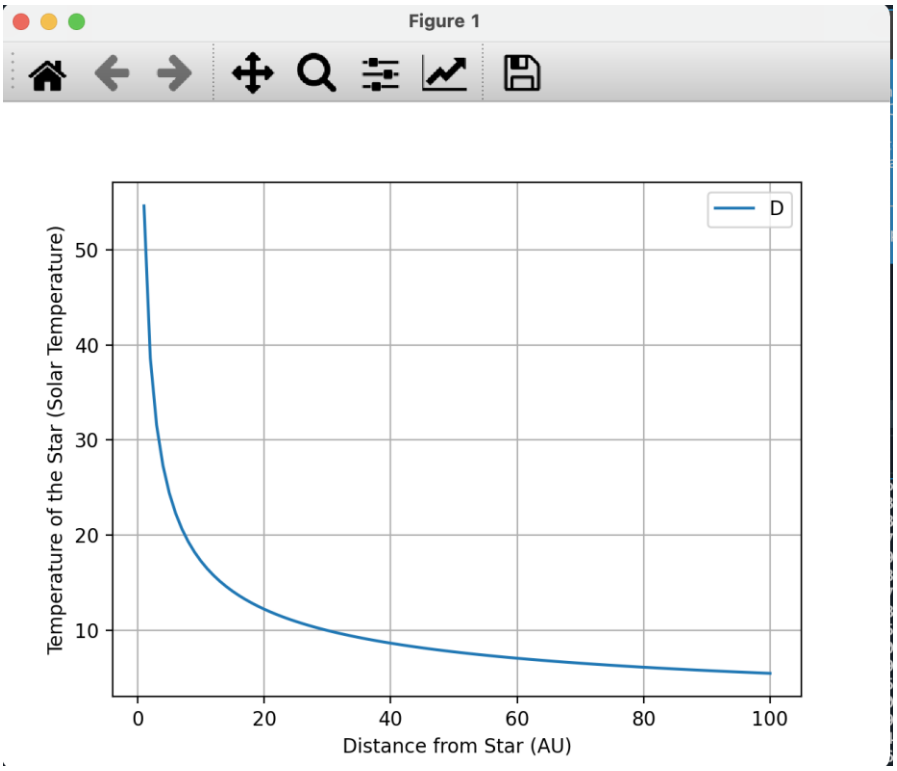


FIG: TEMPERATURE VS DISTANCE CURVE

ERROR ANALYSIS

The amount of error depends on the observational data entered by the user and the decimal point up to which the value is entered.

The Eccentric Anomaly can only be calculated for orbits with $e \leq 0.95$ with the transcendental formula (which is not algebraic) for approximation which is iterated four times to obtain an acceptable value of Eccentric Anomaly up to a certain accuracy. True Anomaly uses the Eccentric Anomaly calculated before and thus have certain error as well. Furthermore, the values calculated using True Anomaly will also account for the said errors.

Assuming that the binary star system consists of perfect blackbodies, thus we can calculate the peak radiation wavelength and Surface Temperature at different distances from the star using Temperature-Luminosity Relation.

The image contour that the code displays through the ImgPlot() function is designed in a way to enable user to determine the peak radiation wavelength. However, the accuracy of that is something we weren't able to determine due to the unavailability of the sourced/cited images. Furthermore, this method only works for visible range images.

LIMITATIONS

The code is unable to account any of the spectroscopic as well as astrometric part of the data, which is usually used to perform all calculations related to the binary systems. Thus, our code works only for the visual binaries.

The ImgPlot() function enables user to determine the peak radiation wavelength, however, due to the lack of source images, the usability of the function is still unknown.

The simulation deviates from the true value often because of background processes, if left running for a long time. We tried to add a GUI slider in the simulation, but the computer weren't able to process the commands and eventually the simulation broke down.

CONCLUSION

In conclusion, we see how the values of the quantities input and the calculations done by the code is applied in the simulation to demonstrate the motion of binary stars with respect to each other in the reference plane. The code also enables the user to determine the peak radiation wavelength using the contours of the telescopic images in the visible range and further analyses the various orbital elements of the binary system.

ACKNOWLEDGEMENT

This project was a great learning experience for all of us. We would like to thank our professor, Ms. Priya Johari for providing us with this opportunity and all her inputs and help during this project.

BIBLIOGRAPHY

https://sites.astro.caltech.edu/~george/ay20/Ay20-Lec4x.pdf

http://www.mit.edu/~iancross/8901_2019A/readings/Carroll_Ostlie_on_binary_stars.pdf

Heintz, W.D., 1990, *Astron. Astrophys. Suppl*., **82**, 65.

http://ugastro.berkeley.edu/infrared10/adaptiveoptics/binary_orbit.pdf