Modelling a Binary Star System

Alikhan Murat

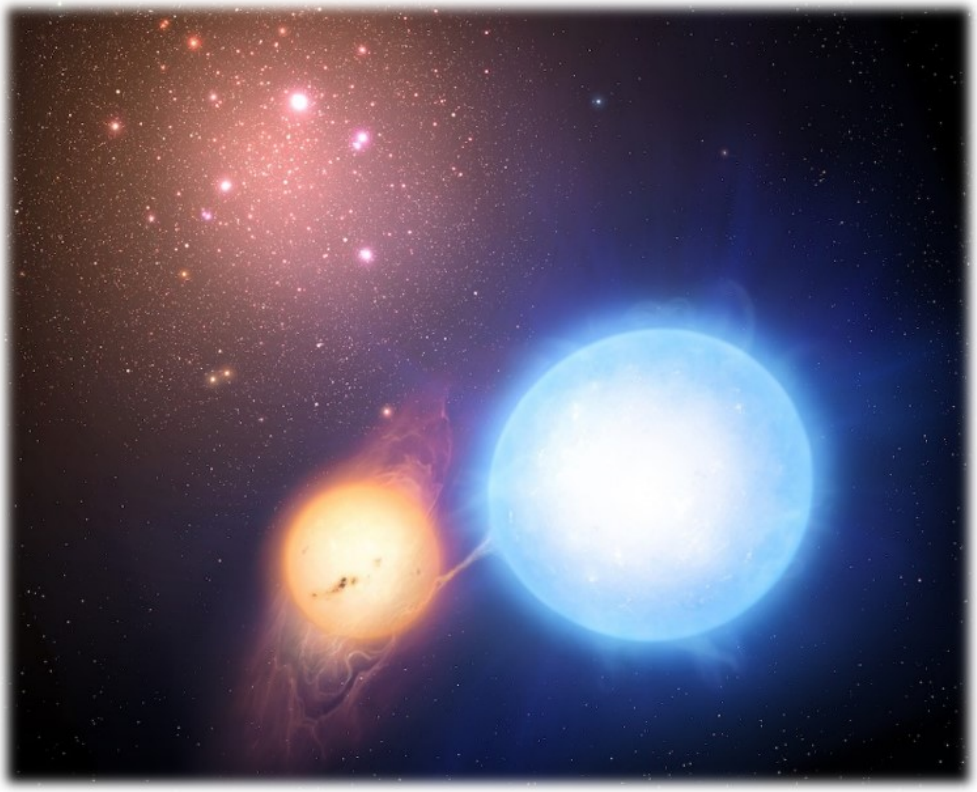


Table of Contents

1 Introduction

In this paper, I will outline the steps I took to simulate a binary star system using Newtonian mechanics in python. By collecting data from the simulation, it is possible to use it as a point of reference for real-world data on binary stars. This could allow us to infer more information about the stars in a given binary system and improve current methods for the detection of these systems.

Section 2 outlines current information on binary stars. This includes the different types of binary stars, known binary stars, and the expected orbits of the stars. In section 0 I explain the theory required to simulate the star system and the techniques I used to collect data about them. Finally, in section 0 I evaluate and present the data collected in the prior section.

Most of the code will be in Appendix A. This includes important sections of code that describe my implementation in more detail; however, this will omit boilerplate code not needed to understand the simulation. The entire codebase can be found on: https://github.com/amurat233/Binary-Star-System-Simulation.

2 Introduction to Binary Stars

A binary star is a system of two stars orbiting around a common centre of mass (COM). Over 80% of the individual points of light observed in the sky are systems of more than one star [1]. Binary stars are the most common multiple star systems, and they are also the easiest to model due to their simplicity. They are classified based on the way they can be detected; I discuss this in more detail in section 2.1. The orbits of binary stars are very similar to simple planetary orbits and obey Kepler's laws; I discuss the structure of the orbits in section 2.2.

2.1 Classification

There are four main types of binary stars: visual, astrometric, spectroscopic, and eclipsing [2]. Visual binaries are the simplest of the four, they are binary stars where you can see both stars orbit around each other through a telescope. This can be seen in Figure 1 where a visual binary can be seen completing around a quarter of an orbit. Due to their simplicity, I will be ignoring this class of binary stars in my investigation.

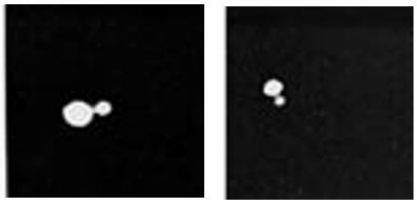


Figure 1: A visual binary in 1908 (left) and 1920 (right)

Astrometric binaries are like visual binaries in the way that they rely on tracing the path of a star to detect. In astrometric binaries only one star is visible; therefore, it appears as though it is orbiting around an invisible object since the visible star "wobbles". From this, it can be inferred that there is another star whose gravity is affecting the path of the visible star. An example of this type of binary is the Sirius system. Sirius A's much dimmer counterpart Sirius B was only discovered due to the slightly wave-like orbit of Sirius A [3]. Demonstrating an astrometric binary is also trivial since it involves hiding one of the stars; therefore, I decided to omit these stars from my investigation as well.

Spectroscopic binaries are the first of the two classes that come up in my investigation. They are detected by analysing the absorption spectra from the star system. Due to the nature of their orbits, one of the stars in the system will be moving towards the observer while the other will be moving away from the observer. This creates a set of two different absorption spectra, one slightly blueshifted and the other slightly redshifted. By measuring the redshift over time, you can determine information about the binary star such as its orbital period and perhaps even its eccentricity as I will further discuss in section 4.2. Figure 2 aims to visualise this effect, the absorption spectra of star A are blueshifted and the opposite is true for star B.

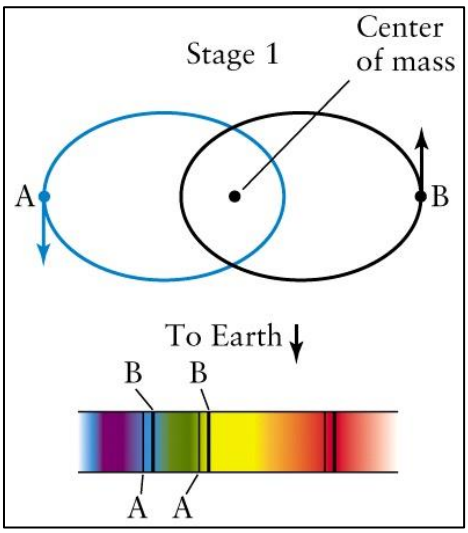


Figure 2: Spectroscopic binary

I also investigated eclipsing binary stars; these are systems whose luminosity changes over time due to the stars eclipsing each other. When the star closest to the observer covers part or all of the star further away from the observer it also blocks the light emitted by the star. As a result, the total luminosity of the system decreases every time the stars eclipse each other. By measuring the luminosity of a system over time, we can determine the sizes, masses, rotation rates, temperatures, and orbits of the stars [4]. An example of a luminosity-time graph can be seen in Figure 3.

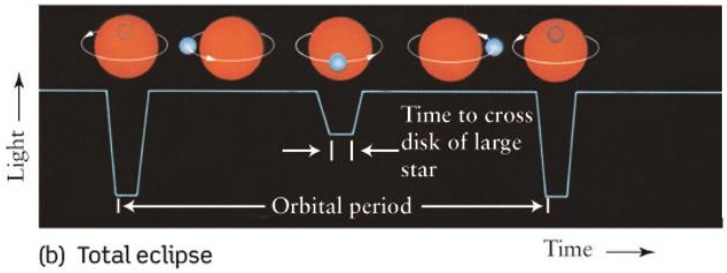


Figure 3: Eclipsing binary

2.2 Orbits

The orbits of binary stars follow Kepler's laws of planetary motion, namely the way that they have elliptical orbits with the centre of mass at one of the foci and that a line connecting the star to the centre of mass sweeps out equal area in equal time. I verify that my simulation obeys both laws in section 4.1.

Both stars orbit around a common centre of mass, the stars are always on opposite sides of this point and the distance to the centre of mass from one of the stars can be

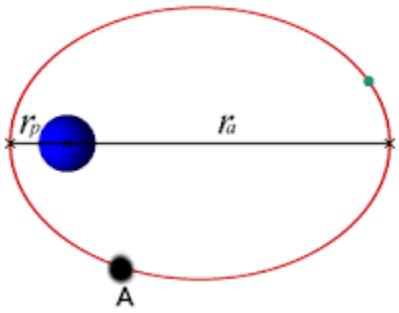
calculated from equation (1) where r_1 is the distance to the centre of mass from the star with mass M_1. r_2 and M_2 are the second star's counterparts of these values.

$$r_1 M_1 = r_2 M_2 \tag{1}$$

Since the total distance between the stars $r = r_1 + r_2$ we can rearrange equation (1) to equation (2) allowing us to find the distance to the centre of mass from star 1.

$$r_1 = \frac{r M_2}{M_1 + M_2} \tag{2}$$

The measure of how elliptical the stars' orbits are is known as the eccentricity. A larger eccentricity indicates a more circular orbit. It is defined as the ratio between the distance between the foci of the orbit and the length of the major axis of the ellipse. It has a value between 0 and 1. It can also be calculated by using only the distance from the apoapsis to the COM r_a and the distance from periapsis to the COM r_p[1]. This can be seen in equation (3).



$$e = \frac{r_a - r_p}{r_a + r_p} \tag{3}$$

[1] The apoapsis is the point where the star is furthest away from the centre of mass of the system. The periapsis is the point where the star is closest to the centre of mass.

3 Theory and Simulation (How to play God)

In this section, I outline the theory needed to simulate the orbits of the stars and how I used the data collected from the simulation to calculate the redshifts of the individual stars (section 4.2) and the total luminosity of the system (section 0).

3.1 Newton s Law of Universal Gravitation

The simulation was based on Newton's law of universal gravitation shown in equation (4) where G, M_1M_2, and r are the gravitational constant, mass product, and distance, respectively.

$$F = G\frac{M_1M_2}{r^2} \tag{4}$$

I designed the simulator to be able to simulate more than just two bodies attracting each other, as a result every celestial body in the simulation must exert a force on every other body in the simulation that is directly proportional to the product of the bodies masses and inversely proportional to the square of their distances.

At each time step, I calculate the resultant force on every celestial body in the scene by summing the vector forces exerted on it by all the other celestial bodies in the scene. From this, I use Newton's second law ($F = ma$) to calculate the acceleration of the object and by adding it to the current velocity I can get its new velocity. I repeat this with every celestial body in the scene before I update their positions all at once. If the objects were moved individually the simulation would be unstable and inaccurate since

upon moving the object you change the force it exerts on the other objects in the scene. By repeating this over a very large number of timesteps you can simulate the entire orbits of the stars. However, the stars have inertia and so they tend to move off-screen, to counteract this, at each frame I also subtract the velocity of the centre of mass to "focus" on the COM. An example of the same simulated orbit with and without focus can be seen in Figure 4. The mass of the centre star is thrice the mass of the outer star. Note that all the units in the simulation are arbitrary and do not represent any physical distances.

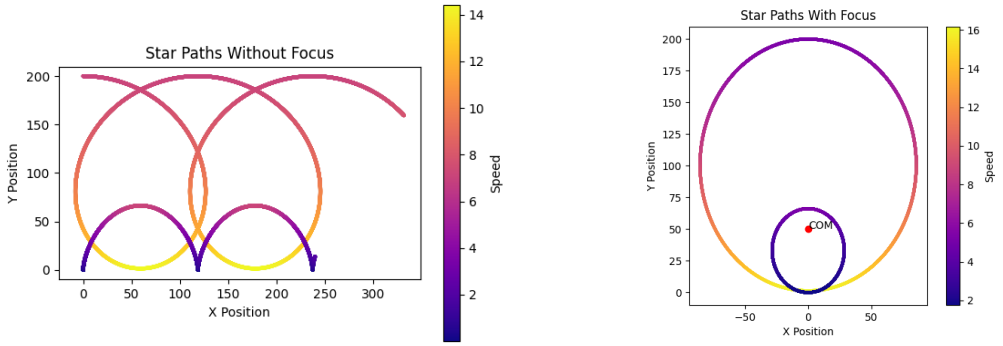


Figure 4: An orbit.

In a three-body system, you can also focus on one of the stars to see the orbits from its perspective.

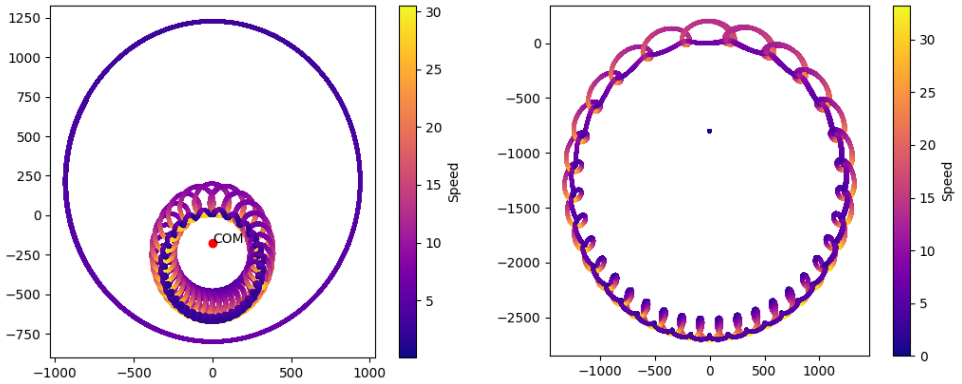


Figure 5: Three stars with focus as COM (left) and the outer star (right).

From these graphs you can see that the stars move fastest when they are closer to periapsis and slowest at apoapsis, this agrees with Kepler's second law which I further verify in section 4.1.

The code for the "celestialBody" class that the stars are based on can be found in Appendix A.

3.2 Redshift Measurement

To simulate spectroscopic binary stars, I had to find a way to measure the redshift of the stars at any time step. I was able to use the position of the observer, the position and velocity of the star, and a constant c for the speed of light to determine the star's redshift. Redshift can be calculated from wavelengths using equation (5) [5].

$$z = \frac{\lambda_{obs}}{\lambda_{src}} - 1 \tag{5}$$

At speeds near the speed of light λ_{obsv} can be calculated using equation (6) where v is the recessional velocity [6].

$$\lambda_{obs} = \lambda_{src}\sqrt{\frac{c+v}{c-v}} \tag{6}$$

Combining (5) and (6) we get equation (7):

$$z = \sqrt{\frac{c+v}{c-v}} - 1 \tag{7}$$

Since c is a universal constant, I simply set it to 100; however, this choice was mostly random as I chose it simply because I felt it was sufficiently larger than the maximum speed of the stars in my simulated system. The recessional velocity was calculated at every frame from the star's relative position to the observer and the velocity of the star. This was done by resolving the velocity vector parallel to the relative position vector and could be done using the dot product as you can see in equation (8).

$$\overrightarrow{p_{rel}} = \overrightarrow{p_{star}} - \overrightarrow{p_{obs}}$$

$$v = \frac{\overrightarrow{v_{star}} \cdot \overrightarrow{p_{rel}}}{|\overrightarrow{p_{rel}}|} \tag{8}$$

3.3 Luminosity Measurement

Two problems stand in the way of calculating the total luminosity of the system: determining the luminosity of the individual stars and calculating by how much the closer star covers the further star.

The first of the two is by far the easiest to solve. The relationship between the mass (M) and luminosity (L) of a star in the main sequence is stated in equation (9) [7]. Since every "celestialBody" object already has a mass attribute I used it to calculate the luminosity of each star in the system. The constant of proportionality was set to 1.

$$L \propto M^{3.5} \tag{9}$$

Finding the proportion of the luminosity that reaches the observer was an exercise in projective geometry. To simplify the calculations, I had to assume that the inclination of the stars' orbits were 0 degrees relative to the observer. I also assumed that the star

emits light uniformly (i.e. the edges of the star have the same luminosity as the centre of the star). Having made these assumptions, I was able to find the total luminosity of the system using the radii of both stars, their positions, their luminosities, and the position of the observer.

The first step was to project the further star onto a plane so that it is the same distance from the observer as the closer star. This resizes the star in such a way that the proportion of the sizes of the two stars is the same as what is seen by the observer. This can be seen in Error! Reference source not found..

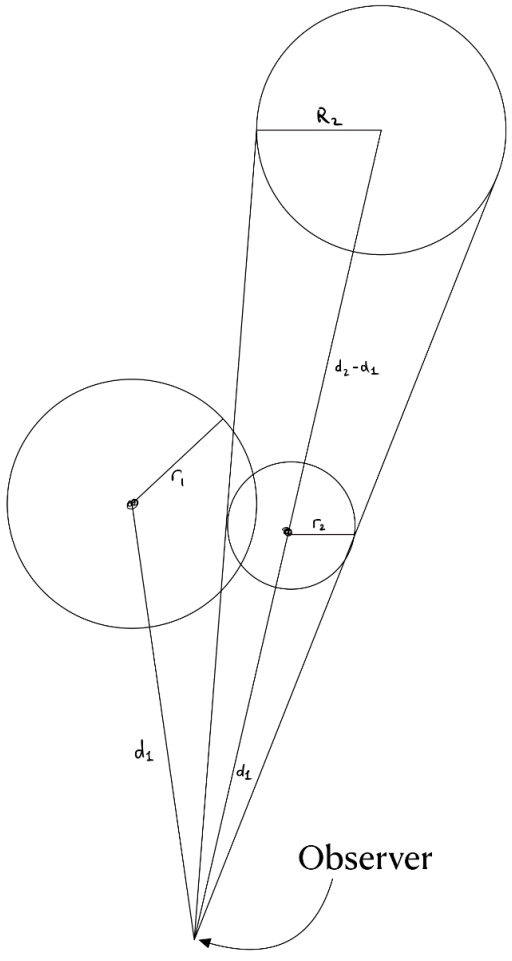


Figure 6: Resizing the eclipsed star.

First the ratio between the distance from the observer of the further star and the closer star is calculated. Then both the resized star's position vector $\overrightarrow{p_2}$ and radius r_2 are calculated by multiplying the original stars position vector and radius by

$$\overrightarrow{p_2} = \frac{d_1}{d_2}\overrightarrow{P_2} \tag{10}$$

$$r_2 = \frac{d_1}{d_2}R_2 \tag{11}$$

Once the star has been resized it falls under three possible scenarios: the star isn't being eclipsed, the star is partially eclipsed, or the star is maximally eclipsed. The next step to calculate the system's luminosity is dependent upon which of these scenarios the stars fall into.

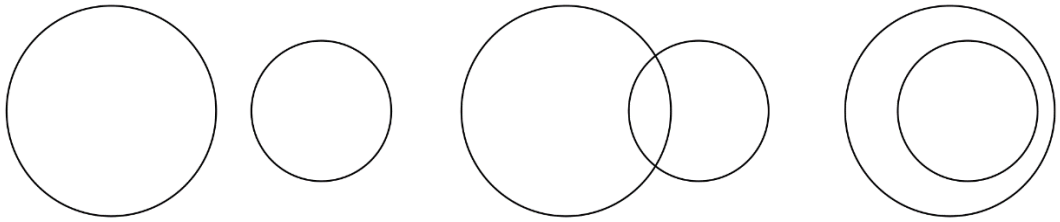


Figure 7: The types of eclipses. From left to right: no eclipse, partial eclipse, maximal eclipse.

The most trivial type is no eclipse. It can be identified if the distance between the stars d is greater than the sum of their radii $r_1 + r_2$. The resulting luminosity is just the sum of each star's luminosities.

The luminosity of the system is also easy to calculate for the maximally eclipsing system. This scenario is split into a further two types, one where the eclipsing star is bigger than the resized star and one where it is smaller. If the eclipsing star is larger than the star being eclipsed ($r_1 \geq r_2$) and $d \leq r_1 - r_2$ then the further star is completely eclipsed meaning all the light reaching the observer is emitted by the closer star, so the total luminosity is just the luminosity of the closer star. In the case where $r_1 < r_2$ and $d \leq r_2 - r_1$ then the closer star is maximally eclipsing the closer star; however, the further star can still be seen. The proportion of the eclipsed star that isn't covered A is given by

equation (12). Using this proportion, the total luminosity $L = L_1 + AL_2$. Where L_1 and L_2 are the luminosities of the eclipsing and eclipsed stars respectively.

$$A = \frac{\pi r_2^2 - \pi r_1^2}{\pi r_2^2} \tag{12}$$

The final scenario is the hardest to calculate as it requires some relatively complex geometry. The luminosity is still given by $L = L_1 + AL_2$, however calculating A requires a very different approach.

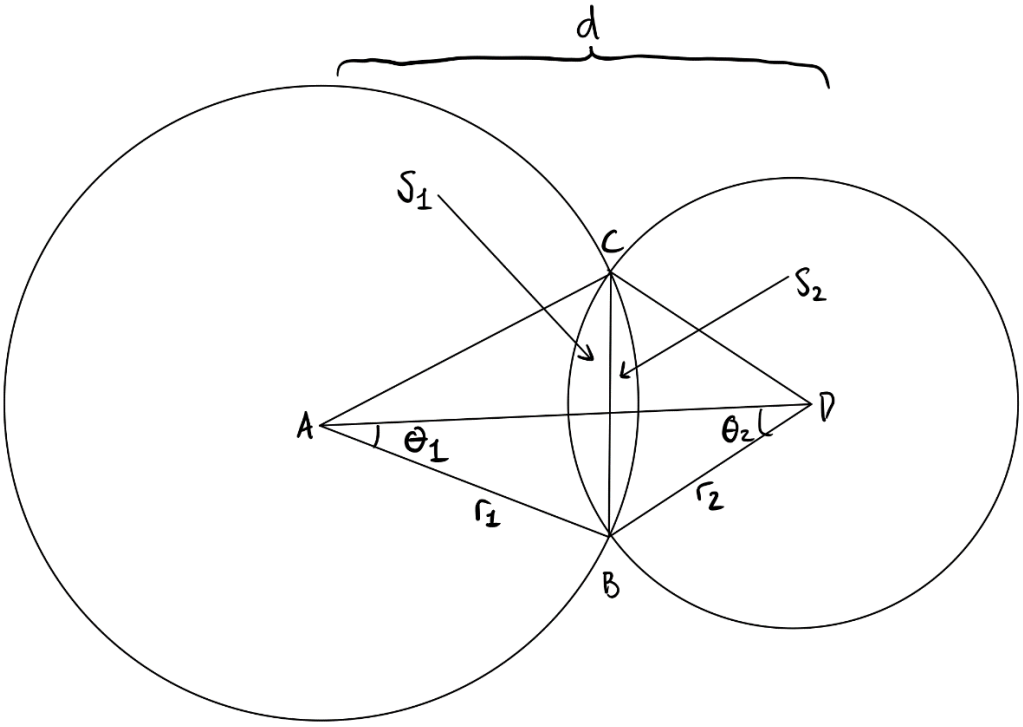


Figure 8: Partial eclipse.

$$A = \frac{\pi r_2^2 - (S_1 + S_2)}{\pi r_2^2} \tag{13}$$

To find the segment areas S_1 and S_2, the areas of ΔABC and ΔDBC (A_{ABC} and A_{DBC}) and of the sectors ABC and DBC ($\widetilde{A_{ABC}}$ and $\widetilde{A_{DBC}}$) need to be calculated. I will only calculate the area S_2 but the calculations for S_1 are almost identical and are left as an exercise to the reader.

First, we must find the angle θ_1 using cosine rule as seen in equation (14). From this it is easy to calculate A_{ABC} and $\widetilde{A_{ABC}}$ as you can see in equations (15) and (16). The area of the segment is given in equation (17).

$$\theta_1 = arccos\left(\frac{d^2 + r_1^2 - r_2^2}{2dr_1}\right) \tag{14}$$

$$A_{ABC} = \frac{1}{2} r_1^2 \, sin(2\theta_1) \tag{15}$$

$$\widetilde{A_{ABC}} = r_1^2 \theta_1 \tag{16}$$

$$S_2 = \widetilde{A_{ABC}} - A_{ABC} \tag{17}$$

By checking at each time step which of these three scenarios is occurring and calculating the luminosity of the system accordingly, we can plot a graph of luminosity against time that closely resembles one of a real eclipsing binary star.

4 Analysis of Simulation Data

In this section I check the validity of my simulation by first checking that it obeys Kepler's second law. Following this I investigate the factors that effect the shapes of the redshift-time and luminosity-time graphs such as the velocities, masses, and eccentricities of the stars.

4.1 Agreement with Kepler s Laws

Kepler's second law states that a line from a celestial body to the centre of mass about which it orbits traces the same area in the same period of time. From this it follows that travel faster when it is closer to the centre of mass than it does when it is further away from the centre of mass. We already saw that this is true in Figure 4 which showed a plot of two stars orbiting each other. However, is this change in speed in accordance with Kepler's second law?

The first step in answering this question is to find the area traced by the line. To do this I used a triangular approximation since the stars already moved in discrete steps. So by finding the area of n successive triangles formed by the star's previous position, the star's current position, and the COM you can find the area traced by the star after n time steps. To find the area of each triangle I used the following equation [8] where A_x is the x-coordinate of point A and B_y is the y coordinate of point B and so on:

$$Area = \left| \frac{A_x(B_y - C_y) + B_x(C_y - A_y) + C_x(A_y - B_y)}{2} \right| \tag{18}$$

If we let A_k be the area of the kth triangle, then the area traced after n time steps is:

$$\sum_{k=1}^{n} A_k$$

Using this method, I was able to get a list of areas traced every n time steps. From this I calculated the maximum deviation from the average area traced as a percentage.

Provided that the length of each time step was sufficiently short[2], I found that the maximum deviation was always negligible (around 1%) suggesting that Kepler's second law was being obeyed.

4.2 Redshift

The investigation I made regarding redshift was how the angle at which the binary star was observed affected the shape of the redshift time graph at different orbital eccentricities.

At eccentricities close to 0 the stars are orbiting at near constant speeds in a circular shape. As a result, you would expect the shape of the redshift time graph to be sinusoidal as the recessional velocity of the stars only depends on the angle at which the star is moving. This is confirmed in Figure 9. In the figure we can also see that the more massive star has a much smaller maximum redshift. In fact, it is inversely proportional to the mass of the star since the larger star is 5 times heavier and has a maximum redshift that is 5 times smaller. This is still roughly true at higher eccentricities; however, the deviation from a fifth the maximum redshift of the lighter star is greater.

---

[2] A longer time step length would let me simulate a longer actual time but in less detail. This normally leads to more orbits being simulated but much less accurately.

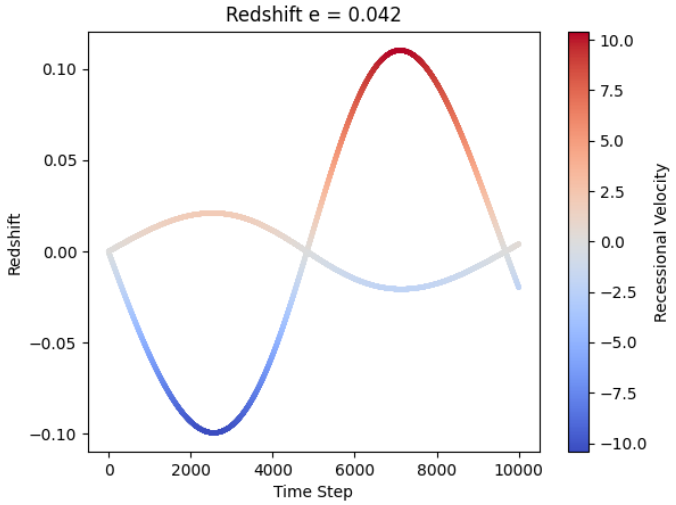


Figure 9: Redshifts at e = 0.042

Due to the symmetry of the orbits the angle at which the binary star is observed has very little effect on the shape of the graph. For this reason, I will exclude these graphs. At higher eccentricities, the graphs become far more interesting, this is mainly due to the variable radial velocities of the stars that causes the sin graph to change shape. In Figure 10 you can see the redshift of the outer star as seen from different angles.

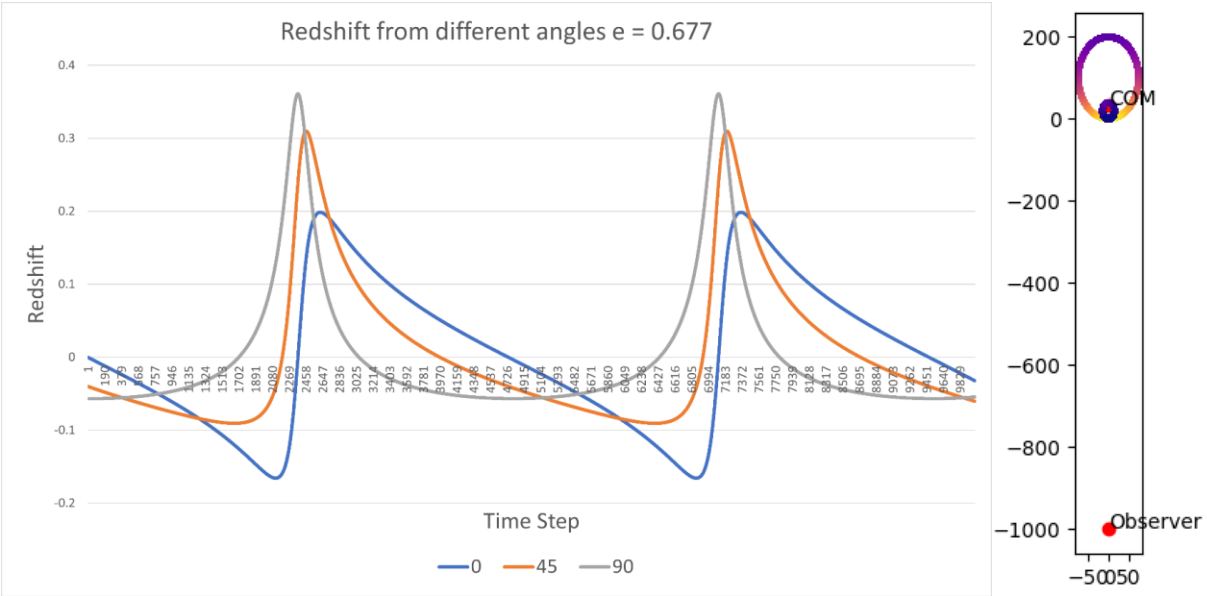


Figure 10: Redshift at different angles from the vertical. 0 degrees shown on right.

As you can see from the graph, being perpendicular to the semi-major axis[3] increases the extremes in redshift. This is because the maximum recessional velocity is very close to the maximum velocity of the star at the apoapsis; therefore, it also has the highest redshift. The rest of the curve is much closer to 0 since the stars are travelling almost perpendicular to the observer making the recessional velocity smaller. From the graphs we can also deduce that the outer star is travelling away from the observer, who is to the right of the system, at apoapsis since the redshift is positive when it is furthest from 0.

Overall, at higher eccentricities the graph of redshift against time is the least extreme when the observer is parallel to the semi-major axis. It is also possible to estimate the relative masses of the stars by measuring their redshifts. Their exact masses can also be found by applying Kepler's third law to their orbital period which is the same as the period of the redshift-time graph.

---

[3] The long side of the ellipse.

4.3 Luminosity

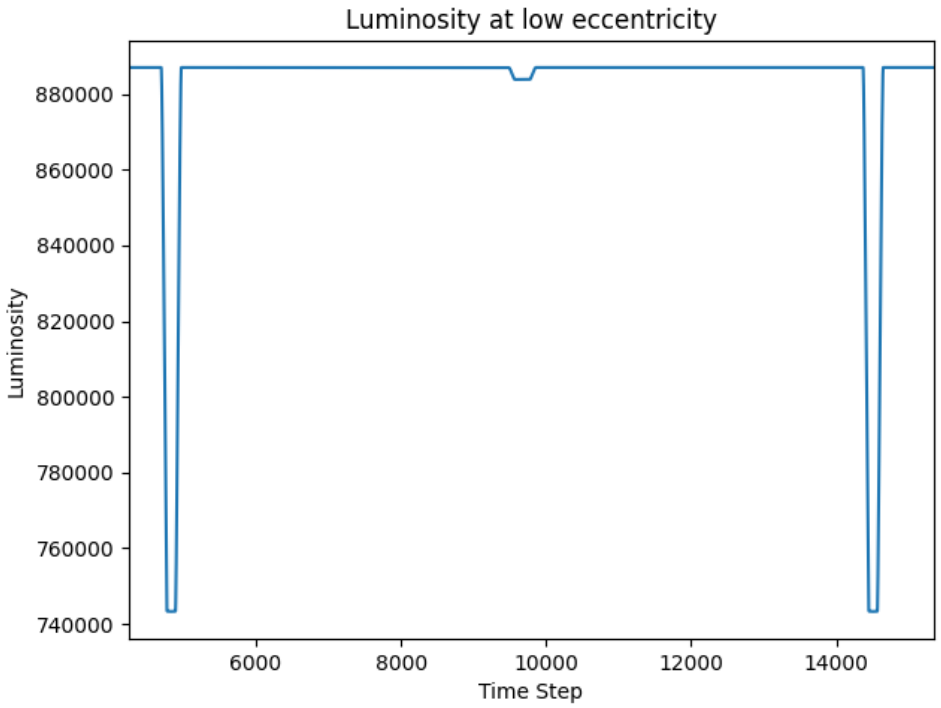


Figure 11: A graph of luminosity against time

In the graph above you can see a plot of luminosity for a binary star with a low eccentricity as these graphs are most useful when it comes to analysing them [9]. Although it is in theory possible to calculate the eccentricity, longitude of periastron, radii, and masses of the stars (when supplemented with the stars' radial velocity), I did not have the time to understand the calculations required to do so. The maths also relied on the stars' fractional radii to be strongly correlated, whereas I just picked mine at random.

However, using the graph above and the simulation I was able to verify that the larger dip did happen when the brighter heavier star was being eclipsed by the dimmer star. As expected, the ratio by which the luminosity fell during each eclipse was roughly the 3.5^{th}

power of the ratio between the masses of the stars. You can also easily extract the orbital period of the stars by simply measuring the time between the first and second primary eclipse[4].

Looking at the graph you can also spot an intriguing pattern upon zooming in on the base of the primary eclipse. This is the fact that the base is not flat but slightly bent as you can see in Figure 12. During the entire period of where the line suddenly changes gradient the more massive star is maximally eclipsed, so why does the luminosity change? What I believe is happening is that this is caused by the eclipsed star moving away and thus appearing smaller while the closer star moves closer making it larger; therefore, the further star becomes more eclipsed over time.

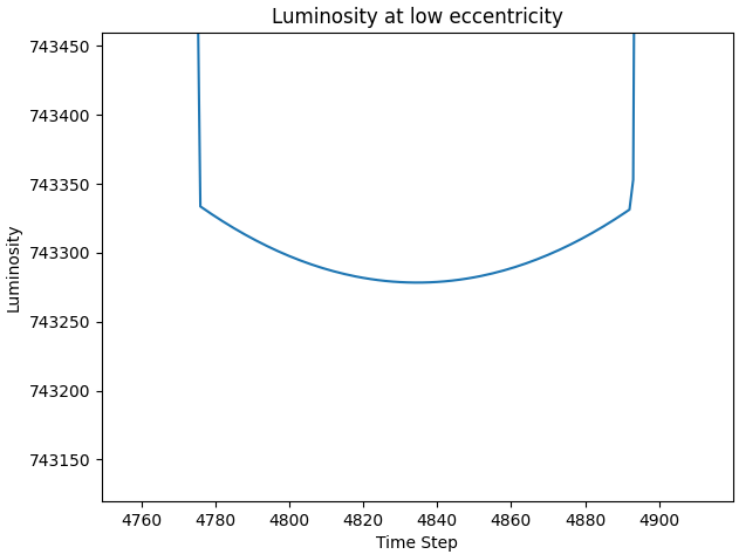


Figure 12: Bend at the bottom of primary eclipse curve

[4] When the brighter star is eclipsed (the larger drop)

5 Conclusion

Overall, my luminosity and redshift graphs had the same shape I expected. Furthermore, the fact that my simulations tended to obey Kepler's laws confirms that it mimics real life accurately; therefore, I would call this experiment a success since real world data about luminosity and redshift could be correlated to one of my simulations to predict properties about stars. However, my simulation had some flaws that anyone who wishes to undertake it could work on. I did not account for the inclination of the observer as it was fixed at 0^{o}, changing this would lead to a more interesting investigation of the luminosity of the system since it could result in total eclipses never occurring. Furthermore, my simulation was slightly unstable since the eccentricities of the two stars, although similar, were not identical like they should have been. Also, further analysis of luminosity data is required. With all these changes this could truly become the perfect investigation; however, I lacked the time to implement them.

6 Bibliography

[1] Space.com Staff, 'Binary Star Systems: Classification and Evolution', *Space.com*. https://www.space.com/22509-binary-stars.html (accessed Jun. 01, 2021).

[2] 'Lecture 13: Binary Star Systems'. https://sites.ualberta.ca/~pogosyan/teaching/ASTRO_122/lect13/lecture13.html (accessed May 24, 2021).

[3] 'Sirius | Facts & Location', *Encyclopedia Britannica*. https://www.britannica.com/place/Sirius-star (accessed Jun. 01, 2021).

[4] CrashCourse, *Binary and Multiple Stars: Crash Course Astronomy #34*. 2015.

[5] L. Bergström and A. Goobar, *Cosmology and Particle Astrophysics*, 2nd ed. Springer, 2006, pp. 77, Eq.4.79.

[6] '5.8: Doppler Effect for Light', *Physics LibreTexts*, Nov. 02, 2016. https://phys.libretexts.org/Bookshelves/University_Physics/Book%3A_University_Physics_(OpenStax)/Book%3A_University_Physics_III_-_Optics_and_Modern_Physics_(OpenStax)/05%3A__Relativity/5.08%3A_Doppler_Effect_for_Light (accessed May 21, 2021).

[7] 'Mass Luminosity Relationship'. https://www.cliffsnotes.com/study-guides/astronomy/the-hertzsprung-russell-diagram/mass-luminosity-relationship (accessed May 28, 2021).

[8] 'Area of a triangle (Coordinate Geometry) - Math Open Reference'. https://www.mathopenref.com/coordtrianglearea.html (accessed Jun. 05, 2021).

[9] J. Southworth, 'Eclipsing Binary Stars: the Royal Road to Stellar Astrophysics', p. 8.

7 Appendix A

7.1 The Celestial Body Object

The celestial body is initialised as follows:

```
class celestialBody:
    def __init__(self, mass, radius, initial_velocity, initial_position, G, time_step):
        k = 1 # The constant of luminosity
        self.mass = float(mass)
        # The brightness of the star, proportional to the 3.5th power of the mass.
        self.luminosity = k * self.mass**3.5
        # Radius of the star, needed when calculating the luminosity of the system.
        self.radius = float(radius)
        # Stars positional properties.
        self.initial_velocity = initial_velocity
        self.velocity = initial_velocity
        self.position = initial_position
        # Constants of the universe.
        self.G = G
        self.time_step = time_step
        # Stores the star's position and velocity at every frame.
        self.history = []
```

Its movement is defined using the following two methods:

```
# Changes the velocity of the star based on the force acting on it
# by all the other stars.
def update_velocity(self, all_bodies):
    # Loops over all the celestial bosies in the simulation
    for body in all_bodies:
        # If the body is not itself
        if body != self:
            # Calculates the acceleration from the gravitational force exerted on it.
            rsquared = (self.position - body.position).dot(self.position - body.position)
            force_dir = self.normalize(self.position - body.position) # The direction of the force (unit vector)
            acceleration = force_dir * self.G * body.mass / rsquared # The acceleration as a vector
            # Updates the velocity using the acceleration.
            self.velocity -= acceleration * self.time_step

# Updates the position by using the velocity.
def update_position(self):
    # Adds the current position to the history
    self.history.append([self.position[0], self.position[1], np.sqrt(self.velocity.dot(self.velocity)), self.velocity])
    # Updates the position based on the current velocity
    self.position += self.velocity * self.time_step
```

The area inscribed by the star every n time steps was calculated using these methods:

```
# Calculates the area of a traingle with vertices A,B, and C.
def area_triangle(self, A, B, C):
    return np.absolute((A[0]*(B[1]-C[1]) + B[0]*(C[1]-A[1]) + C[0]*(A[1]-B[1]))/2)

# Calculates the area incribed by the star in a certain time for the entirety
# of the star's lifetime.
def area_inscribed(self, time, com):
    areas = []
    curr = 0
    nex = 1
    while True:
        area = 0
        for i in range(time):
            curr_pos = self.history[curr]
            try:
                nex_pos = self.history[nex]
            except:
                return areas
            curr += 1
            nex += 1
            area += self.area_triangle(com.position, curr_pos, nex_pos)
        areas.append(area)
```

The orbit's eccentricity was found using this class method:

```
# Gets the distance to apoapsis and periapsis and uses it to calculate eccentricity.
def get_apep(self, com):
    # Extracts just the positions of the stars relative to the COM from it's history.
    positions = [np.array(position[:2]) - com.position for position in self.history]
    # Calculates the magnitude of these positions.
    distances = [np.linalg.norm(position) for position in positions]
    # Finds the distance to apoapsis and periapsis and uses it to find e.
    max_d_index = np.argmax(distances)
    min_d_index = np.argmin(distances)
    self.ap_distance = distances[max_d_index]
    self.pe_distance = distances[min_d_index]
    self.e = (self.ap_distance - self.pe_distance)/(self.ap_distance + self.pe_distance)
```

7.2 Running the simulation

The actual simulation was quite simple to run and was just done by running the following code after defining some parameters, such as the stars, beforehand, for more detail refer to the full source code on GitHub.

```
for i in range(simulation_steps):
    # Updates the velocity of all the stars
    for star in all_stars:
        star.update_velocity(all_stars)

    #Focuses all the stars if focus_on[0] is true.
    if focus_on[0]:
        functions.focus(all_stars, focus_on[1])

    # Updates the position of all the stars
    for star in all_stars:
        star.update_position()

    # Moves COM
    com.update_position()
    com.update_velocity()
    luminosities.append(functions.calculate_luminosity(all_stars, observer_position))
```

7.3 Important Functions

There were some important functions that I wrote in a separate "functions.py" file to clean up my simulation code. This is where the bulk of the calculations was done. However, I will only discuss the ones needed for the redshift and luminosity measurement and omit the miscellaneous functions.

This code calculates the redshift using the method outlined in section 3.2:

```
# Calculates the redshift given the spped of light and the recessional
# velocity.
def calculate_redshift(c,v):
    numerator = np.sqrt(c+v)
    denominator = np.sqrt(c-v)
    redshift = numerator/denominator - 1
    return redshift

# Given the position of the observer and information about the star it calculates
# the recessional velocity and uses that to find the redshift.
def get_redshifts(observer_position, star, c = 100):
    star_positions = [[star.history[i][0],star.history[i][1]] for i in range(len(star.history))]
    star_velocities = [star.history[i][3] for i in range(len(star.history))]

    redshifts = np.zeros(len(star_positions))
    recessional_velocities = []

    for i, position in enumerate(star_positions):
        relative_position = np.array(position) - observer_position
        mag_position = np.linalg.norm(relative_position)
        star_velocity = star_velocities[i]
        recessional_velocity = star_velocity.dot(relative_position)/mag_position
        redshifts[i] = calculate_redshift(c, recessional_velocity)
        recessional_velocities.append(recessional_velocity)

    return redshifts, recessional_velocities
```

This code is used for the luminosity simulation. It's an implementation of my method mentioned in section 3.3.

```
# Calculates the brightness of the star system at a given time. BROKEN
def calculate_luminosity(all_stars, observer_position):
    # Finds which star is closest and arranges them in increasing order of distance.
    # Only stores the stars relative position, radius, and luminosity.
    distances = [np.linalg.norm(star.position - observer_position) for star in all_stars]
    sorted_stars = []
    for distance, star in sorted(zip(distances, all_stars)):
        sorted_stars.append([star.position - observer_position, star.radius, star.luminosity])
    distances = sorted(distances)

    # "Resizes" the further star so that it is the same distance as the closer star
    d1_d2 = distances[0]/distances[1] # closer_distance/further_distance
    sorted_stars[1][1] = sorted_stars[1][1]*d1_d2
    sorted_stars[1][0] = sorted_stars[1][0]*d1_d2

    # The distance d between the closer star and the resized star
    d = np.linalg.norm(sorted_stars[0][0]-sorted_stars[1][0])

    # Checks if the closer star is covering the further star,
    # if it isn't then the luminosity doesn't change.
    if d >= sorted_stars[0][1] + sorted_stars[1][1]:
        return sorted_stars[0][2] + sorted_stars[1][2]

    # Other values needed to calculate coverage
    r1 = sorted_stars[0][1]
    r2 = sorted_stars[1][1]

    if r1>=r2:
        if d <= r1 - r2:
            return sorted_stars[0][2]
    else:
        if d <= r2-r1:
            proportion_uncovered_2 = 1 - (np.pi * r1**2)/(np.pi * r2**2)
            return sorted_stars[0][2] + sorted_stars[1][2]*proportion_uncovered_2

    # Calculates what proportion of the further star is covered by the
    # closer star.
    theta1 = cosine_rule(r2,r1,d)
    theta2 = cosine_rule(r1,r2,d)

    area_triangle_1 = 0.5*(r1**2)*np.sin(2*theta1)
    area_slice_1 = theta1 * r1**2
    area_section_1 = area_slice_1 - area_triangle_1

    area_triangle_1 = 0.5 * (r2**2) * np.sin(2*theta2)
    area_outer_slice_2 = r2 ** 2 * (np.pi - theta2)
    area_uncovered = area_outer_slice_2 + area_triangle_1 - area_section_1
    proportion_uncovered_2 = area_uncovered/(np.pi * r2**2)

    return sorted_stars[0][2] + sorted_stars[1][2]*proportion_uncovered_2
```