Chapter 1

PHOEBE and QX Cas -

Kshitij Duraphe[a]
[a]College of Engineering Pune

PHOEBE is a tool for analyzing binary star systems and is one of the best in the business. Working with a combination of neural networks and classical methods and the fact that PHOEBE has an easy-to-use interface means that PHOEBE is relatively easy to use. However, the learning curve is steep, and it takes time for a complete beginner to find out what exactly PHOEBE's doing. Applying PHOEBE to QX Cas, an eclipsing binary in Cassiopeia, did give very fruitful results. Further analysis of QX Cas revealed that the system is probably a contact binary now, which cannot be analyzed well by PHOEBE

Keywords: PHOEBE, binary stars, light curves

1 Foreword

This report is meant to be read like a book and is meant for people who are completely new to binary star analysis and PHOEBE. A polished version of this report is found on my Github.

2 Introduction

A binary star system is a type of star system in outer space where two stars orbit around a common center of mass, called the *barycenter*. These two stars are *gravitationally bound* to each other. Many stars in the observable universe are part of binary systems. In fact, many stars are not limited to binary systems, but can be part of triple, quadruple, or general multiple star systems.

The orbital periods of binaries and distances of binaries vary greatly. Some stars can orbit their center of mass in a few days, but can also take a few hundred

years to complete a single revolution. Some systems are so close that the stars practically touch each other.

2.1 Why are Binary Stars important?

Binary stars are important because they form what is called the *two-body problem*, that is, if we know that two abstract objects exert some force on each other, we can predict their motion *exactly* by using classical mechanics. Of course, this statement is not entirely true in real life, because every object exerts a force on every other object. You cannot ignore the effect of other objects. This poses a difficult conundrum: If we need to analyze every other object, how in the world can we make accurate predictions for our two bodies? In that case, how can we predict the motion of binary stars? The answer to this is simple. The force between two celestial objects, on the most general scale, is the force of gravity, which drops off over distance according to the square of the distance between two interacting objects *and* relies on the interacting bodies being massive enough to actually have a visible effect. That's why we ignore the effect a box can have on an apple close by; the Earth's pull is simply too large for these two objects to have a visible effect on each other! Note that I said *visible*; in theory, the apple and the box do interact, but if you sit down and work out the mathematics of the situation, you'll find that it'll take you more than the estimated lifespan of the universe to actually see a visible change in their position!

Coming back to binary stars: analyzing them is equivalent to solving a two-body problem. Space is unbelievably empty. Stars are very likely to not encounter anything gravitationally strong enough to affect them for their entire lifetimes. A very large amount of binary stars fall into this category, so it is entirely possible to predict their motion for the remainder of their lives.

Now that we've established that binary stars are a two-body problem, we must note that any system which has more than two stars is *unsolvable*. For three stars, you encounter the three-body problem. Simply analyzing the three-body problem tells us that there exists *no* solution at all! Caveat: you can *approximate* the solution, and in fact, this is exactly how you predict the motions of systems where the number of interacting bodies is greater than 2. All that attempting to solve the three-body problems tells us that there is no solution according to standard mathematical equations that vary in a way that is currently known (such a solution is called *analytic*). Extend this to the $n-body$ problem and we can mathematically prove that the solution is unattainable for the vast majority of cases.

Because you can calculate the orbit of binary stars, you can estimate their masses. If you can estimate their masses, you can make a decent guess at their radius and density using empirical relationships. There are also several ways to guess at the masses of single stars using empirical relationships between the mass of the star and the star's luminosity.

Binary stars provide a way for astronomers to analyze star systems far away from the solar system. Analyzing them may also give insight into the large-scale structure and evolution of the universe as a whole. Thus, it's very important to study binary stars!

2.2 The types of observable binaries

Binary stars systems can be of several types. There's ways to classify binary star systems according to their *configuration*, and there's ways to classify binary star systems according to the way they were observed. Practically, determining the configuration of a binary star system you observed is very difficult, so we classify stars observationally.

2.2.1 Visual Binary

A visual binary star is a double star system that can be resolved with a telescope. Be careful here - visual binaries may not always be in a binary star system. They may just lie on the same line of sight. Stars in visual binary systems may orbit each other, and if they do, generally at a distance of a few hundred AU. These stars do not draw material off each other because they are so far apart, but are gravitationally bound. Examples of visual binaries include stars in the Centauri system, such as α Centauri, the closest star system to Earth. α Centauri is in fact a binary star system, with two stars, α Centauri A and α Centauri B, orbiting each other with a period of approximately 80 years. The distance between them is approximately 23 AU.

2.2.2 Spectroscopic Binary

Many binary systems are simply too far away to be resolved by optical telescopes. Other binary systems may be too close in order to successfully resolve them. Such binaries are analyzed using *Spectroscopic Data*. Of the closer systems, many have orbital periods within the range of a few days or months. To analyze them, you detect spectral shifts in their Doppler lines. The spectral data you obtain from a single observation is a combination of the spectra of both stars. If one of the stars is moving away from us, then we observe shifting towards the redder

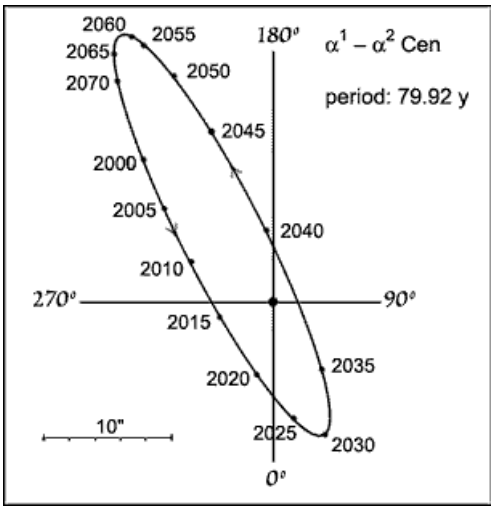

Figure 1: Observations of α Centauri

side of the spectrum (called *redshift*). If a star moves towards us, the spectrum shifts to the bluer side (called *blueshift*). As the stars continue orbiting, we will eventually observe these spectra shift the opposite way. This means that the star that initially moved away from us is now moving towards us and vice versa. This allows us to predict the velocity of stars using the well-known formula for Doppler-shifting:

$$f_o = \sqrt{\frac{1 - \frac{v}{c}}{1 + \frac{v}{c}}} f_s \tag{1.1}$$

where f_o is the observed frequency (or reciprocal of the observed wavelength), f_s is the frequency of the source (or reciprocal of the source wavelength), v is the velocity of the object being measured *relative to the observer*, and c is the speed of light.

(Note that spectroscopic data is measured at wavelengths, in which case you need to take the reciprocal of the equation and solve.)

There are obviously a few things we need to consider when measuring the redshift of binary star systems. The first and most important thing to note is that the *orbital plane* of the binary star system has to be inclined (this will be explained later on). If the system is at a right angle towards us, we will not observe any Doppler shift. The inclination of a binary star system is very important when analyzing the data. The second thing to consider is that you need to separate the observed spectrum into its individual components. You need the individual

spectra for the stars in a binary system to accurately determine their orbital velocities.

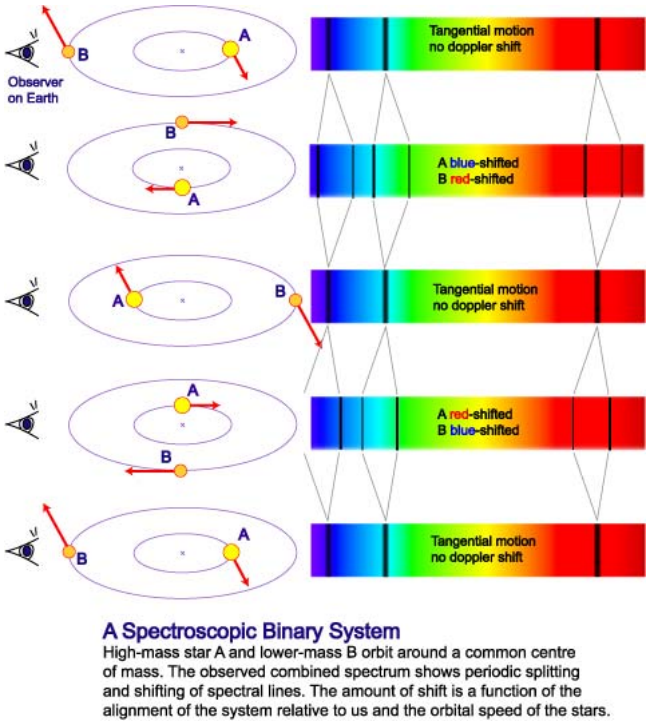

Figure 2: Observing a spectral binary

Luckily, many binary stars observed through spectroscopy revolve around their barycenter in orbits that have a very low eccentricity, due to tidal forces between the stars circularizing their orbits. This means that we can approximate their orbits as circular and use very standard mathematical methods in order to calculate the orbit and guess star parameters within acceptable error ranges.

2.2.3 Eclipsing Binaries

Many stars show a change in their apparent magnitude when observed over time, and that change is periodic. This could be due to several reasons. One main reason involves internal forces and mechanisms of the star changing its intrinsic luminosity. These stars are called pulsating variables. The second reason is that the observed star is in fact a binary star system with its orbital plane nearly edge-on to us. The changing apparent magnitudes mean that the stars periodically eclipse each other. These stars are called *eclipsing binaries* and are analyzed with the help of a figure called a *light curve*. Light curves are plots of the apparent magnitude of the star versus time. Many light curves do not actually plot the magnitude with respect to time. Instead, the time is converted to the phase of the orbit and the apparent magnitude is plotted. Combining many such observations, taken over large periods of time, into a magnitude-phase curve is known as phase-folding, phase-shifting, or simply just folding. Many eclipsing binaries are spectroscopic, and a few are visual as well. Because of this, they interact with each other and unusual stellar effects come into play. For example, one star may leech material from the other, effectively 'eating' the other star in a process known as accretion.

Figure 3: An artist's impression of accretion in a binary star system

Light curves of binary star systems show two dips. The smaller dip accounts for the hotter star eclipsing the cooler one, and vice versa. The only reason we observe this eclipsing is because these stars are so far away, which means that their angular size is too small to differentiate effectively. This means it does not matter which star is 'bigger': to us, they're both very very small! The diagram below shows an eclipsing binary system which is useful to intuitively understand light curves.

It is reasonably easy to estimate stellar parameters after analyzing light curves. For example, the Stefan-Boltzmann Law allows us to estimate the temperature of

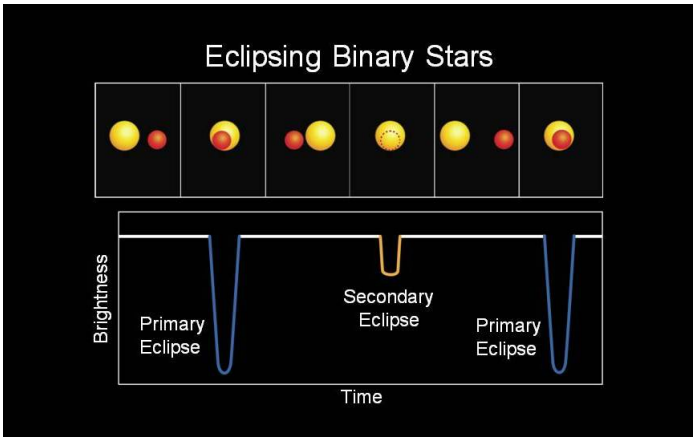

Figure 4: Light Curve of Kepler 16-b

a star from the luminosity. Of course, no star is a perfect black body, and no star will ever be a perfect black body, but reasonable estimates of stellar temperatures allow us to analyze stars very effectively. The important thing to remember is that smaller dips or smaller eclipses occur when the cooler star is eclipsed, and vice versa.

2.2.4 Astrometric Binaries

Let's talk a bit about observations. You can observe stars whenever you want. How do you sync up your observations with other observatories? The answer to this question is that you spend a significant amount of time correcting for the motion of the Earth around the Sun. You can correct for the motion of the Sun around the center of the galaxy, but in general, most observations are taken with respect to the Sun.

Just like the Sun, stars move about freely in space. However, since so many of them are so far away, we're lucky if we notice any significant change in the span of a year or two. Sky surveys allow us to calculate the change in the position of stars with respect to other stars. Stars that are relatively closer to Earth will show significant motion if we get lucky. We cannot measure the velocity of stars through space directly. What we can measure, however, is the component of their velocity *away* from us, and the component of their velocity *transverse* to us. This allows us to get two components of their velocity vector and we can, in theory, calculate their motion exactly. Practically, data over several decades has to be compared to get any reasonable answer, because a decade is a very small

amount of time in astronomy. The motion of the stars relative to other stars is called *proper motion*.

If a star shows a periodic proper motion, we can infer the presence of an unseen companion near it. This is because of gravitational effects. We have a situation where a bright star and a dimmer companion orbit a barycenter. Stars observed this way are called astrometric binaries. These stars are difficult to observe because of the time span required to analyze them. The most famous star is Sirius, which was discovered to be an astrometric binary.

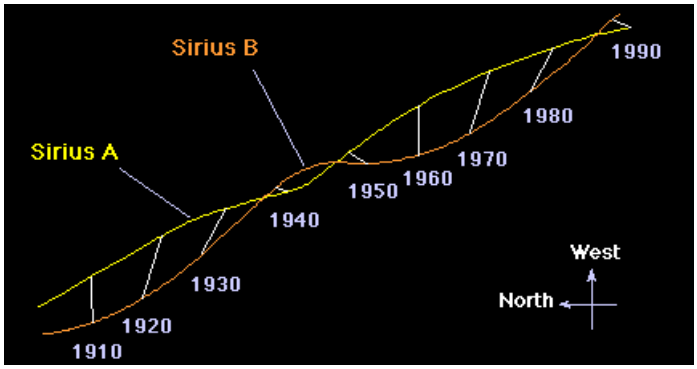

Figure 5: The motion of the Sirius star system

Other exotic types of binary stars, such as binary pulsars, exist, but analyzing them is a specialty in itself.

2.3 How do you analyze a binary star system?

2.3.1 The general problem

Let's look at a general example before making some simplifying assumptions

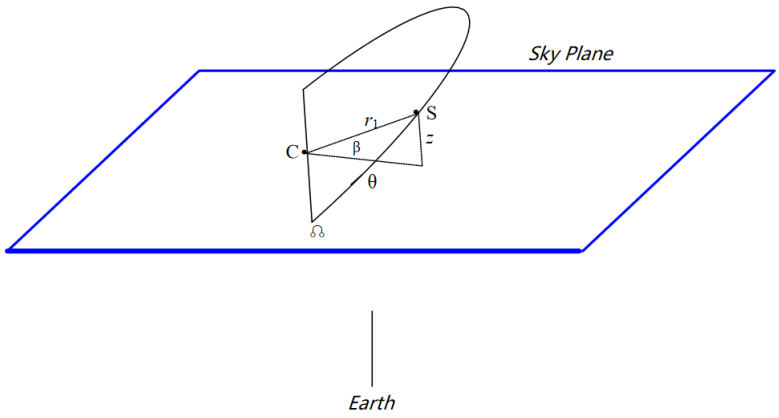

Figure 6: Viewpoint

Assume that you look at only one of the stars in the binary star system. How accurately do you know the position of the star? Effectively pinning down the position of an object in the sky is painstakingly difficult and requires you to know many parameters. The table below explains some of the more esoteric terms used below that a beginner would not know. (These are part of what are called orbital parameters, but for the sake of this report let's assume that a complete beginner is reading it)

Here, the star S orbits the center of mass of the binary star system, labeled C. The 'Sky Plane' is merely the plane normal to the line of observation passing through the center of mass. Assume that S is z units 'above' the plane, its distance from C is r_1, and its argument of latitude is θ. Then we instantly get $z = r_1 sin\beta$. If the orbital plane is aligned at an angle i to the sky plane, then $sin\beta = sinisin\beta$. θ is the sum of the argument of periapsis and the true anomaly, say v, so $z = r_1 sinisin(\omega + v)$. If S is moving at a speed of V_0 with respect to the Sun, then the true velocity becomes $V = V_0 + \dot{z}$. Since the equation of the ellipse can be defined as $r_1 = \frac{a_1(1-e^2)}{1+cosv}$, $\dot{r_1}$ becomes $\frac{r_1 e\dot{v}sinv}{1+ecosv}$.

The angular momentum per unit mass of S for its orbit is $r_1^2\dot{v} = na_1^2\sqrt{1-e^2}$, where n is the mean motion, $\frac{2\pi}{P}$, the time period. Rearranging these equations,

Table 1: Some esoteric terms

Name	**Explanation**
Periastron	The point in an orbit of a celestial body closest to the star it orbits
Ascending Node	The point in the orbit after which the body starts moving 'north' of the reference or sky plane
True anomaly	The angle between the direction of periastron and the current location of the body, both measured from the center
Eccentricity	A unique number that determines the shape of a conic section. Can be defined as the ratio of the sines of two angles, the ratio between distances from a focus and a directrix, or in other ways
Argument of latitude	The sum of the true anomaly and the argument of periapsis

we end up with $V = V_0 + K(cos(\omega + \nu) + ecos\omega$, where K is $\frac{na_1 sini}{\sqrt{1-e^2}}$

Since we know the true anomaly, we can calculate the eccentric anomaly, as the inverse tangent of $\frac{\sqrt{1-e^2}sin\nu}{e+cos\nu}$. By Kepler's formula, we can calculate the mean anomaly, $M = E - esinE$, where E is the eccentric anomaly. As the mean anomaly is $M = \frac{2\pi}{P}(t - T)$, we have an equation for the radial velocity as a function of time.

The calculation of the mass function is trivial. Since $n^2 a_1^3 = GM$, where M is $\frac{m_2^3}{(m_1+m_2)^2}$, K is simply $\frac{Gm_2^3 sin^3 i}{(1-e^2)a_1 sini(m_1+m_2)^2}$, and the mass function $\frac{m_2^3 sin^3 i}{(m_1+m_2)^2}$ can be determined. The *Minimum mass* can be determined by assuming m_1 to be known, as setting $i = \frac{\pi}{2}$.

All of this so far proceeds from the fact that we know the system. That is, we know sufficiently many orbital parameters and values of motion required to calculate the orbit exactly at any point in the past or future (which makes the orbit *deterministic*. We can generate light curves and velocity curves from this system. However the practical process (knowing the light and velocity curves and then determining the orbital elements) is what we truly need to do in order to truly solve for an orbit. The theory developed above explains the orbit in a compact way. However, practically calculating binary data from the orbital curves is an ongoing problem which has no general algorithm to solve it. In theory, we only need to work in reverse in order to determine the orbital and stellar parameters. Practically, the development of a general algorithm solving all binary stars is grounds for a Nobel Prize!

In general, having $V = V_0 + K(cos(\omega + \nu) + ecos\omega$, we can find K, because the difference between the minimum and maximum is $2K_1$. Halve this to get K, and

we can get the value *asini*, the length semimajor axis multiplied by the incline of the orbital plane. Beyond that we cannot do anything unless further assumptions are made.

For the sake of an example, let's make a simplifying assumption. The assumption is that the orbit is circular. The situation then looks something like this:

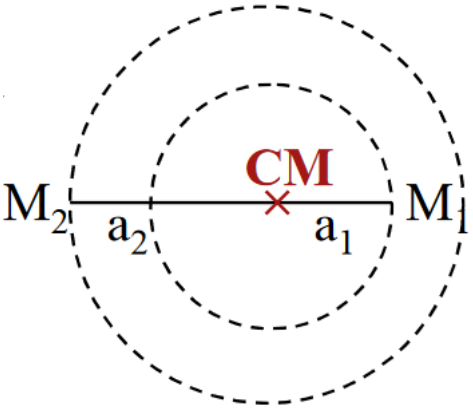

Figure 7: A simple circular binary star system

The only two forces acting in the system are the force of gravitation and the centripetal force, which balances it out. For a stable system, they must be equal. The gravitational force is $F_g = \frac{Gm_1m_2}{r^2}$ and the centripetal force is $F_c = m_2r_2\omega^2$, where ω is $\frac{2\pi}{P}$, P being the orbital period.

Substituting the value of ω and equating the forces, we obtain *Kepler's Third Law for a circular binary star system*, which states $\frac{P^2}{r^3} = \frac{4\pi^2}{G(m_1+m_2)}$, where G is the universal gravitational constant.

From the definition of the center of mass, $m_1a_1 = m_2a_2$. Let $a = a_1 + a_2$. Then $a_2 = \frac{m_1}{m_1+m_2}a$. If this system was a visual binary, you would see each orbit, so you could effectively determine the ratio of masses $\frac{m_1}{m_2}$. If we know the distance to the star system, then we know the angular separation which when combined with P gives us the sum of the masses. This gives us both m_1 and m_2.

If we observe the system spectroscopically, then the situation looks something like this:

The velocities are constant around the orbit. Then we instantly get $Pv_1 = 2\pi a_1$ and $Pv_2 = 2\pi a_2$. We can *only* observe the velocities relative to our line of sight, though, so v_1 becomes $v_1 sini$. The ratio of those observed radial velocities is still

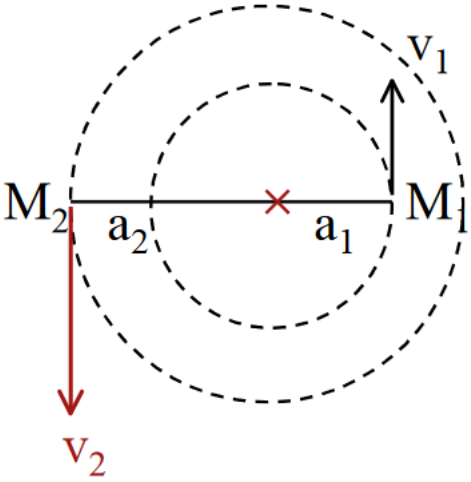

Figure 8: A simple circular binary star system - moving stars

$\frac{m_1}{m_2}$. We can also find the mass ratio if we see spectral lines from both stars. In this case, we do not need to know the inclination. The sum of masses can also be found out in the following manner, by using Kepler's third law:

$a_1 + a_2 = \frac{P}{2\pi}(v_1 + v_2)$

$P^2 = \frac{4\pi^2 a^3}{G(m_1+m_2)} = \frac{(P(v_1+v_2))^3}{2\pi G(m_1+m_2)}$ which leads to $m_1 + m_2 = \frac{P}{2\pi G}(v_1 + v_2)^3$. The cube of *sini* appears in the denominator if we take observed velocities. We can thus determine the sum of masses and the individual masses if and only if the inclination of the system is known.

Eclipsing binaries exhibit specific features in the observed data that are easily recognized because of their geometrical orientation. Eclipses can, in principle, be used to obtain the following information:

- Stellar radii and stellar masses. Methods such as interferometry and occultation are much less practical because interferometry is sensitive to distance and occultation to the time of recording the observation.
- The shape of the eclipse itself constrains the inclination.
- Eclipse duration and their relative positions constrain eccentricity, argument of periastron, and the sum of the radii of both components in terms of their separation.

- The ratio of eclipse depths indicates the degree of surface coverage and the relation between the stars' surface temperatures.
- The ratio of magnitudes in different passbands describes the amount of interstellar extinction.
- Eclipse timing contains the orbital period and changes in the orbital period if observed over a number of days
- Effects such as gravity darkening, limb darkening, starspots, and reflection all constrain the variation of luminosity on the surface.

3 Observations

Essentially, all of our knowledge of distant celestial objects comes from the analysis of electromagnetic waves radiated by those objects. There is no way to bounce light off any object outside of a few close ones because of the distances involved. All astronomical data is fundamentally united by the time when it was taken. Constructing a unified time scale for historical data is a large ongoing problem in itself which tends to become more tedious as newer standards are introduced.

This problem is tedious not only because of the various ways in which we can keep time, but also because of the corrections that need to be applied to the observations to account for the motion of the Earth and the Sun. Modern astronomical observations also require relativistic effects to be taken into consideration due to how precise they are. For most data sets, their time of recording is synced to a time called the *Heliocentric Julian Date*, or HJD. The time taken for by an imaginary Sun to move across the celestial equator (the great circle of the imaginary celestial sphere on the same plane as the Earth's equator) and come back to the same longitude is called a *mean solar day*. The HJD simply measures the number of mean solar days starting from the noon on January 1, 4713 BC. The center of gravity of the solar system is used as a reference point. This is because light emitted by a star on the opposite side of the Sun as observed from Earth takes around 16 minutes more to reach the Earth, when compared to the amount of time it takes 6 months later. For stellar observations which need to be extremely precise, HJD is used and is calculated by the following formula:

$$HJD = JD(Geocentric) + TR(cos\lambda_s cos\alpha cos\delta + sin\lambda_s(sin\epsilon sin\delta + cos\epsilon cos\delta sin\alpha) \quad (1.2)$$

Here, T is equal to 499: the amount of time taken for light to travel 1 AU. R = 1 AU is the Earth-Sun mean distance. λ_s is the *ecliptic longitude* of the Sun, the angular distance of the Sun along the ecliptic (orbital plane) from a primary direction (generally chosen to be the direction of the Earth-Sun direction vector at vernal equinox). α and δ are the right ascension and declination of the star. ϵ is the Earth's angular tilt, 23.45°.

This equation is approximate; many calculators, scripts, and software packages have their own implementation of the HJD which is far more rigorous. Coefficients for the calculation are given in the annual National Almanacs published by the United States.

HJD corrections also have a significant impact on RVs, because Earth's motion around the Sun must also be accounted for. Generally, heliocentric correction is sufficient for most binary stars, but significant corrections are needed for properly mapping pulsar timings.

4 Simulating Binary Stars

In general, observations of a real-life binary star system only include the system's light and radial velocity curves. Depending on the type of binary star system, the observed curves can vary wildly. Some types of light curves are shown below.

Factors like the orbiting stars' sizes and temperature, the distance between the two stars, the presence of features such as starspots, the eccentricity of the orbit, tidal effects, and accretion all play a role in determining the light curve of stars. Similarly, the radial velocity curves of star systems are heavily affected by the eccentricity of the orbit. Perfectly circular orbits, or orbits that are as close to circular as possible show radial velocity curves that are sinusoidal.

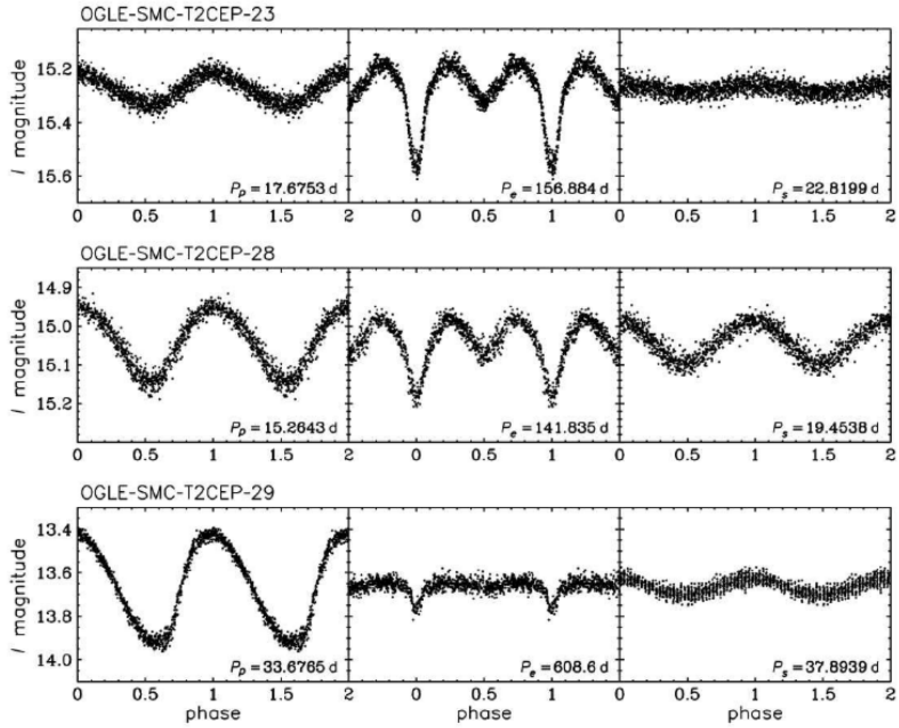

Figure 9: Different types of light curves

However, elliptical orbits cause the light curves to skew and deviate from ideal sine curve shapes.

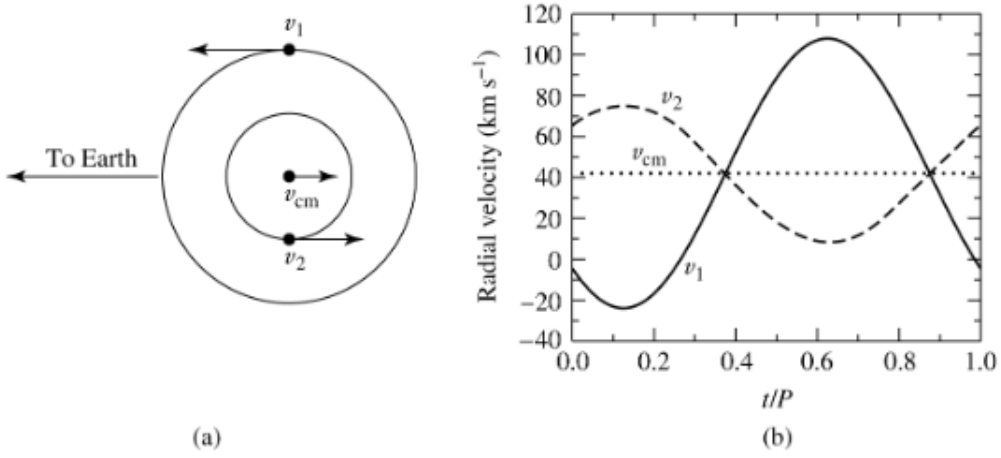

Figure 10: Radial velocity curves for perfectly circular binary star orbits

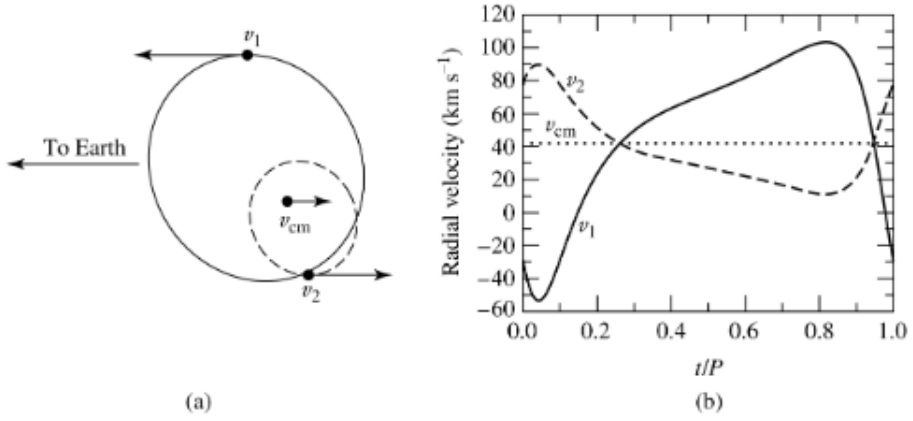

Figure 11: Skewed radial velocity curves for elliptical binary star orbits

It is up to the observers to solve the problem in reverse. Even if one ignores all other parameters and assumes ideal conditions for the stars themselves, the problem is still fundamentally unsolvable, as seen above. Factoring in the different morphologies of different systems, observations that need to be either normalized or synchronized with each other, and the general difficulty in dealing with a large number of parameters affecting the behavior of single stars, there is no general algorithm that results in a unique solution for each system. Each system has to be manually handled in order to produce solutions with the least amount of uncertainty and error. Moreover, different algorithms need to be applied at different stages to get different parameters. Each algorithm may require a different set of input parameters, which have to be obtained from previous parameters. In short, solving this reverse, or *inverse problem* is extremely difficult, time consuming, and computationally expensive.

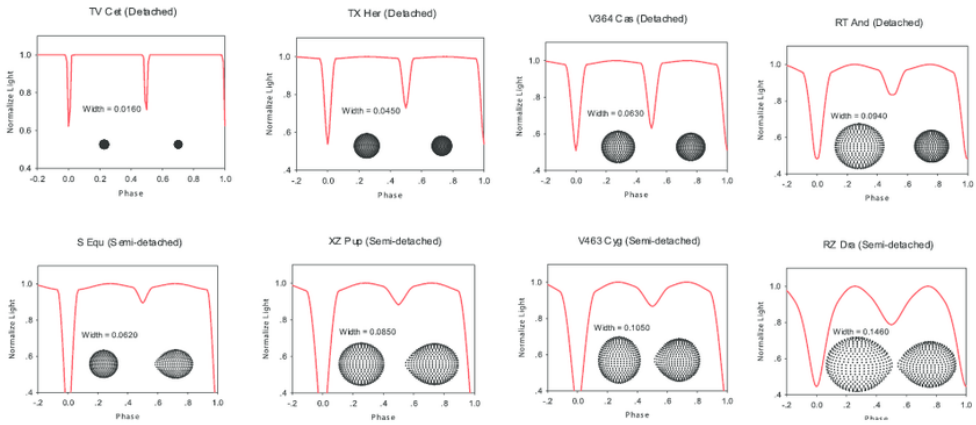

Figure 12: Different types of light curves for different binary star system configurations

The general approach to solving an inverse problem is as follows:

1. Fit a curve to the observational data.
2. Create a synthetic model that results in these curves.
3. Compare and contrast the synthetically generated curve and curves fitted to observational data.
4. Estimate the parameters of the star system from your synthetic model and perform further analysis on the data in order to validate your guesses.

Many institutions and individuals across the world have come up with their own algorithms and programs that create curves to fit data. For example, VARTOOLS, developed by Princeton University, is designed to operate on large sets of light curves using statistical methods. It succeeds previous scripts such as PYKE, PERANSO, and VSTAR, all of which were designed to operate on light curve and are still in use today. Scripts that perform individual analyses, such as PyOrbit and ldtk, are often incorporated into or derived from these tools. A large problem is comparing and contrasting the results obtained by different scripts and selecting the best among them. In recent times, software development has moved towards combining different scripts and using statistical methods and machine learning to choose the most optimal solution among these scripts.

5 PHOEBE - an introduction

PHOEBE stands for PHysics Of Eclipsing BinariEs. PHOEBE is a script written in Python specifically used to generate synthetic models for binary star systems, based on a number of user inputs. PHOEBE considers a large number of parameters in order to generate light curves, radial velocity curves, various animations and plots, as well as 3D models for the binary system given as user input. It combines many historical scripts and uses statistics and machine learning in order to generate its models. In PHOEBE terminology, these models are called *forward* models. In general, PHOEBE users give input data to PHOEBE in order to generate light curves, temperature profiles, and other various data for a given system.

PHOEBE, and especially PHOEBE-2, the rewrite of the original PHOEBE software, has been developed specifically to help with the analysis of Kepler data. For context, the Kepler space telescope collected photometric data from around 150,000 stars for a period of 9 years. Kepler's advanced photometer enabled very precise measurements, necessitating improvements in software to analyze these data. With the wide availability of Kepler and TESS data through easy-to-use interfaces such as the Python package Lightkurve, PHOEBE acts as a tool to make very precise guesses and give insight into these data.

PHOEBE is also built to be flexible and modular. With the way PHOEBE is designed, updating reference tables, physics models, and observables does not require a rewrite of the entire code. PHOEBE's framework is modular: users can pick-and-choose which components they require and which components they wish to ignore for the duration of the computation. PHOEBE is written in Python in order to remove entry barriers and make the software more accessible to newer users. It focuses on ease-of-use, reliability, and simplicity in order to achieve this.

Its original esoteric scripting language is no longer used, and has been replaced with a modern Python interface.

The main data structure of PHOEBE is neither an object or a set of function calls, as is standard for Python, due to it being an object-oriented language. The data structure is instead what PHOEBE calls a *bundle*. A PHOEBE bundle is essentially a collection or a database of parameters, many of which can be attached with observational and synthetic datasets. This is in stark contrast to classical Python code, which relies on objects with methods and attributes, or calling function after function.

6 Simulations using PHOEBE

6.1 The PHOEBE Bundle

A very general overview of simulating a synthetic model using PHOEBE is as follows:

1. Initialize a PHOEBE bundle. For most use cases, phoebe's `default_binary()` bundle is sufficient.
2. Modify (or declare) the parameters of the bundle with known or estimated values, to give PHOEBE a better chance of computing the optimal solution for your observations.
3. Add observational datasets to this PHOEBE bundle.
4. Run the PHOEBE bundle's `run_compute()` method. This will compute the binary system based on the given parameters and will try to fit the data to the observational datasets.
5. Once computation is finished, we can animate the system, plot graphs, and view various results such as the synthetically observed fluxes directly.

Obviously, a great deal of subtlety goes into the actual computation. Let's look at a step-by-step explanation of how to compute your first synthetic model in PHOEBE, along with the code required.

6.2 Simulating PHOEBE's default model

The code below is the bare minimum for simulating a PHOEBE model.

```
import phoebe

b=phoebe.default_binary()

b.run_compute()

print(b.get_value(qualifier='fluxes', context='model')
    )

afig, mplfig = b.plot(show=True)
```

The end result of running this script as a `.py` file will produce the following matplotlib figure.

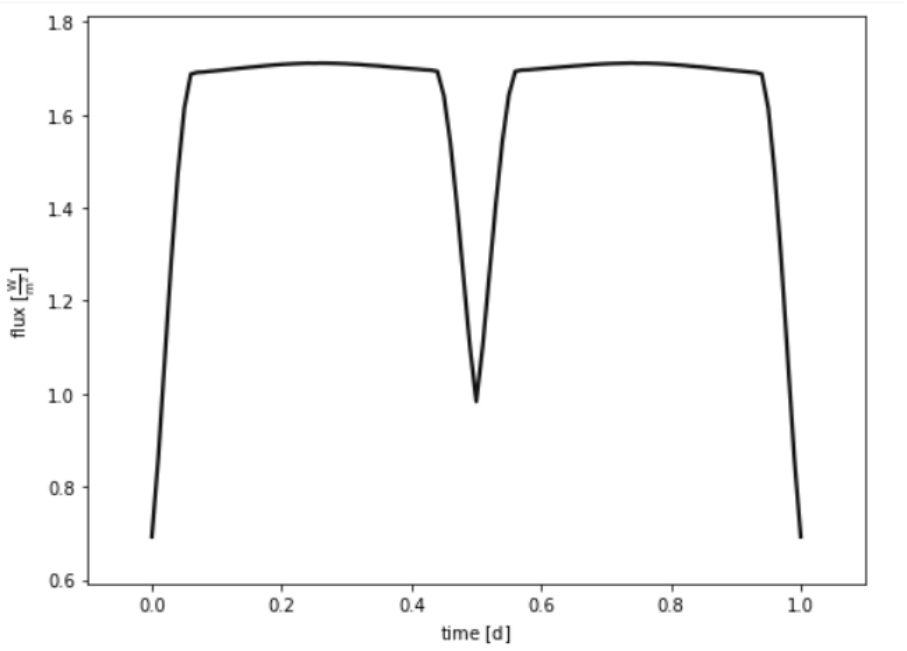

Figure 13: PHOEBE's default binary system

We observe that the y-axis unit is a unit of flux, and the x-axis unit is time. This is misleading, however. PHOEBE's defaults graphs are plots of flux vs the phase of the system. The default model simply happens to have the phase and time period match up on purpose.

We can modify the default model by messing around with the parameters in PHOEBE's default bundle.

What does PHOEBE's default bundle actually contain? Simply running `b` gives the following output:

```
<PHOEBE Bundle: 133 parameters | contexts: component,
    system,
compute, figure, constraint, setting>
```

We see that the PHOEBE bundle contains 133 parameters and it has 6 contexts. It should be noted that the default bundle simply generates a binary star system that has a bunch of default values assigned to it. At first glance, it may not be clear what contexts are in PHOEBE. This is because PHOEBE's approach to modifying individual parameters is unique. In general scripts, accessing parameters usually involves isolating the data tables they belong to, and filtering through that. For example, uniquely identifying an animal, for instances, requires knowing what kingdom, phylum, class, order, family, genus, and species it belongs to. In the same way, identifying a parameter for a general script requires knowing what 'kingdom', 'phylum', and so on it belongs to, and filtering through each and every subset until you isolate the parameter.

PHOEBE's approach is a bit different. While PHOEBE does have a rough hierarchy of sorts, you are able to access each and every parameter in the bundle by filtering through the hierarchy in any way you desire. An outline of PHOEBE's rough hierarchy for the default bundle is:

Contexts
|
Components
|
Qualifiers
|
Keys
|
Parameters

Each parameter has a `context`, a `component`, a `qualifier`, and a `key`. You can access each parameter by filtering down through the tags using `b.filter()` for a bundle `b`. However, PHOEBE makes this process immensely simple by adding *twigs*, which allow you to use the `@` symbol and arrange the tags in the hierarchy *any way you want*, and access parameters that way.

For example, if you want to modify the temperature of the primary star in the system, you could modify the value in three ways:

- You could use `b.set_value(qualifier='teff', component='primary', value=6500)`

- You could use `b["teff@primary"]=6500`
- You could use `b["primary@teff"]=6500`

As you can see, there is a multitude of ways in which you can access this single component, and the order doesn't matter, for most cases. There are a few edge cases where you need to filter through the hierarchy to set values manually, however these cases are not in this report's scope and must be dealt with on a case-by-case basis. Fundamentally, drilling down through the parameter list using the b.filter() inbuilt method is the correct way to approach this, but for practical coding, twigs are much easier.

If we actually set the temperature of the primary star to 6500K, the light curve changes from the default set to this:

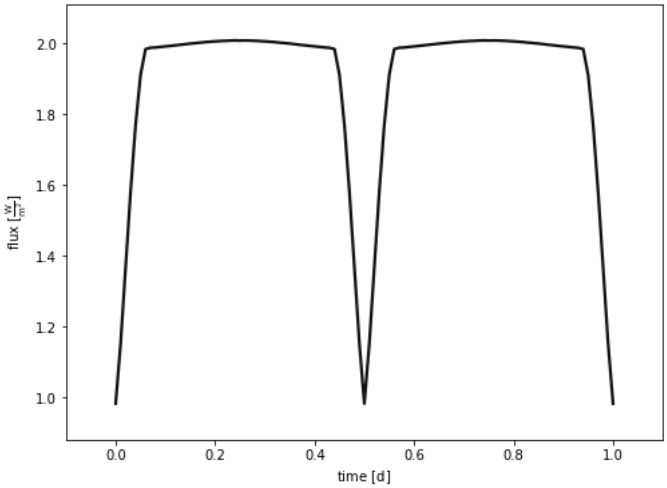

Figure 14: Setting the primary star's temperature to 6500K

Playing around with these parameters can generate a large amount of synthetic models, some of which are shown below.

As an addendum, the code for figure 17 given below. Note how the twigs are purposefully made nonhierarchical, to demonstrate PHOEBE's ability.

```
b["ecc"]=0.22
b["incl@binary@orbit@component"]=40
b["pitch@primary@star@component"]=0.4
b["pitch@secondary@component@star"]=0.5
b["q"]=1.234
```

You can access all the parameters using `print(b.filter().info)`.

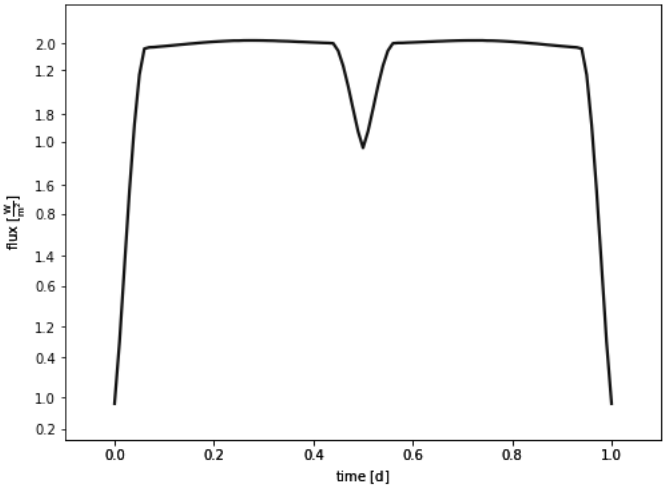

Figure 15: Setting the primary star's temperature to 6500K, secondary to 5000

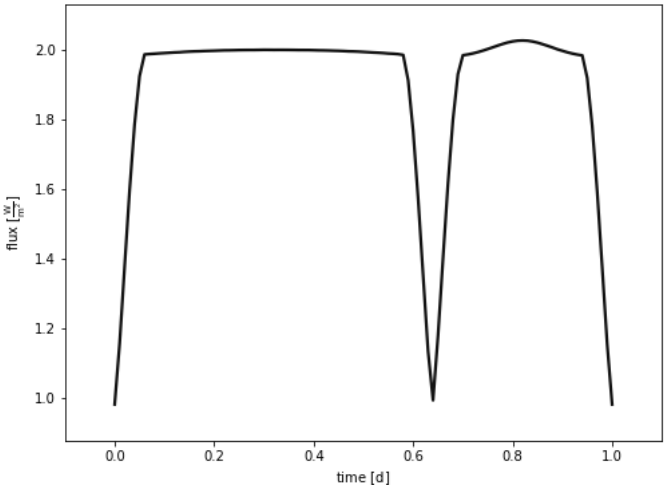

Figure 16: Setting the eccentricity of the orbit to 0.22

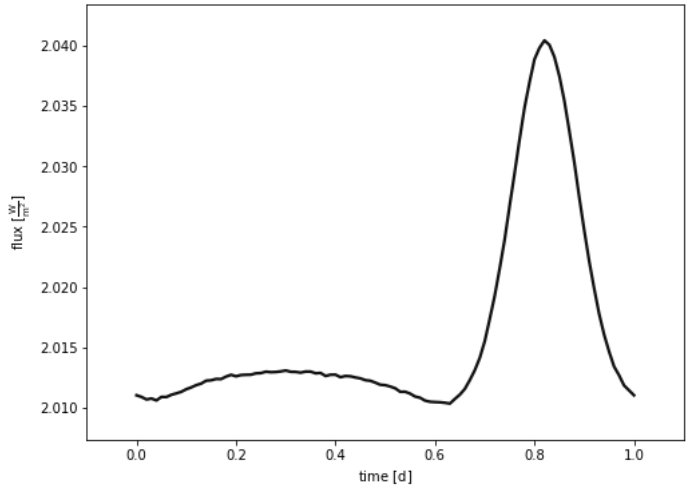

Figure 17: Setting the eccentricity of the orbit to 0.22, the inclination of the orbital plane to 40 degrees, the pitch of the primary star's stellar rotation axis to 0.4, the secondary's to 0.5, and setting the mass ratio as 1.234

Twig	Description
Av	Extinction Av
Rv	Extinction law parameter
abun	Abundance/Metallicity
asini@binary@orbit@component	Projected semi major axis
asini@constraint	expression that determines the constraint
asini@star@component	Projected semi major axis of the component in the orbit
atm	Atmosphere table
auto_add_figure	Whether to automatically add figure parameters when a dataset is added with a new dataset type
auto_remove_figure	Whether to automatically remove figure parameters when the referenced dataset/solution are removed.
boosting_method	Type of boosting method

color@latest@figure	Color to use for figures in which color_source is set to model
color@lc01@lc@figure	Color to use for figures in which color_source is set to dataset
color@star@figure	Color to use for figures in which color_source is set to component
color_source	Source to use for color. For manual
comments@phoebe01@latest@model	User-provided comments for this model. Feel free to place any notes here.
comments@phoebe01@phoebe@compute	User-provided comments for these compute-options. Feel free to place any notes here - if not overridden
compute_phases@binary@lc01@lc@constraint	expression that determines the constraint
compute_phases@binary@lc01@lc@dataset	Phases associated with compute_times.
compute_times	Times to use during run_compute. If empty
contexts	Contexts to include in the plot
datasets	Datasets to include in the plot
dec	Declination
default_time_source	Source to use for highlight/uncover time for any figure in which time_source is set to default.
dict_filter	Filters to use when using dictionary access
dict_set_all	Whether to set all values for dictionary access that returns more than 1 result
distance	Distance to the system
distortion_method	Method to use for distorting stars
dpdt	Time derivative of orbital period (anomalistic)

dperdt	Time derivative of argument of periastron
dynamics_method	Which method to use to determine the dynamics of components
ebv@extinction@constraint	expression that determines the constraint
ebv@system	Extinction E(B-V)
ecc	Eccentricity
eclipse_method	Type of eclipse algorithm
ecosw@binary@orbit@component	Eccentricity times cos of argument of periastron
ecosw@binary@orbit@constraint	expression that determines the constraint
enabled	Whether to create synthetics in compute/solver run
esinw@binary@orbit@component	Eccentricity times sin of argument of periastron
esinw@binary@orbit@constraint	expression that determines the constraint
exptime	Exposure time (time is defined as mid-exposure)
fluxes@lc01@lc@dataset	Observed flux
fluxes@lc01@phoebe01@latest@lc@model	Model (synthetic) flux
freq@binary@orbit@component	Orbital frequency (sidereal)
freq@constraint	expression that determines the constraint
freq@star@component	Rotation frequency (wrt the sky)
fti_method	How to handle finite-time integration (when non-zero exptime)
gravb_bol	Bolometric gravity brightening
hierarchy	Hierarchy representation
highlight	Whether to higlight the current time(s) (see times_source and times parameters).
horizon_method	Type of horizon method
incl@binary@orbit@component	Orbital inclination angle

incl@star@component	Inclination of the stellar rotation axis
incl@star@constraint	expression that determines the constraint
intens_weighting	Whether passband intensities are weighted by energy or photons
irrad_frac_lost_bol@star@component	ratio of incident bolometric light that is lost/ignored
irrad_frac_lost_bol@star@constraint	expression that determines the constraint
irrad_frac_refl_bol	ratio of incident bolometric light that is used for reflection/irradiation (heating without redistribution)
irrad_method	Which method to use to handle all irradiation effects (reflection
l3	Third light in flux units
l3_mode	Whether third light is given in units of flux or as a fraction of total light
latex_repr@figure	Representation to use in place of the component label when rendering latex representations of parameters. If blank
latex_repr@lc01@lc@figure	Representation to use in place of the dataset label when rendering latex representations of parameters. If blank
ld_coeffs_source_bol	Source for bolometric limb darkening coefficients (used only for irradiation
ld_func_bol	Bolometric limb darkening model (used only for irradiation).
ld_mode	Mode to use for limb-darkening
ld_mode_bol	Mode to use for bolometric limb-darkening (used only for irradiation).

legend	Whether to draw the legend
linestyle@latest@figure	Linestyle to use for figures in which linestyle_source is set to model
linestyle@lc01@lc@figure	Linestyle to use for figures in which linestyle_source is set to dataset
linestyle@lcfig01@lc@figure	Default linestyle when plotted via run_figure
linestyle@star@figure	Linestyle to use for figures in which linestyle_source is set to component
linestyle_source	Source to use for linestyle. For manual
logg@star@component	logg at requiv
logg@star@constraint	expression that determines the constraint
long_an@binary@orbit@component	Longitude of the ascending node
long_an@star@component	Longitude of the ascending node (ie. equator) of the star
long_an@star@constraint	expression that determines the constraint
ltte	Correct for light travel time effects
marker@lc01@lc@figure	Marker (datasets only
marker@lcfig01@lc@figure	Default marker (datasets only
marker@star@figure	Marker (datasets only
marker_source	Source to use for marker (datasets only
mask_enabled	Whether to apply the mask in mask_phases during plotting
mask_phases	List of phase-tuples. Any observations inside the range set by any of the tuples will be included.
mass@star@component	Mass
mass@star@constraint	expression that determines the constraint

mean_anom@binary@orbit@component	Mean anomaly at t0@system
mean_anom@binary@orbit@constraint	expression that determines the constraint
mesh_method	Which method to use for discretizing the surface
models	Models to include in the plot
ntriangles	Requested number of triangles (won't be exact).
passband	Passband
pblum	Passband luminosity (defined at t0)
pblum_component	Which component's pblum will be provided
pblum_mode	Mode for scaling passband luminosities
per0	Argument of periastron (defined at time t0@system)
period@binary@orbit@component	Orbital period (defined at t0@system
period@star@component	Rotation period (wrt the sky)
period@star@constraint	expression that determines the constraint
phases_t0	t0 to use when converting between compute_times and compute_phases as well as when applying mask_phases
phoebe_version	Version of PHOEBE
pitch	Pitch of the stellar rotation axis wrt the orbital inclination
q	Mass ratio
ra	Right ascension
requiv	Equivalent radius
requiv_max@star@component	Critical (maximum) value of the equivalent radius for the given morphology
requiv_max@star@constraint	expression that determines the constraint

run_checks_compute	Compute options to use when calling run_checks/run_checks_compute or within interactive checks.
run_checks_figure	Figures to use when calling run_checks/run_checks_figure or within interactive checks.
run_checks_solution	Solutions to use when calling run_checks/run_checks_solution or within interactive checks.
run_checks_solver	Solver options to use when calling run_checks/run_checks_solver or within interactive checks.
sample_from	distributions or solutions to use for sampling. If pointing to a solution
sigmas	Observed uncertainty on flux
sma@binary@orbit@component	Semi-major axis of the orbit (defined at time t0@system)
sma@star@component	Semi major axis of the component in the orbit
sma@star@constraint	expression that determines the constraint
solver_times	times to use within run_solver. All options will properly account for masking from mask_times. To see how this is parsed
syncpar	Synchronicity parameter
t0	Time at which all values are provided. For values with time-derivatives
t0_perpass@binary@orbit@component	Zeropoint date at periastron passage of the primary component
t0_perpass@binary@orbit@constraint	expression that determines the constraint

t0_ref@binary@orbit@component	Zeropoint date at reference point for the primary component
t0_ref@binary@orbit@constraint	expression that determines the constraint
t0_supconj	Zeropoint date at superior conjunction of the primary component
teff	Mean effective temperature
time_source	Source to use for highlight/uncover time for this individual figure (or set to default to respect the default_time_source parameter).
times@lc01@lc@dataset	Observed times
times@lc01@phoebe01@latest@lc@model	Model (synthetic) times
uncover	Whether to uncover up to the current time(s) (see times_source and times parameters).
vgamma	Constant barycentric systemic velocity (in the direction of positive RV or negative vz)
web_client	Whether to default to using the web-client over a locally installed desktop-client when opening the UI from the desktop client.
web_client_url	Default location of web-client. Will only be used if web_client is True.
x	Array to plot along x-axis
xlabel_source	Whether to automatically or manually provide label for the x-axis
xlim_source	Whether to automatically or manually set the x-limits.
xunit_source	Whether to automatically or manually set the x-units.
y	Array to plot along y-axis

yaw	Yaw of the stellar rotation axis wrt the orbital longitude of ascending node
ylabel_source	Whether to automatically or manually provide label for the y-axis
ylim_source	Whether to automatically or manually set the y-limits.
yunit_source	Whether to automatically or manually set the y-units.

As you can see, a lot of parameters exist. Setting them all manually is difficult and time-consuming, and it often happens that we know only a few parameters from this set with any reasonable uncertainty. This is not that big of an issue, because of...

6.3 Constraints

If, in the previous example, you tried to set the effective temperature of one star to 100K, PHOEBE would not run computations on the system. This is because one, stars aren't actually that cold, and two, there are certain reasonable limits that PHOEBE places on the values and how they vary. This is because PHOEBE parameters are related to quite a few others, and it is important to check whether each parameter makes sense or not. This is called *constraining* the parameter, and it is is one of the most vital parts of PHOEBE.

You can find out how each parameter is constrained by running something like

```
b.get_parameter(qualifier='mass', component='primary',
    context='constraint')
```

which results in the following output:

```
<ConstraintParameter: {mass@primary@component} =
    (39.478418 * ({sma@binary@component} ** 3.000000))
    / ((({period@binary@component} ** 2.000000) * ({
    q@binary@component} + 1.000000)) *
    2942.2062175044193281792104244232 18) (solar units)
    => 0.9988131358058301 solMass>
```

What does this tell us? It tells us that the mass of the primary star is calculated by a deterministic formula, which requires us to know the length of the semimajor axis, the orbital period of the binary system, and the mass ratio. You can verfiy that these relations are consistent with the derivation posted above. PHOEBE has inbuilt formulae for many parameters, along with acceptable ranges. The best way to go about understanding these constraints is by continuously working with PHOEBE and making errors. Listing all of the constraints is foolish. Instead, we can find out what parameters constrain a given parameter and what our given parameter constrains. This can be done with Python's all-powerful `print` command, like so:

```
print(b.get_parameter(qualifier='mass', component='
    primary', context='constraint'))
```

which results in the following output

```
Parameter: mass@primary@component
                    Qualifier: mass
                  Description: Mass
                        Value: 0.9988131358058301
                               solMass
              Constrained by: sma@binary@component
                  , period@binary@component,
                  q@binary@component
                   Constrains:
                       logg@primary@component,
                       mass@secondary@component
                   Related to:
                       requiv@primary@component,
                       logg@primary@component,
                       sma@binary@component,
                       period@binary@component,
                       q@binary@component,
                       mass@secondary@component
```

PHOEBE returns the constraints of each parameter and how it constrains others. For the exact formula, you simply run this method on the constrained parameter and see the relation.

An example is shown below:

```
b.get_parameter(context='constraint', kind='star',
    component='primary', qualifier='logg')
```

which results in

```
<ConstraintParameter: {logg@primary@component} = log10
    ((({mass@primary@component} / ({
    requiv@primary@component} ** 2.000000)) *
    2942.206218) * 9.319541) (solar units) =>
    4.389498704454577>
```

We can also *flip* these constraints, based on the input data. For example, if the mass of one of the stars in the binary star system was given in the paper, then we need to solve for the period. This can be done by running:

```
b.flip_constraint('mass@primary', solve_for='period')
```

and the computation can proceed as normal.

6.4 Datasets

As mentioned before, you can add observational datasets to PHOEBE. However, the code snippets given above did not add any dataset. This is because PHOEBE created a dataset by itself according to its preset defaults.

The name dataset is slightly misleading. To put it simply, PHOEBE can run its synthetic model over certain periods of time or phase, either using user-inputted numericals arrays or creating one on its own. If we wish to evaluate the light curves for only a certain phase range, then the way we do it is as follows:

```
import phoebe
from phoebe import u
b=phoebe.default_binary()
b["ecc"]=0.5
b["incl@binary@orbit@component"]=40
b["pitch@primary@star@component"]=0.4
b["pitch@secondary@component@star"]=0.5
b["q"]=1.1

b.add_dataset(phoebe.dataset.lc,
                compute_phases=phoebe.linspace(0,0.5,21)
                    ,
                dataset='lc01')
b.run_compute()
afig,mplfig=b.plot(show=True)
```

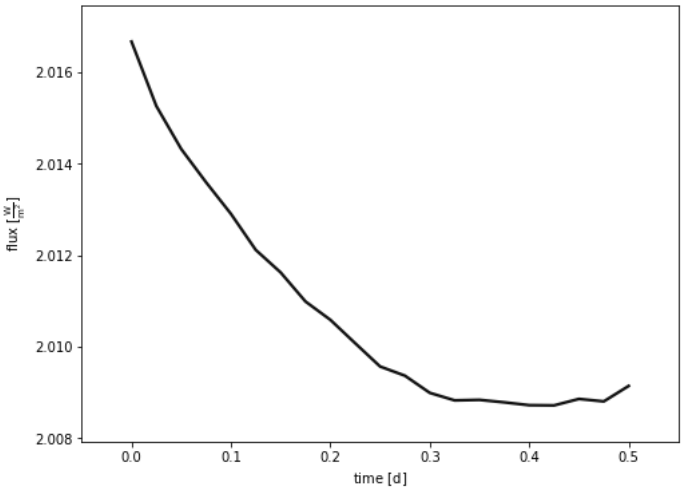

Figure 18: A light curve computed with the following parameters over a phase space ranging from 0 to 0.5

which results in the light curve:

Here, we added a dataset that specifically showed us the light curve. There are five types of datasets available in PHOEBE. These datasets are the light curve, orbit, radial velocities, spectral line profiles, and the meshes. Of these five, the light curve, radial velocities, and spectral profiles are the easiets to understand. A radial velocity curve computed over the same phase space would be as follows:

You can also load datasets with observations. Getting actual observations and processing them to fit with PHOEBE is time-consuming, so for the sake of an example, we will add a random dataset.

```
import phoebe
from phoebe import u
b=phoebe.default_binary(contact_binary=True)
b["incl@binary@orbit@component"]=77
b["q"]=1.1

b.add_dataset('lc', times=[0,1,3,4,5,6,22], fluxes
    =[1,0.5,0.8,0.3,0.9,0.1,0.0525], dataset='lc01',
    overwrite=True)
b.add_dataset('rv',
                times=[0,1,2,5,6,7,9],
```

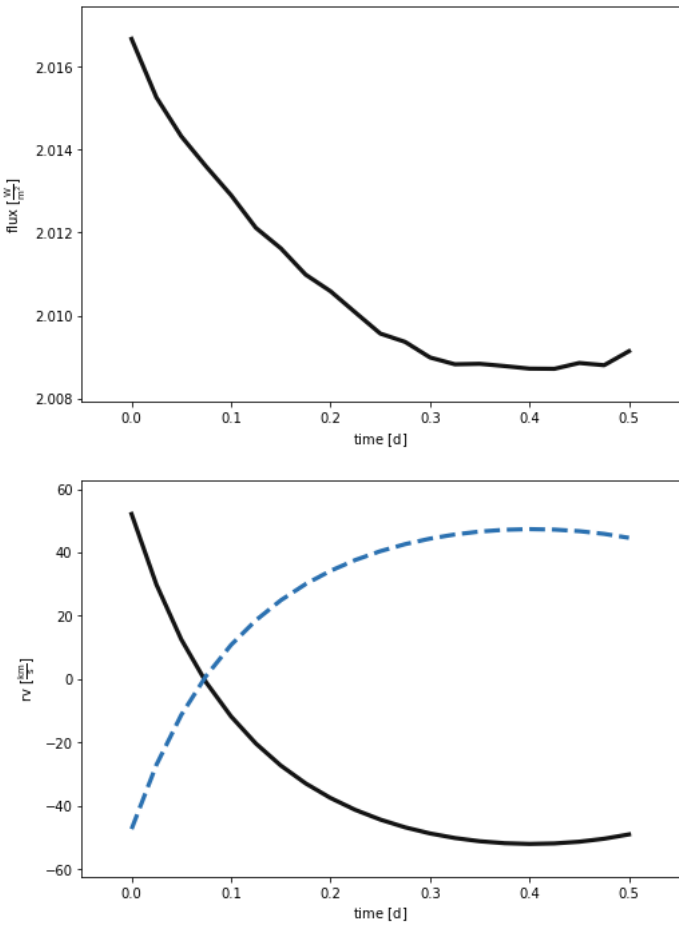

Figure 19: The LC and RVs for the following configuration

```
                rvs=[12,13,14,15,22,25,-21],
                component='primary',
                dataset='rv01',
                overwrite=True)
b.run_compute()
afig, mplfig=b.plot(show=True)
```

This code is configured for a *contact binary*, because PHOEBE threw errors for these observations for a normal detached binary system. These arrays are garbage data, which would never be the case for a real binary system. I generated these arrays randomly, and succeeded in confusing PHOEBE. The output of this code shows PHOEBE's results in analyzing garbage data. Please note that actual data would never be as bad as this.

Each dataset type has its own parameters, which can be modified to allow onboard processing of data if the user does not wish to preprocess the data. These parameters are complicated and extremely subtle. It is sufficient to understand the types of datasets PHOEBE takes.

6.4.1 Light Curves

Light curves are readily available across many spectra, and PHOEBE takes these light curves as input to produce a best-fit for the model. It should be noted that PHOEBE uses flux, that is $\frac{W}{m^2}$ as its unit for light curves. Many available light curves have different units, such as $\frac{electrons}{s}$ or simply apparent magnitude. Care should be taken to convert all of these other units into PHOEBE-compatible data.

6.4.2 Radial Velocities

Arguably the simplest dataset, RV datasets do not take much effort to add into PHOEBE. This is because RVs are usually reported in $\frac{km}{s}$, which happens to be the default unit PHOEBE uses. Simply add the dataset along with the times it was recorded at and PHOEBE will handle the rest.

6.4.3 Spectral Line Profiles

Light curves and radial velocity curves have the time of observation as their independent variable. However, spectral line profiles have the wavelength as an extra dimension. This means that each parameter in the LP dataset is tagged with its own time. We need to pass times when we want to add a dataset, though.

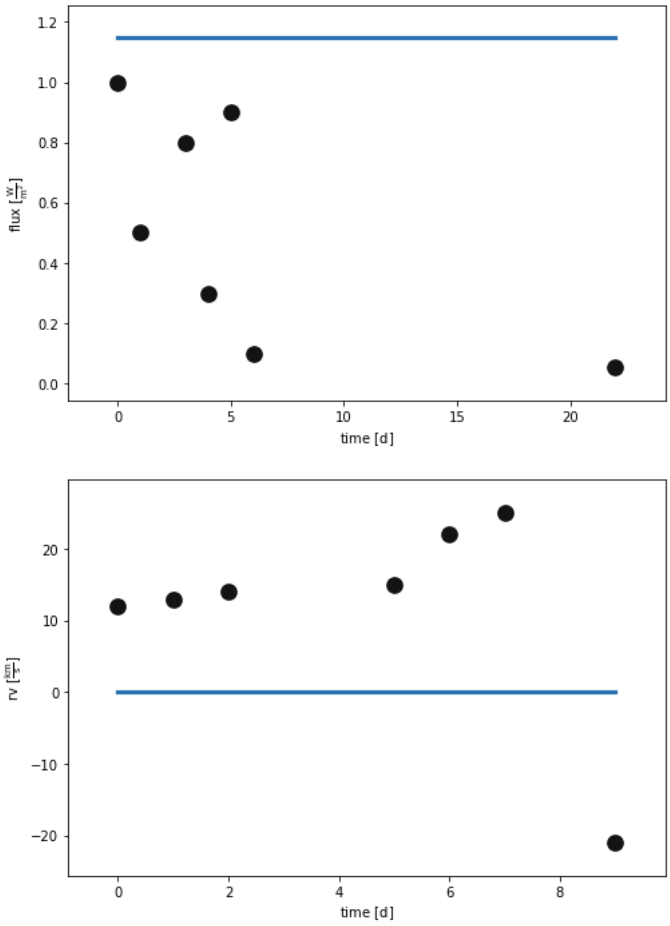

Figure 20: Garbage data simulations with PHOEBE

6.4.4 Orbital Data

In PHOEBE, you cannot add observational orbital data. This is because the orbital dataset contains the following parameters:

```
ParameterSet: 3 parameters
       compute_times@orb01@dataset: [] d
C      compute_phases@orb01@dataset: []
            phases_t0@orb01@dataset: t0_supconj
```

stlisting These values are computed automatically by PHOEBE. You can choose to compute relativistic effects using the `ltte` qualifier, but in most cases this is not required because binary star movement is slow enough for relativistic effects to not make a large change in the outcome. Note that this does not relate to the Doppler effect for spectroscopic observations, the formula of which includes the speed of light as a factor.

6.4.5 Meshes

PHOEBE can produce 3D models of stars and star systems and produce animations of how the stars actually move around each other, as well as what light and radial velocity curves we would observes. To create a 3D model, PHOEBE creates a mesh with coordinate points representing the star. It can then map attributes such as star spots and limb darkening on these points.

6.5 Plotting and Animation

Let's plot a system with PHOEBE. We can create models without observational data, but to make it look more interesting, let's create a synthetic model, obtain data from it, and use that data to create animations.

```
import phoebe
from phoebe import u
import numpy as np

b=phoebe.default_binary()
b["ecc"]=0.4
b["incl@binary@orbit@component"]=20
b["pitch@primary@star@component"]=0.1
b["pitch@secondary@component@star"]=0.2
b["q"]=1.134
```

```
b.add_dataset('lc', compute_phases=phoebe.linspace
    (0,1,101))
b.run_compute()


times = b.get_value('times', context='model')
fluxes = b.get_value('fluxes', context='model') + np.
    random.normal(size=times.shape) * 0.01
sigmas = np.ones_like(times) * 0.08

afig, mplfig=b.plot(show=True)
```

which produces

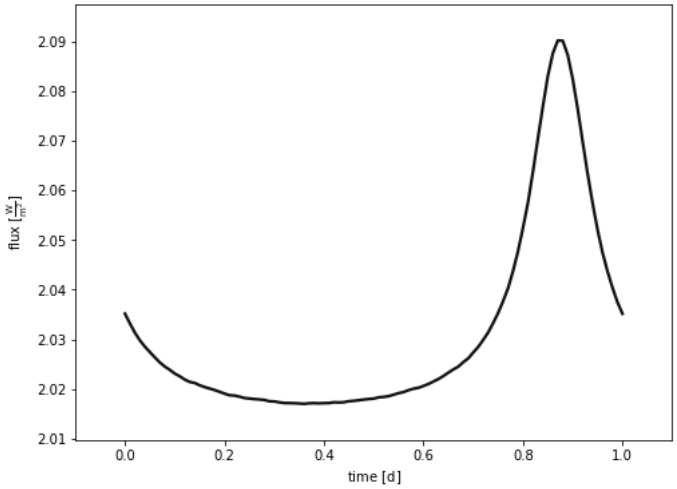

Figure 21: Random star system to generate data with

Sigmas are simply the error in measurement. We multiply with np.random() to make sure that the error is randomized as it would be in a true observation. Note that the error here is around 8

Let's fit this model to another star system, this time with slightly different parameters. This can be thought of as us guessing at the star's parameters. We have one set of observations, admittedly generated, but if we ignore that fact then we have a normal set of observations. We have now somehow guessed at the parameters of the star system and try to produce a synthetic model out of it. We have ocmputed this over a larger time period because we wanted to.

```
import phoebe
from phoebe import u
import numpy as np

b = phoebe.default_binary()
b["ecc"]=0.2
b["incl@binary@orbit@component"]=77
b["pitch@primary@star@component"]=0.6
b["pitch@secondary@component@star"]=0.023
b["q"]=2.135

b.add_dataset('orb', compute_times=np.linspace
    (0,4,1000), dataset='orb01', component=['primary',
    'secondary'])
b.add_dataset('lc', times=times, fluxes=fluxes, sigmas
    =sigmas, dataset='lc01')
b.run_compute()
afig, mplfig = b.plot(show=True)
```

This produces:

Figure 22: What we obtained from our 'guess' based on the generated data

Let's go one step further. For some reason, we have come to the conclusion that the observational light curves means that there are star spots on the secondary star. Star spots, in general, are relatively darker areas of the star's surface. They are analogous to sunspots, spots on the solar system's Sun. While sunspots are relatively small, starspots are much bigger. This is a slightly tricky statement. Stars *probably* have spots that are the same size as sunspots, but we can't detect them! So we say that the spots we *can* detect are starspots.

In PHOEBE, you can add starspots to a position on the star. A spot is defined by the colatitude (where 0 is defined as the North (spin) Pole) and longitude (where 0 is defined as pointing towards the other star for a binary, or to the observer for a single star) of its center, its angular radius, and the ratio of temperature of the spot to the local intrinsic value.

`print(b.filter(feature='spot01'))` gives us the following output:

```
ParameterSet: 5 parameters
              colat@spot01@feature: 0.0 deg
               long@spot01@feature: 0.0 deg
```

```
            radius@spot01@feature: 1.0 deg
          relteff@spot01@feature: 1.0
enabled@spot01@phoebe01@com ...: True
```

We will add that to our synthetic guess and see what happens. Let's make it slightly interesting and assume that this spot's attributes are `[12,34,1,0.9]` according to the PHOEBE list. The code will then be:

```
import phoebe
from phoebe import u
import numpy as np

b = phoebe.default_binary()
b.add_spot(component='secondary', feature='spotsec')
b["ecc"]=0.2
b["incl@binary@orbit@component"]=77
b["pitch@primary@star@component"]=0.6
b["pitch@secondary@component@star"]=0.023
b["q"]=2.135

b.add_dataset('orb', compute_times=np.linspace
    (0,4,1000), dataset='orb01', component=['primary',
    'secondary'])
b.add_dataset('lc', times=times, fluxes=fluxes, sigmas
    =sigmas, dataset='lc01')
b.add_dataset('mesh', times=[0,0.25,0.5,0.75,1.0],
    columns=['teffs'])

b.set_value(qualifier='relteff', feature='spotsec',
    value=0.9)

b.set_value(qualifier='radius', feature='spotsec',
    value=1)

b.set_value(qualifier='colat', feature='spotsec',
    value=12)

b.set_value(qualifier='long', feature='spotsec', value
    =34)
```

```
b.run_compute()
afig, mplfig = b.plot(show=True)
```

which leads to:

Why are there so many stars in the spots figure? This is because you can view spots at individual times, because spots move. We didn't do that here just to make the image look more interesting.

To animate the figure, we will simply use

```
afig, mplanim = b.plot(y={'orb': 'ws'}, fc='teffs', ec
    ='None', animate=True, save='animations_1.gif',
    save_kwargs={'writer': 'imagemagick'})
```

as the final line of code.

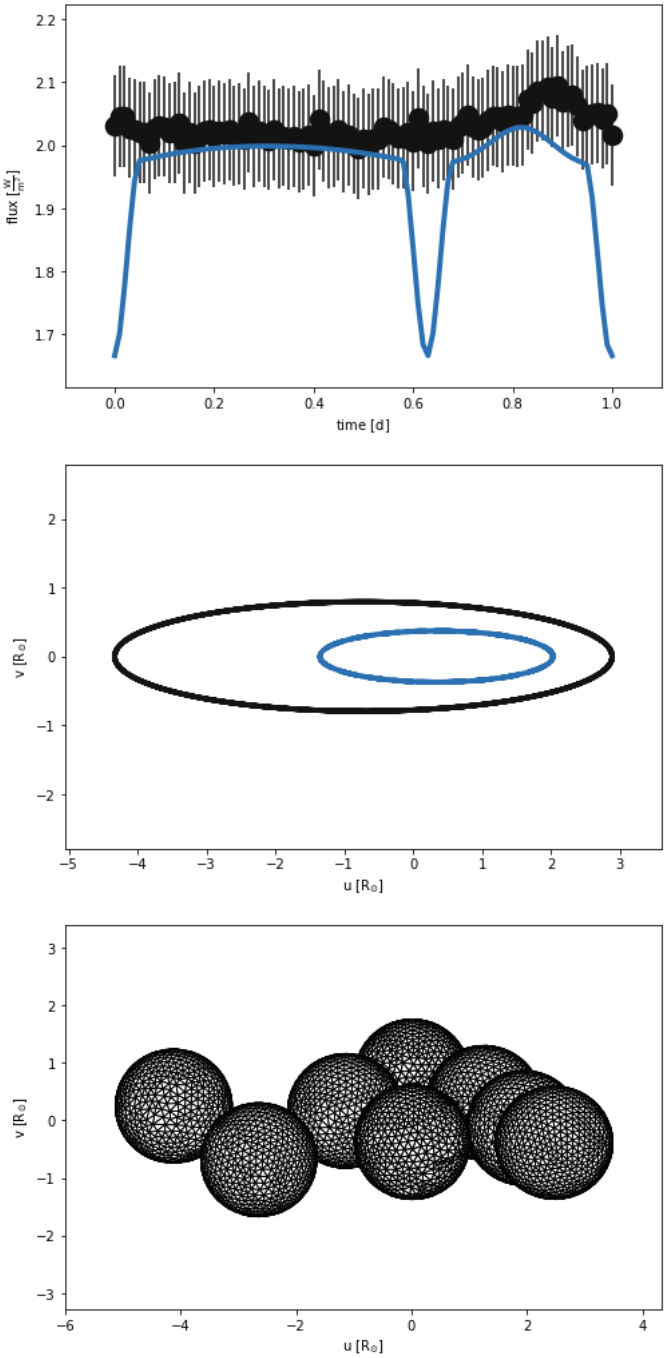

Figure 23: Spots!

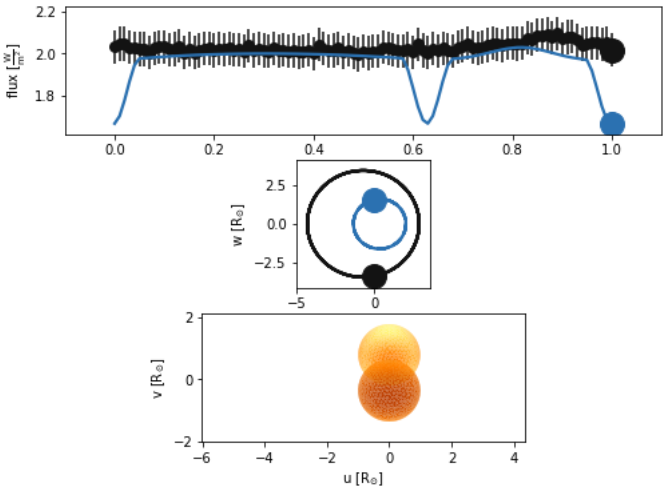

Figure 24: This image is one frame from the animation - this particular code sequence saves it to animations_1.gif

6.6 Solving the inverse problem and sampling

By now you must have realized just how vast and intricate the field of binary star analysis is. Solving a binary star system correctly warrants a publication and there is no general algorithm. Fitting curves to binary stars is problematic because we cannot know what type of curve to fit. We can try fitting sinusoids, polynomials, factorial variations, and so on, but it will still not give us a defined answer.

The difference between running a forward model and the general workflow for the inverse problem can be summarized in one image:

PHOEBE includes wrappers around several different inverse-problem ”algorithms” with a common interface. These available ”algorithms” are divided into three categories:

- Estimators: provides proposed values for a number of parameters from the datasets as input alone, not requiring full forward-models via `run_compute`.
- Optimizers: runs off-the-shelf optimizers to attempt to find the local (or global) solution.
- Samplers: samples the local parameter space to estimate uncertainties and correlations.

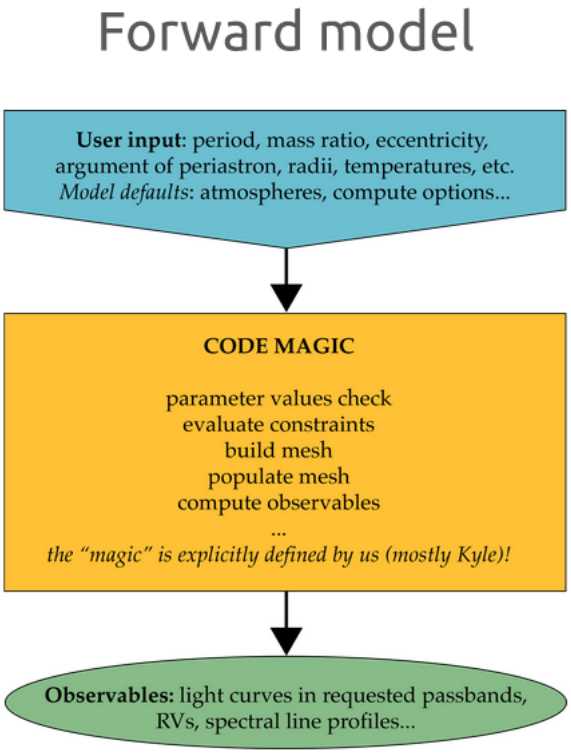

Figure 25: The different workflows - 1

In simple terms, estimators, which are trained on models form a neural network that attempts to guess at system values. Optimizers run several standard algorithms to refine these values. Samplers analyze the local parameter space in order to find correlations between different values and the uncertainty between them.

These algorithms do not provide a definitive answer in one go. Repetition is often needed, and sometimes the algorithms themselves refuse to compute values. This is because the variable spaces they are defined for may not always be applicable to real-life observations.

As a simple example, here is the tutorial from PHOEBE's official website which provides a very simplistic overview of how to solve the inverse problem. It features a synthetic dataset which is then put into an estimator. We already did this in the garbage data simulator above, but those featured initial guesses that

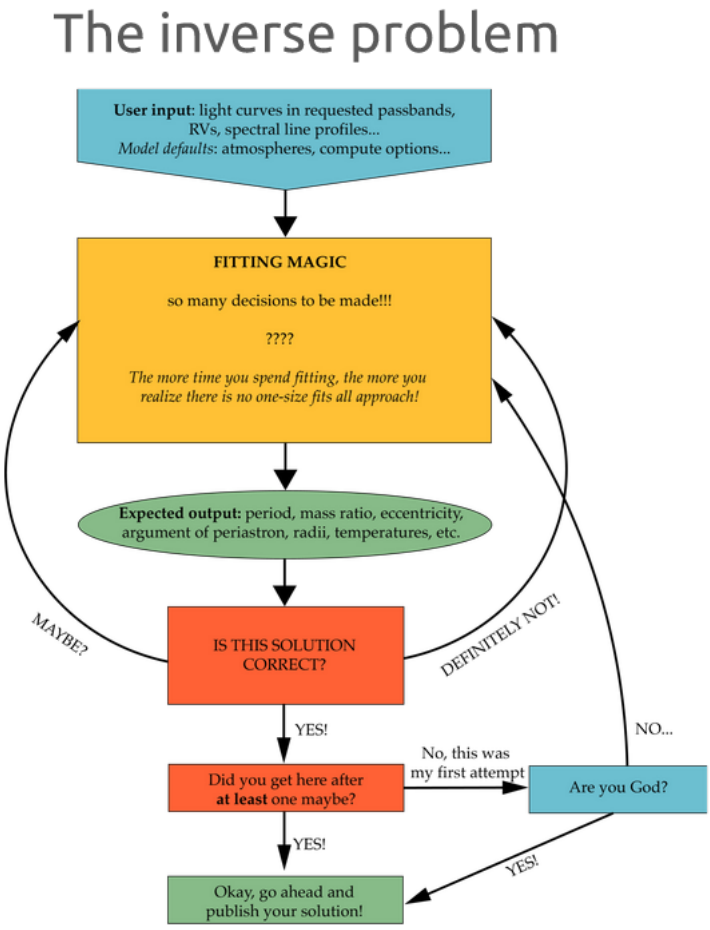

Figure 26: The different workflows - 2

were, indeed, garbage. Estimators simply give us an informed guess into what the initial values could be.

```
b = phoebe.default_binary()
b.add_dataset('lc', compute_phases=phoebe.linspace
    (0,1,101))
b.run_compute(irrad_method='none')

times = b.get_value('times', context='model')
fluxes = b.get_value('fluxes', context='model') + np.
    random.normal(size=times.shape) * 0.01
sigmas = np.ones_like(times) * 0.02

b = phoebe.default_binary()
b.add_dataset('lc', times=times, fluxes=fluxes, sigmas
    =np.full_like(fluxes, fill_value=0.1))
b.add_solver('estimator.lc_geometry', solver='
    my_lcgeom_solver')
b.run_solver(solver='my_lcgeom_solver', solution='
    my_lcgeom_solution')
_ = b.plot(solution='my_lcgeom_solution', show=True)
```

How does the solver know if its guesses are reasonably right? PHOEBE uses what is known as a `merit function`, a statistical tool described in great detail in the release paper that helps the solver decide if its synthetic model is 'close enough' to observational data.

This merit function uses logarithmic probability analysis in order to estimate if its synthetic model is correct or not.

6.6.1 Sampling

The word sampling has many definitions in science, including, but not limited to, converting analog signals to digital ones, estimating the characteristics of a population from a subset, and analyzing small parts of signals to manipulate them further.

PHOEBE's samplers, especially the emcee sampler, is used to estimate the errors in the synthetic model when comparing it to the observed data. Emcee sampling is more formally known as *Ensemble Sampling*, which is an evolved form of *Thompson Sampling* used in statistics. Thompson sampling, in its most basic

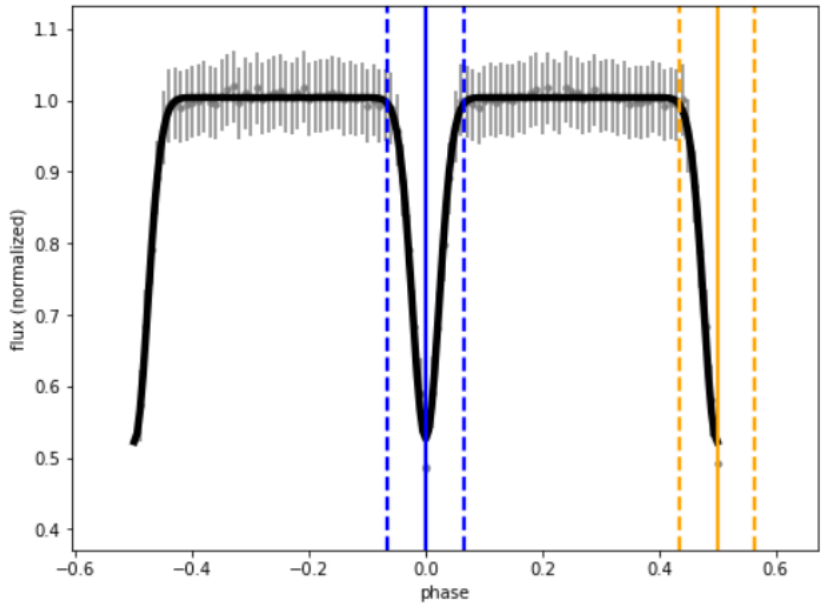

Figure 27: The outcome of a solver

form, is a short-cut technique (called a *heuristic*) that gives probabilities to estimate the next best action to take.

The above sentence can be broken down as follows:

1. Suppose you're performing a task and you're faced with a number of choices.
2. For simplicity's sake, let's assume you have three dogs who are hungry and begging you for food.
3. You only have a limited amount of food. You can't feed all three dogs fully.
4. What action do you take at this point? You can feed one dog fully and leave the other two practically starving. You can also feed two dogs so that they're mostly full and leave the third dog starving. You can divide the ood equally among three dogs. You can split the food in a way that each dog gets food depending on their size.
5. The underlying condition here is that you don't have enough time to go buy more food because the dogs want the food *now*. In this case, a human unconsciously applies a heuristic and gives food to the dogs. The type of

heuristic differs from person to person. Some people will feed all three dogs equally. Some people may not feed the dogs at all. Some people may feed only one dog and not the other two. In any case, this decision is arrived at *quickly*, instead of the person thinking it through.

6. Thompson sampling does the same thing for general problems. If you have a fixed amount of resources and some tasks competing for them, the Thompson sampler provides probabilities that tell you that *if* you do action X, it will *probably* give you a bigger chance of maximizing the output of all competing actions.

7. The way the Thomson sampler does this is by computing an integral over various parameters, their likelihood functions, posterior distributions, and prior observations, and returning a set of probabilities.

So what does ensemble sampling actually do? Notice the last point. A *posterior distribution* is required for Thompson sampling to give probabilities. That means that you need to give the Thompson sampler a probability distribution that tells it *how* the output will *probably* be *if* you do action X. This is fairly standard; you compute a few posterior distributions beforehand and feed it into your integral to get the probabilities of achieving them.

However, getting the posterior distributions is difficult. It is even more difficult if, like in PHOEBE, your observations are based on the output of a neural network. Neural networks in general are black boxes - it's impossible to estimate what goes on inside them in order for them to arrive at a solution. Instead of trying to wrestle with the neural network, it is easier if you statistically analyze its output. Ensemble sampling goes through a complex algorithm in order to produce Thompson distributions in order to make informed guesses about the future.

In PHOEBE, the emcee sampler analyzes the neural network in order to produce a solution that helps itself guess the next iteration better. How does it do that? Complicated statistics! Here's example code, the first part of which is ripped directly from PHOEBE's tutorial that *might* help you understand emcee sampling better.

First off, we'll generate some fake observations.

```
b = phoebe.default_binary()
b.set_value('ecc', 0.2)
b.set_value('per0', 25)
b.set_value('teff@primary', 7000)
```

```
b.set_value('teff@secondary', 6000)
b.set_value('sma@binary', 7)
b.set_value('incl@binary', 80)
b.set_value('q', 0.3)
b.set_value('t0_supconj', 0.1)
b.set_value('requiv@primary', 2.0)
b.set_value('vgamma', 80)

lctimes = phoebe.linspace(0, 10, 1005)
rvtimes = phoebe.linspace(0, 10, 105)
b.add_dataset('lc', compute_times=lctimes)
b.add_dataset('rv', compute_times=rvtimes)

b.add_compute('ellc', compute='fastcompute')
b.set_value_all('ld_mode', 'lookup')
b.run_compute(compute='fastcompute')

fluxes = b.get_value('fluxes@model') + np.random.
    normal(size=lctimes.shape) * 0.01
fsigmas = np.ones_like(lctimes) * 0.02

rvsA = b.get_value('rvs@primary@model') + np.random.
    normal(size=rvtimes.shape) * 10
rvsB = b.get_value('rvs@secondary@model') + np.random.
    normal(size=rvtimes.shape) * 10
rvsigmas = np.ones_like(rvtimes) * 20

b = phoebe.default_binary()

b.add_dataset('lc',
              compute_phases=phoebe.linspace(0,1,201),
              times=lctimes,
              fluxes=fluxes,
              sigmas=fsigmas,
              dataset='lc01')

b.add_dataset('rv',
              compute_phases=phoebe.linspace(0,1,201),
              times=rvtimes,
```

```
                rvs={'primary': rvsA, 'secondary': rvsB
                    },
                sigmas=rvsigmas,
                dataset='rv01')

b.add_compute('ellc', compute='fastcompute')
b.set_value_all('ld_mode', 'lookup')
```

Note what we did here. We have used the same variable `b` again, simply adding the generated datasets and errors to it.

From here on, there's a few things we can do. Obviously, we need to produce a model for the system guess at. We can try guessing at the system parameters by running PHOEBE's estimators and optimizers until we get a solution that looks good. Emcee will then probe the parameter space and estimate the uncertainties in each parameter. We can give the optimizers and estimators some initial guesses using known data. In the official tutorial, the model used as an initial guess is very close to the actual values, in order to make the computations simpler.

```
b.set_value('ecc', 0.2)
b.set_value('per0', 25)
b.set_value('teff@primary', 7000)
b.set_value('teff@secondary', 6000)
b.set_value('sma@binary', 7+0.01)
b.set_value('incl@binary', 80+0.1)
b.set_value('q', 0.3)
b.set_value('t0_supconj', 0.1-0.001)
b.set_value('requiv@primary', 2.0-0.05)
b.set_value('vgamma', 80)
```

Running `b.run_compute(compute='fastcompute', model='orig_model')` on this model results in a pleasing fit to the data.

PHOEBE gives us the option to see the *residuals* of our generated model. A *residual* for each data point is the difference between the actual observed data and the fit our model generates.

Now let's add the emcee solver to our data. We can do this using `b.add_solver('sampler.emcee', solver='emcee_solver')`.

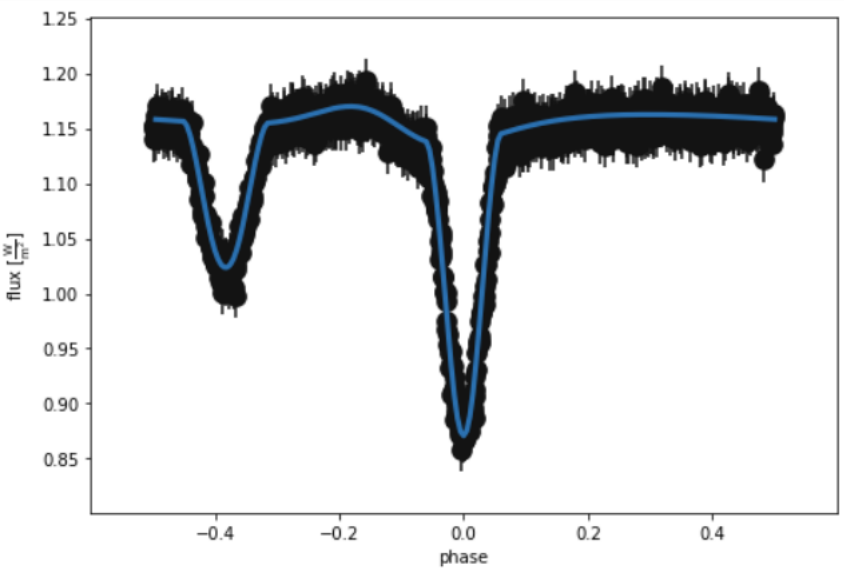

Figure 28: LC fit to generated data using the close model

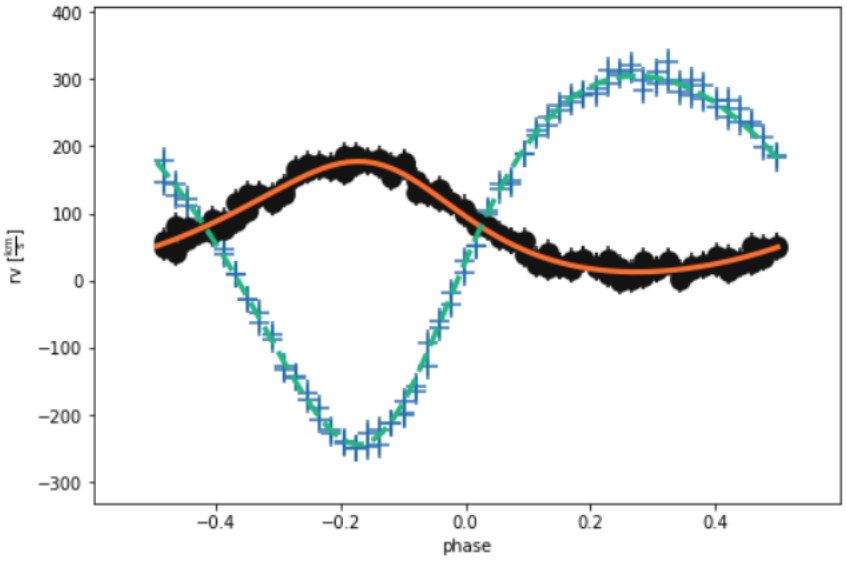

Figure 29: RV fit to generated data using the close model

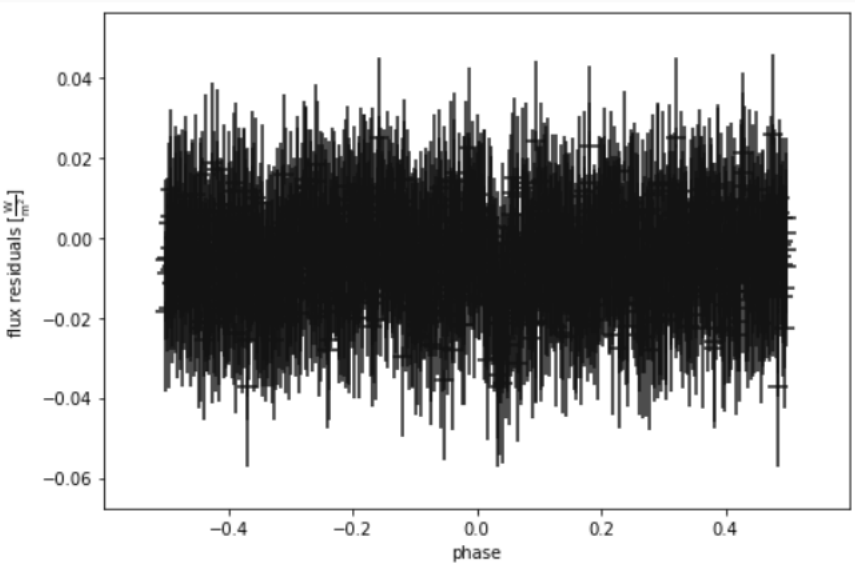

Figure 30: LC Residuals for our close model

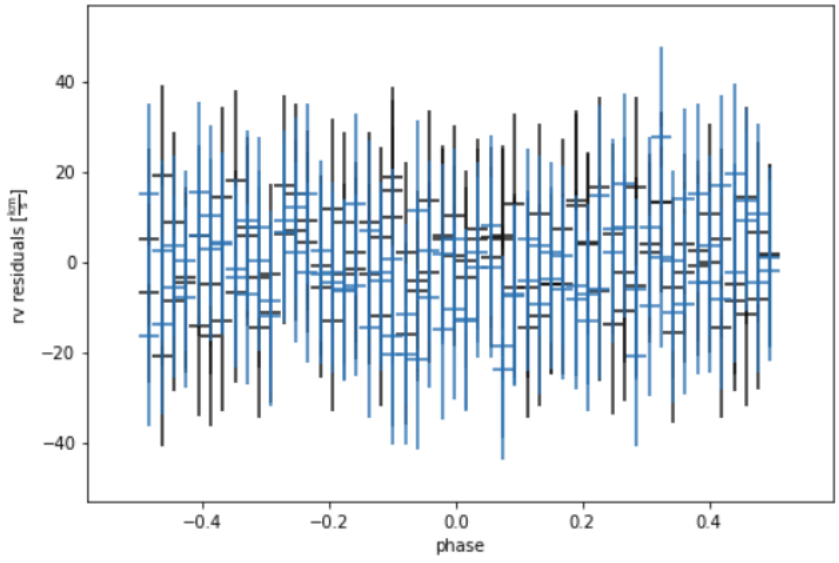

Figure 31: RV Residuals for our close model

We can call it using

```
b.set_value('compute', solver='emcee_solver', value='
    fastcompute').
```

At this point, we can run our solver using `b.run_solver()`. If we do this, however, we're missing out on what emcee really does. Since emcee is specifically used to analyze variables *statistically*, it would be better if we could specify the type of statistical distribution we want to analyze the errors for for our variables. PHOEBE provides a way to add distributions to individual variables. There are three statistical distributions available in PHOEBE: the *Gaussian*, the *Frequency*, and the *Uniform*. Out of these, the *Uniform* distribution is the easiest to understand.

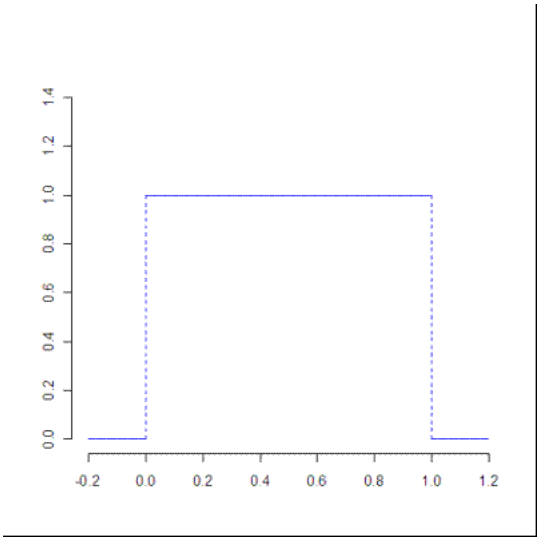

Figure 32: Uniform Distribution

The uniform distribution simply assigns equal probabilities for each possible value of a variable.

The *frequency* distribution, specifically the one implemented in PHOEBE, is called a histogram. A histogram is a distribution that creates 'buckets' of data; it splits the range of available variables into smaller parts. For example, if you have 1000 variables which could all take values from 1 to 100, you could try and find out *how many* variables were between 1-30, 30-32, 32-44, and so on. This is called a histogram of the data, and you can fit the histogram to different distributions.

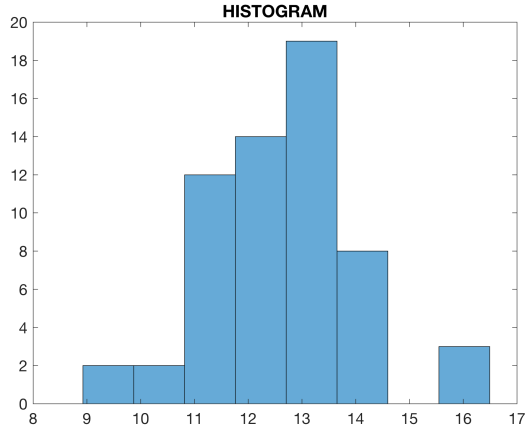

Figure 33: A histogram of data

The final distribution, the *Gaussian*, is one of the best tools of statistical analysis available. It is also called the *Normal Distribution* because of how ubiquitous it is in all areas of analysis. In simple terms, the Gaussian distribution relies on the fact that for many, many experiments, if you take more and more observations, many of them will tend towards a single value. This single value is called the *expected value* of the experiment. Fewer and fewer values in comparison tend to be further away from this expected value. Many of them lie close to the center. You can model the range of what values you expect the observable to take after repeated observations with a curve called a *bell curve*, which is shown below:

Here, μ is the expected value or *mean* of the observation, and σ is the *standard deviation*. The mathematical equation used for generating this bell curve is quite unintuitive. It is sufficient to know that around 68% of the values tend to fall within one standard deviation of the mean. (AN: Bell curves can be generated by using other types of distributions! If you see a bell-shaped curve, do not automatically assume it's Gaussian - but all Gaussian distributions will have bell curves in them).

In PHOEBE, you can assign Gaussian distributions to the variables you want to sample by providing a mean and a standard deviation. You can get around providing the mean by using PHOEBE's internal `gaussian_around` distribution as well, which is covered in the tutorials. You can also add multivariate Gaussian distributions which help with dealing with the *covariance* of two or more parameters. The *covariance* is the degree to which two related variables vary together.

Add a solver to PHOEBE using $\texttt{b.add}_s olver('sampler.emcee', solver = 'emcee\_solver').Theout p$

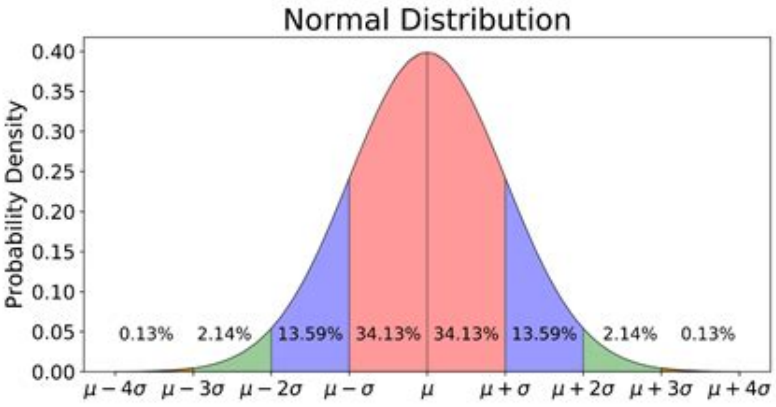

Figure 34: A Gaussian Distribution

```
ParameterSet: 11 parameters
     comments@emcee_solver@solver:
      compute@emcee_solver@solver: phoebe01
   continue_from@emcee_solver@...: None
    init_from@emcee_solver@solver: []
       priors@emcee_solver@solver: []
     nwalkers@emcee_solver@solver: 16
       niters@emcee_solver@solver: 100
   burnin_factor@emcee_solver@...: 2.0
   thin_factor@emcee_solver@so...: 0.5
   progress_every_niters@emcee...: 0
   expose_failed@emcee_solver@...: True
```

We can choose what options we can use to compute by modifying data in the above result table.

Let's add some distributions to some parameters, like so:

```
b.add_distribution({'sma@binary': phoebe.gaussian_around(0.1),
                    'incl@binary': phoebe.gaussian_around(5),
                    't0_supconj': phoebe.gaussian_around(0.001),
                    'requiv@primary': phoebe.gaussian_around(0.4),
                    'pblum@primary': phoebe.gaussian_around(0.2),
                    'sigmas_lnf@lc01': phoebe.uniform(-1e9, -1e4),
                   }, distribution='ball_around_guess')
```

We could've easily added a uniform distribution to more objects, but we're not sure how things vary. Instead we added a uniform distribution to the errors.

Now we run:

```
b.run_compute(compute='fastcompute', sample_from='ball_around_guess
    ', sample_num=20, model='init_from_model')
```

This verifies whether the distributions actually cover all of our observations.

Once done, we simply graph the results as follows:

```
_ = b.plot(dataset='lc01', x='phases',
           marker={'dataset': '.'},
           model='init_from_model', show=True)
_ = b.plot(dataset='lc01', x='phases', y='residuals',
           z={'dataset': 0, 'model': 1},
           model='init_from_model', show=True)
```

and get the following:

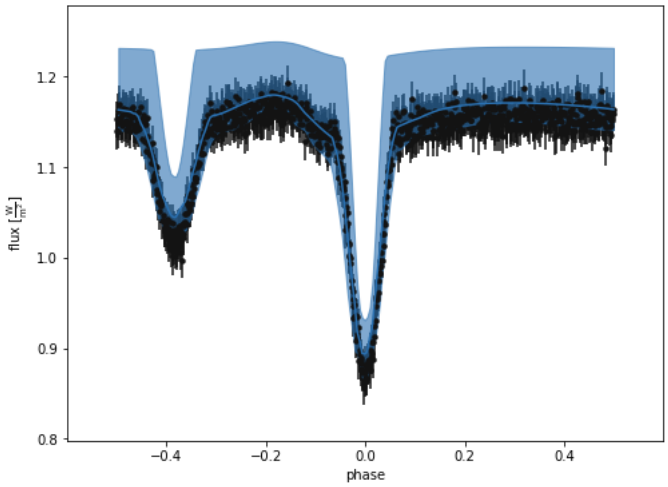

Figure 35: A light curve with possible variance fitted to our model

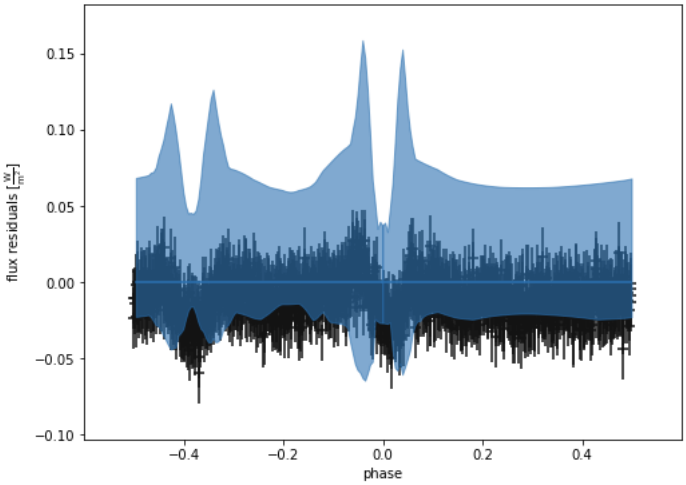

Figure 36: A radial velocity curve with possible variance fitted to our model

Note that we haven't actually sampled the data from our dataset - let's do that now. Let's sample only the light curves for quicker computation. Before running emcee solving, however, we need to pass the values of our distribution to the emcee solver with b.set_value('init_from', 'ball_around_guess'). We must also set the number of 'walkers' (an internal term for emcee that means the number of individual parallel path runs it'll use) and the number of iterations. We cannot estimate how long it will take emcee solvers to convert. We set the number of iterations to something small just to see how it works.

```
b.set_value('niters', solver='emcee_solver', value=250)
print(b.filter(qualifier='enabled', compute='fastcompute'))
b.disable_dataset('rv01', compute='fastcompute')
b.run_solver('emcee_solver', solution='emcee_sol')
```

After this runs, emcee adds new parameters to the bundle under the emcee solution tag. Running print(b.filter(solution='emcee_sol').twigs) results in the following:

```
['
    wrap_central_values@fastcompute@emcee_solver@emcee_sol@emcee@solution
    ', '
    fitted_uniqueids@fastcompute@emcee_solver@emcee_sol@emcee@solution
```

```
', '
fitted_twigs@fastcompute@emcee_solver@emcee_sol@emcee@solution',
 'fitted_units@fastcompute@emcee_solver@emcee_sol@emcee@solution
', '
adopt_parameters@fastcompute@emcee_solver@emcee_sol@emcee@solution
', '
adopt_distributions@fastcompute@emcee_solver@emcee_sol@emcee@solution
', '
distributions_convert@fastcompute@emcee_solver@emcee_sol@emcee@solution
', '
adopt_values@fastcompute@emcee_solver@emcee_sol@emcee@solution',
 'niters@fastcompute@emcee_solver@emcee_sol@emcee@solution', '
nwalkers@fastcompute@emcee_solver@emcee_sol@emcee@solution', '
samples@fastcompute@emcee_solver@emcee_sol@emcee@solution', '
failed_samples@fastcompute@emcee_solver@emcee_sol@emcee@solution
', '
lnprobabilities@fastcompute@emcee_solver@emcee_sol@emcee@solution
', '
acceptance_fractions@fastcompute@emcee_solver@emcee_sol@emcee@solution
', '
autocorr_times@fastcompute@emcee_solver@emcee_sol@emcee@solution
', 'burnin@fastcompute@emcee_solver@emcee_sol@emcee@solution', '
thin@fastcompute@emcee_solver@emcee_sol@emcee@solution', '
lnprob_cutoff@fastcompute@emcee_solver@emcee_sol@emcee@solution
', 'progress@fastcompute@emcee_solver@emcee_sol@emcee@solution',
 'comments@fastcompute@emcee_solver@emcee_sol@emcee@solution']
```

You can *adopt* parameters from emcee's solving once you run it. By adopting a value, future runs will use them as initial values for their sampling. By default emcee adopts everything.

6.6.2 Plotting emcee solutions

Emcee's solutions can be plotted in a wide variety of ways. Emcee's three parameters - burnin, thin, and lnprob_cutoff can all be modified in order to check whether emcee has worked properly. Unfortunately, this takes quite a bit of trial and error to pull off successfully.

```
_ = b.plot(solution='emcee_sol', style='lnprobability', show=True)
```

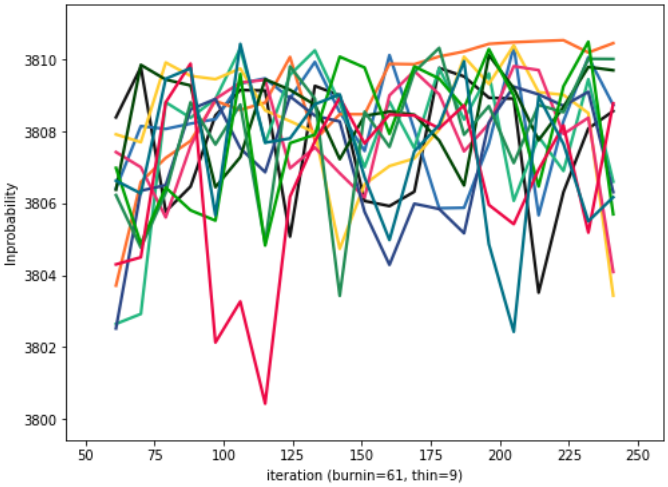

Figure 37: Emcee's guesses

results in:

Setting burnin, thin, and lnprob_cutoff to some values we want, we can generate a *corner* graph and produce the following impressive diagram.

```
b.set_value('burnin', 100)
b.set_value('thin', 1)
b.set_value('lnprob_cutoff', 3600)
_ = b.plot(solution='emcee_sol', style='corner', show=True)
```

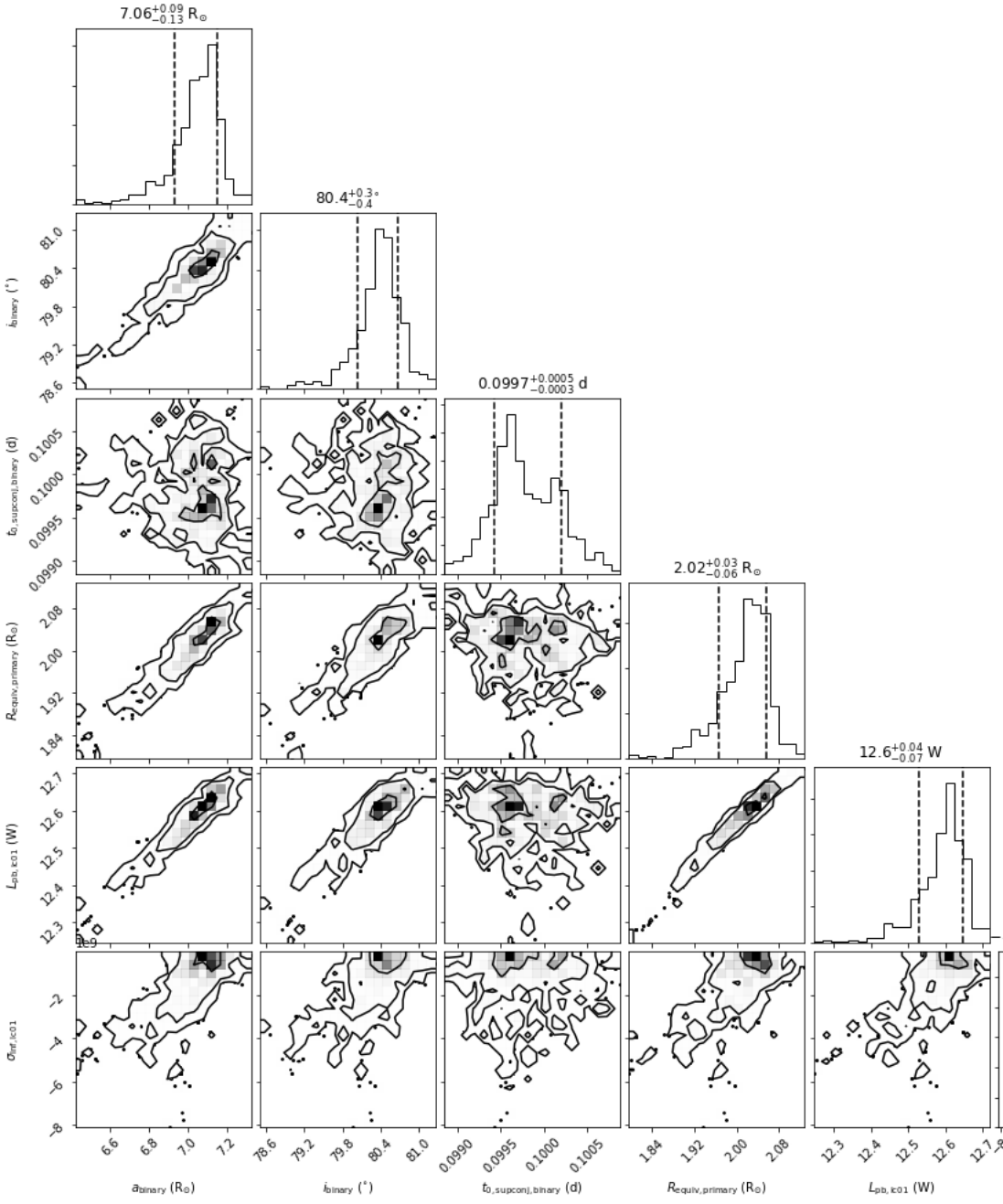

Figure 38: Emcee's impressive graph

From here, it is a simple matter of adopting the solutions or passing them forward.

Sampling is a difficult, time-consuming task that requires detailed knowledge of statistics. Emcee is merely one form of sampling - many more exist, and may be implemented into PHOEBE in the future.

7 QX Cas

We now come to the final part of the report: analyzing an actual star. The star chosen for this specific purpose is a star called *QX Cas*. QX Cas is a binary star system in the constellation of Cassiopeia. The first observations of QX Cas were taken in the 1950s, and it was mapped several times over the future decades, though not thoroughly at all. The light curves of QX Cas, when compared to over time, show a very peculiar trait. The ellipses of the light curves do not dip as time goes by. In fact, observations taken in 2009 do not show a dip at all. This means that QX Cas has stopped eclipsing!

Binary systems that have stopped eclipsing are very rare. This is because most binaries haven't stopped eclipsing in the short period of time we've been observing them. Only a couple of systems - like HS Hydrae and QX Cas - have been studied over the years. It is believed that many such systems exist - after all, why wouldn't they? - but analyzing every single data source for every single binary star system is simply too tedious and time-consuming. The amount of effort required to analyze QX Cas itself is difficult; and as you will see, the analysis is simply not possible in PHOEBE as it is today.

A quick note on *magnitudes* - briefly, the *magnitude* of a star system is a way to measure how bright it is. One might think that stars with a higher temperate would shine brighter, and this is true. However, the distance of a star from the Earth plays a much more major role in determining how bright it is. A large, 'dim' red giant which is very close to the Solar System will be much brighter to us than a small, fiercely burning blue star that's several hundred parsecs away. There are two types of magnitudes - the *apparent* magnitude and the *absolute* magnitude. The *apparent* magnitude is the brightness of the star as seen from Earth - distance and all - compared to a standard reference star. There are various reference stars used, and every stellar survey's reference star may be different, there's no way to know unless it's

specifically mentioned or you ask them, and the magnitudes are often published according to those reference stars, without publishing the reference star itself. For most observations, the star Vega, located in the constellation of Lyra, has been used as the reference. There is a way to determine the *absolute* magnitude of a star from its apparent magnitude. The *absolute* magnitude attempts to fix the problem of stellar distance. It places the star at a distance of 10 parsecs from the Earth, removes any interstellar gaseous medium and light-bending heavy objects, and measures the brightness. Of course, there is no way to do this physically, but this is where mathematics comes to help us.

The mathematical value of the apparent magnitude is:

$$m_x = -2.5log_{10}(\frac{F_x}{F_{x_0}}) \tag{1.3}$$

where F_x is the observed flux of the star, and F_{x_0} is the observed flux of the reference star.

The absolute magnitude can be determined from the apparent magnitude m by:

$$M = m + 5(log_{10}p + 1) \tag{1.4}$$

where p is the stellar parallax. The stellar parallax is simply the difference in the angular position of an object measured from two different lines of sight at the same time. A *parsec* is a unit of astronomical distance equal to approximately 3.26 light years. It is defined as the distance at which 1 AU subtends an angle of one arcsecond. This means that the distance in parsecs to any heavenly body is the reciprocal of its parallax.

Note that since the unit is logarithmic, the lower your magnitude is, the brighter you are.

The QX Cas system is of magnitude 10, and is a member of the young open cluster NGC 7790. It is located at 359.67981822668° RA and +61.16098428654° Dec. This data was obtained from the SIMBAD astronomical database, which is a database that correlates and synchronizes the observations of many different surveys. It is an invaluable tool.

7.1 Some thoughts on SIMBAD

SIMBAD is quite overwhelming to use at first. The kicker is, when you learn to use it, you realize that all the other databases are worse! The

position and distance to QX Cas were obtained from the second release of GAIA data, which is accesible through SIMBAD. Unfortunately, there were no light curves available for this particular star through SIMBAD, so I had to look somewhere else. This is often the case with SIMBAD - it syncs quite a few databases up, but not all of them. Still, it's very good as a starting point.

In 1954, GE Erleksova discovered QX Cas to be an eclipsing binary. A. Sandage provided correct photometry of QX Cas to confirm its eclipsing nature and provided accurate measurements of UBV magnitudes. Early light curves of the star show eclipses with a depth of approximately 0.3 and 0.28, respectively. Observations of this star are scarce as it has not been widely studied in literature. However, data from GE Erleksova's compilation provide us with 1135 observations of QX Cas. Modern light curves of this star are also scarce, because this star has stopped eclipsing over the past half decade. Data from AAVSO and TESS are used to obtain light curves of QX Cas for the 21st century and show that it has stopped eclipsing.

7.2 Light Curves

The original light curves from the first recorded observations are shown below.

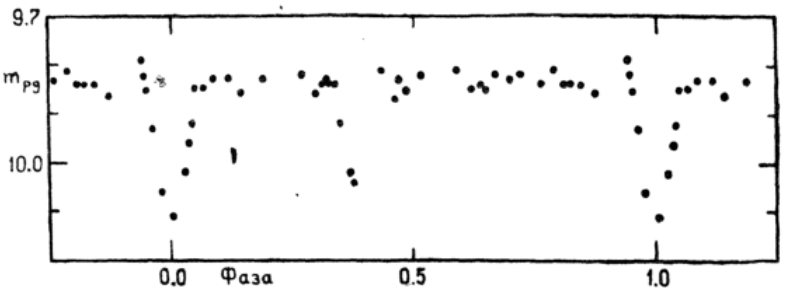

Figure 39: Light Curve as mapped by GS Tsarevsky

1135 observations of QX Cas were compiled by GE Erleksova in 1953. No guesses were ever made at the stellar parameters until 2009, where Michael Bonaro (one of the people who used to develop PHOEBE!) published an analysis of this star's parameters. Research in 2012 published in JAAVSO (the Journal of the American Association of Variable Star

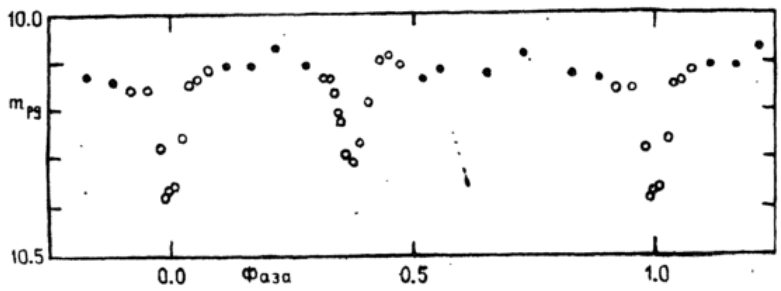

Figure 40: Light Curve as mapped by GE Erleksova

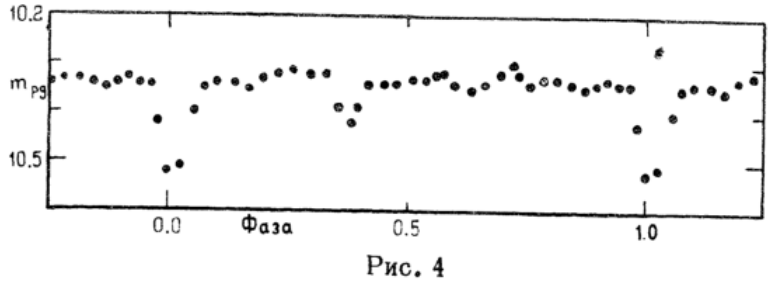

Figure 41: Light Curve as mapped by GE Erleksova with photographic plates

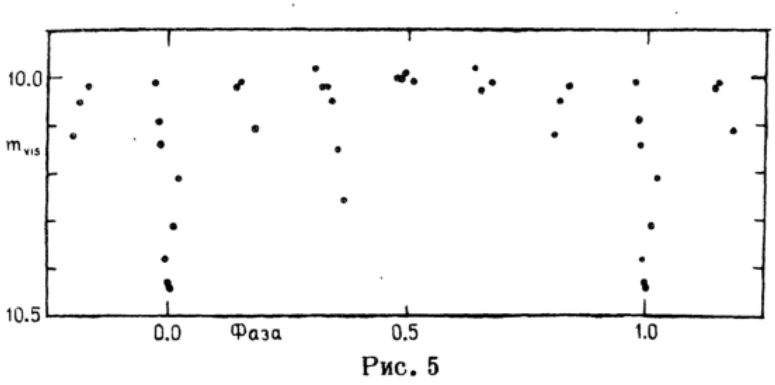

Figure 42: Light Curve as mapped by GA Lange and SV Nekrasova

Observers) provided light curves that showed the decline of QX Cas as a system.

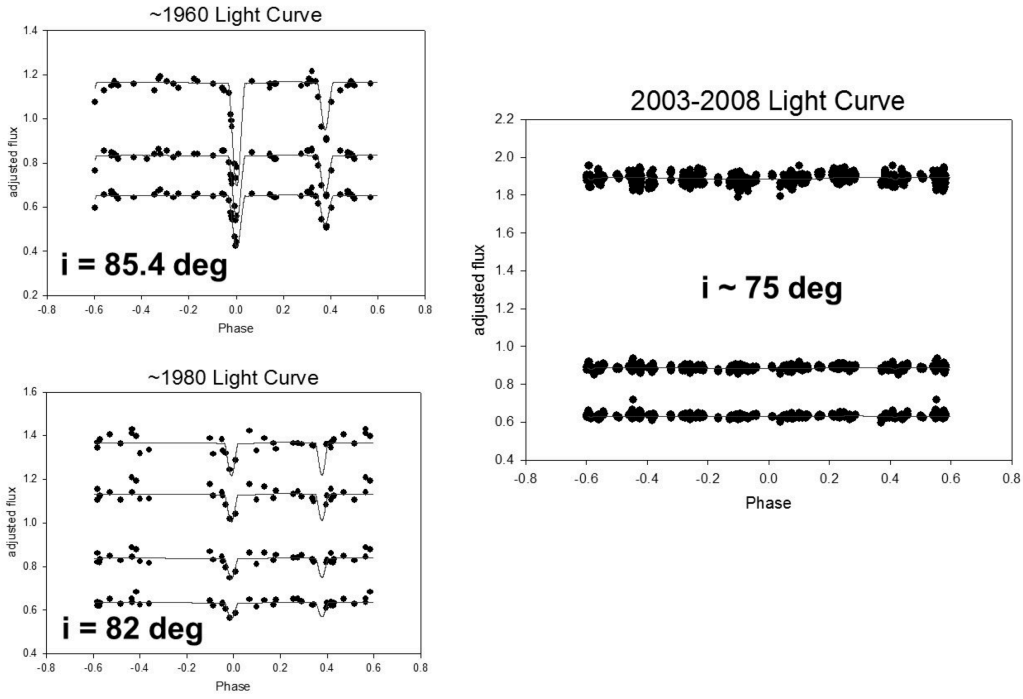

Figure 43: The light curve of QX Cas over many years

I tried to find out GE Erleksova's original compilation of observations. Through a bit of luck, I found a scanned copy of the original paper (written entirely in Russian) on Harvard's ADS (Astronomical Database System). This system contains many papers as well as datasets and SIMBAD references for the stellar objects analyzed in them.

I translated the paper and wrote each and every observation by hand and converted it into a computer-readable form. OCR software was of no use because of the way the data were published. The data itself were published very poorly, often repeating points and not having some observations at all.

However, these data were invaluable because they were the only observations taken when QX Cas was, presumably, still eclipsing without showing signs of decline. The data were available in HJD forms along

with magnitudes which were left unmentioned. Observations were taken in the visible band, the V band, and the B band.

7.3 Bands of observation?

The instruments used for recording the magnitude of a star are photographic plates. Each plate has a filter applied to it. The form of this filter may differ. For GE Erleksova's compilation, the filters were sheets of material allowing only a certain wavelength in. The way these sheets were generated was by using a chemical wash. Unfortunately, no data on this chemical wash exists, and I was forced to assume it was a standard procedure. This was a big problem for me, because the effect of the chemicals would definitely change as the solution evaporated over time.

7.4 Translation

The translation of the paper, along with the data tables, is provided after this paper. The process for generating a light curve from the actual data tables was fairly straightforward. Since the HJD and the period of revolution (6.004 days) was provided, you could phase-shift the data and compile all of the observations into one. It was at this point that I found out that the combination of the time period and the data table was called an *ephemeris*.

I faced an important issue: *how* do you know if the data is in phase? You need to *phase-shift* the data. This was not entirely possible by using Erleksova's observations alone. Later observations of QX Cas, which led to different ephemeris' being published, allow one to phase-shift the data. I proceeded with creating a *phase-folded curve* of QX Cas, which simply 'folds' the various observations into one big light curve that depends on the phase.

Here is what the messy phase-shifted curve of QX Cas looks like:

Of course, this curve looks messy because I don't know how to process the raw data in a way that'll make the curve look good - but don't worry, data from recent observations will look very good!

7.5 Getting data into PHOEBE

PHOEBE accepts only fluxes as the y-axis for its plots. You need to convert magnitudes into fluxes. This is also a fundamentally impossible

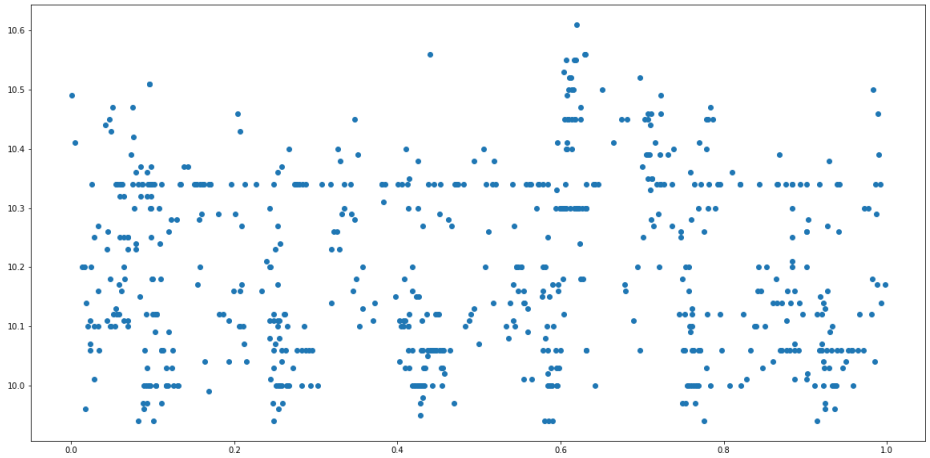

Figure 44: QX Cas' phase-shifted scatter plot - very messy!

task, since the aperture of the observing instrument, accounting for atmosphere, and a myriad of other things make deriving the exact fluxes next-to-impossible. I eventually ended up converting the magnitudes to fluxes using the formula for the magnitude itself, assuming that the reference star had a flux of 1. This didn't seem to lead to any problems, so I went with it.

Further photometric data was available in a paper by Moffett and Barnes, published in 1993. This provided some evidence that the binary star was indeed spiraling out of orbit, however, it was too early to tell accurately.

Spectroscopic observations for QX Cas came from the observations of D. P. Kjurkchieva, D. V. Marchev, and S. Zola, in 2003. These provided data points for calculating the radial velocities of the system - however, there were no photometric observations around this time, as the system itself had heavily receded. It is worth noting that S. Zola worked with Michael Bonaro on solving QX Cas in a more complete way at the end of the decade. A 1980 paper by Popper provides a way to estimate the temperature of stars based on their B-V photometry and eclipse depth. However, I could not find the paper that did this, so I ended up using Ballesteros' formula, which assumes the stars a black bodies and calculate their temperature based on their B-V photometry index. This is fundamentally wrong - limb darkening, star spots, and gravity effects all play a large part in determining the temperatures

of the star system, as we have seen in PHOEBE, but it is the best we have today.

$$T = 4600(\frac{1}{0.92B - V + 1.7} + \frac{1}{0.92B - V + 0.62})K \qquad \text{(Ballesteros' formula)}$$

Which values of B-V do I take? I ended up averaging the temperatures that I obtained from each of the observations. This led me to temperatures of 19,843K for the bigger star and 28,122K for the smaller star, which is in rough agreement with the temperatures provided by Zola et al. There is an interesting observation here - the bigger star has the smaller temperature!

Now that I have the radial velocities, the light curves, the distance, position, and the temperatures for QX Cas, I should be able to attempt the inverse problem in PHOEBE for QX Cas. However, I was faced with an unsurmountable problem: *PHOEBE does not attempt to solve the inverse problem for stars that are this hot. Nor does it accept data points which are not clustered more than 0.25 phase units apart for running its ebai estimator!*. This stymied me, so I eventually ended up preparing a forward model for the radial velocities. I did not attempt to create a light curve for the star because PHOEBE simply refused to create a synthetic model once the temperatures were given as parameters.

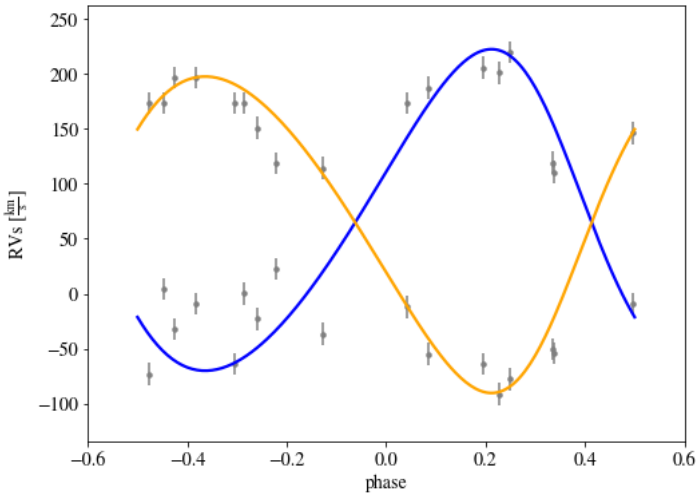

Figure 45: QX Cas' radial velocities using the rv_geometry solver!

Why is QX Cas so interesting, particularly among disappearing eclipsing binaries? Well, *there are periodic signals emanating from QX Cas according to TESS data, years after it stopped eclipsing!*. These signals have a period of 1.1 days, and the cause of this is unknown. I set out to investigate the cause using the incomplete work of J.R.A. Davenport as a base.

7.6 TESS

The TESS project is a project that maps light curves every 30 minutes for sectors of the observable universe. It found interesting things about QX Cas. TESS is accessed by the Python package LightKurve, a very helpful package that provides easy access to various light curves. TESS' units are $\frac{electrons}{s}$ on a small receiver plate instead of $\frac{W}{m^2}$, but since I'm no longer using PHOEBE, it doesn't really matter.

TESS data showed that yes, *something* was going on with QX Cas! The light curves available from TESS data - flux versus time, show some sort of periodicity.

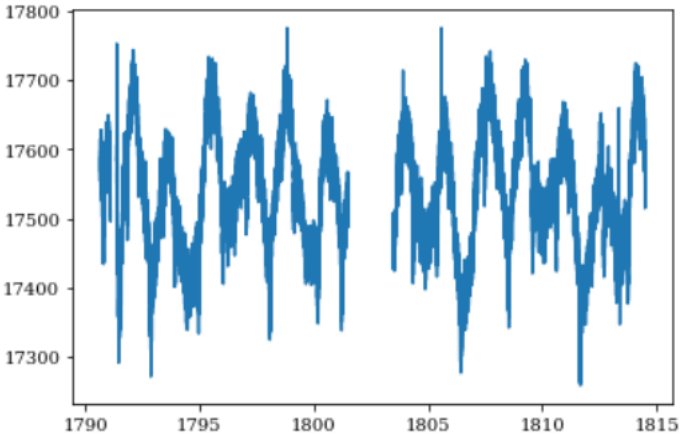

Figure 46: QX Cas' radial velocities using the rv_geometry solver

The format of the data file was .FITS. FITS stands for Flexible Image Transport System, and is the standard data format for sharing astronomical data all across the world. FITS data can be read in a variety of ways, but the Pandas package is sufficient enough for our purposes.

Well, this curve revealed that *something* was definitely up with QX Cas - there really shouldn't be any power source in that region unless it's a star system *behind* QX Cas that's releasing it.

How do we know that the source is periodic? The data are in fact very difficult to read. We use what is called a *Lomb-Scargle estimator* in order to probe the data. This is done by using Pythons *exoplanet* package. This package was originally developed for analyzing transiting exoplanets, but works just as well for binary stars. The Lomb-Scargle estimator works by analyzing an uneven set of periodograms (estimates of how dense a signal is for a certain spectrum) using a Fourier transform of the periodogram fitted to a function invented by Scargle in 1982. Scargle's function is as follows:

$P(f) = \frac{A^2}{2}(\sum_{n=1}^{n} g_n cos(2[t_n - \tau]))^2 + \frac{B^2}{2}(\sum_{n=1}^{n} g_n cos(2[t_n - \tau]))^2$ where A, B, and τ are arbitrary functions of the frequency and the time and mathematically proved that the periodogram can be reduced to classical forms for equally spaced observations if a certain set of conditions were fulfilled. He also used this particular format in order to make the periodogram relatively easy to compute by computers, and also made it insensitive to time scale shifts. You can then apply this function to your data and obtain the time period of the obseervations and the noise at different frequencies.

In practice, the Lomb-Scargle estimator in Python is a 'black box' function where you simply input the data used and return the frequency and the power, if you use autopower() as a calling function.

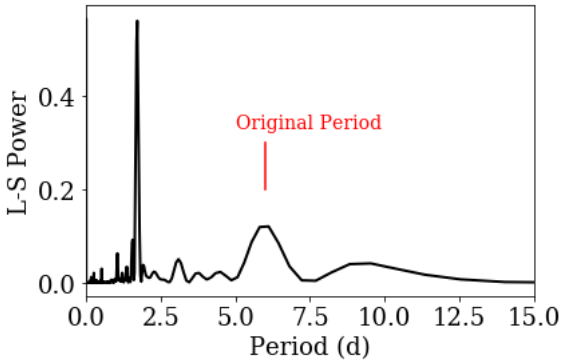

Figure 47: Time periods in the original data

The period originally found was 0.521 days - but if it's a binary star system, you'd have to double it since both stars cross each other.

We are proceeding on the assumption that it's a binary star system after all even though it stopped eclipsing.

To generate a more understandable flux graph, you have to phase-fold the data. A clever way to do this is by converting the time to phase using Pandas' rolling function in order to take 'windows' of data at a time, and convert them to phase. It's a neat trick, but hard to understand at first; I eventually ended up going with the longer but easier to understand method while programming.

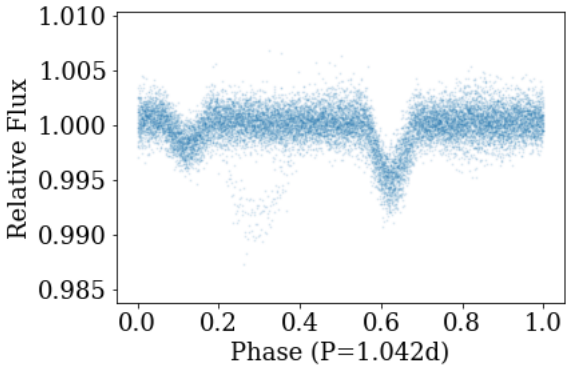

Figure 48: Phase-shifted modern data

Here is the data widened out:

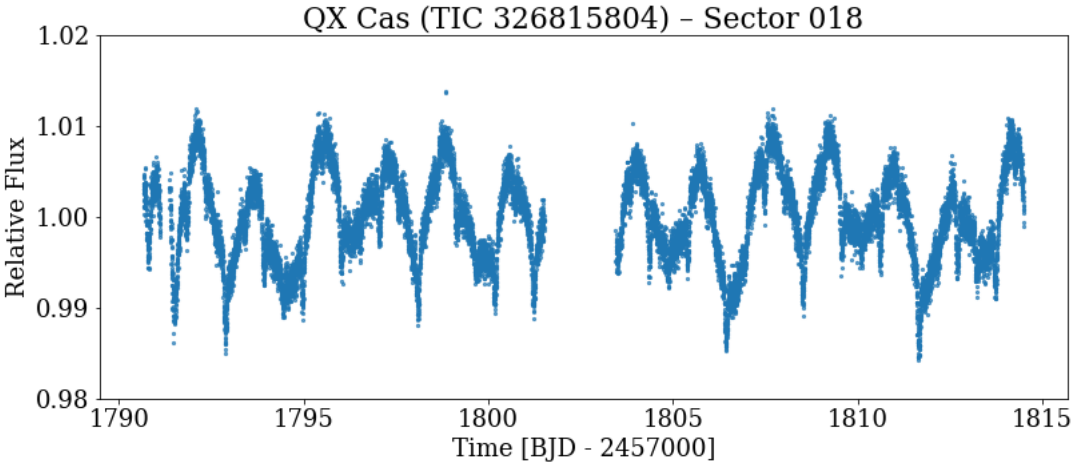

Figure 49: Time data widened out

`The sudden regular dip in magnitude at the beginning is interesting - we really shouldn't have that. Maybe some object in space wandered by here and caused this to happen? In that case, it would be` *`transiting`* `the binary star system and that transit would only happen once. If this transit only happened once, what would be its period converted to phase? There's no real way to know, so let's assume it's 0.3.`

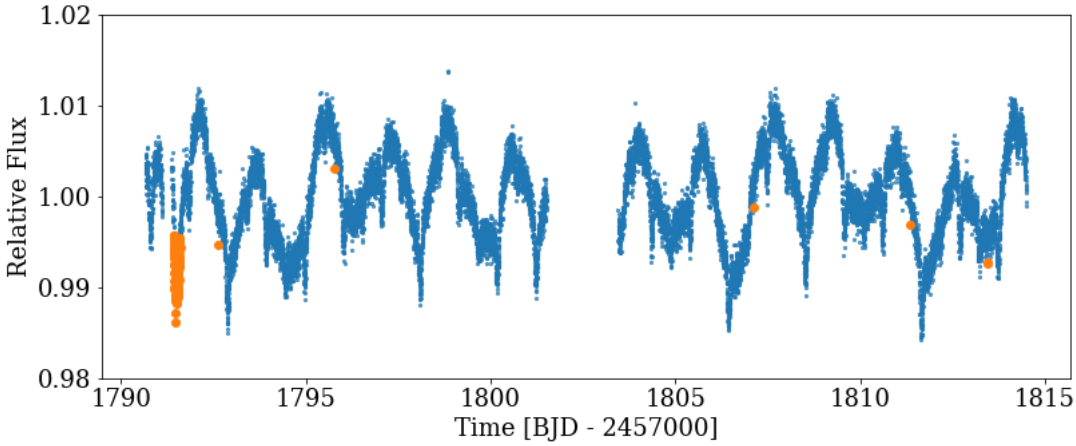

Figure 50: Weird drop

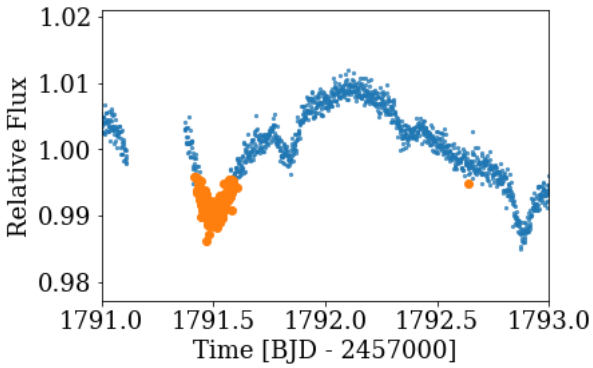

Figure 51: Gap in the data - so probably just TESS' internal noise or data artifacts left over. *Probably* not worth investigating further

`In the second figure, I ended up dividing by the median of the data in order to see what exactly went on - and it really didn't look like`

a transiting planet unless it was very small. HOWEVER - I am very inexperienced, so it could actually be a transiting planet for all I know.

Data from TESS sector 24, which covers that part of the sky, shows that the interesting signals are still present. Of course, the initial plot shows a large gap because of how far apart the observations were taken.

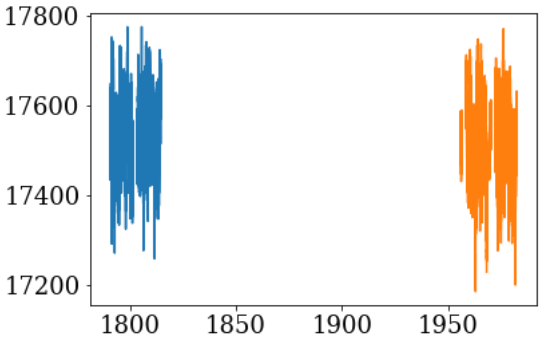

Figure 52: The two observations taken

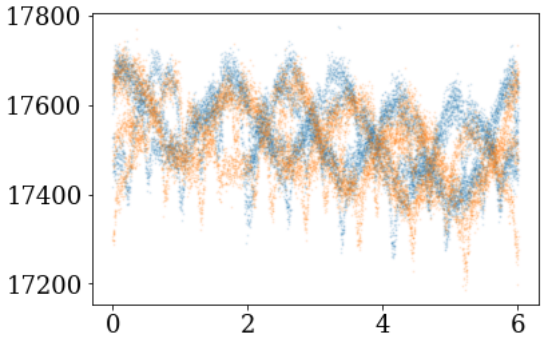

Figure 53: Combining both phase-folded curves

The Lomb-Scargle estimator revealed that there was a new power source in the system. It was also periodic, and had a period of *1.7d*. Where could this power come from? Starspots? Maybe, we'd have to check if that was truly the case in PHOEBE. We'd need to position the star spots and run simulations for a whole bunch of synthetic models. But

with no information on where the starspots could feasibly be, I could not proceed further.

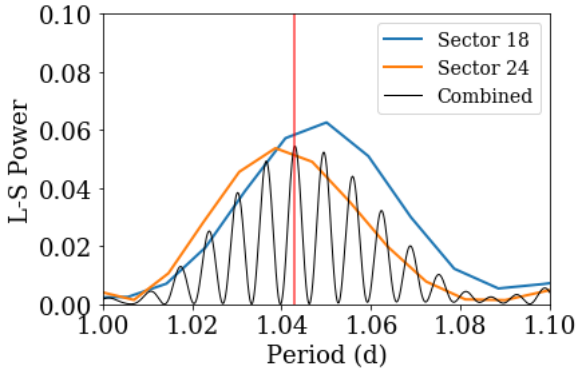

Figure 54: The time periods obtained from Sector 24 shows that there's *something* going on here

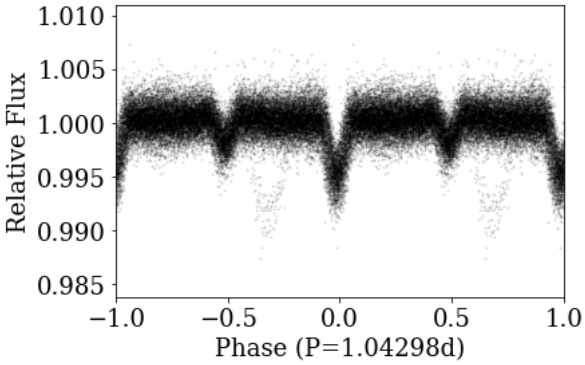

Figure 55: Phase-folded curve from TESS

We need more data in order to characterize this eclipse. By itself, TESS data can't really reveal anything. This is because TESS' observation aperture is large, and a background star may be causing the problem.

Assigning a Gaussian distribution to the variation in power, let's see if we can fit the phase-folded data to a single line.

There may have been some eclipses observable in late 2020. We don't have any data for that, so we have to guess at the atmospheric variation

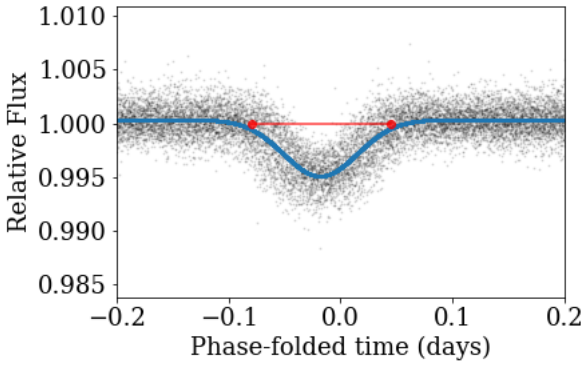

Figure 56: Well you *can*, and it looks good too!

used. And even then, it turned out that the answer was a 'maybe', so I didn't follow it up.

Assuming that the new signal comes from starspots, can we make an empirical model of all three signals?

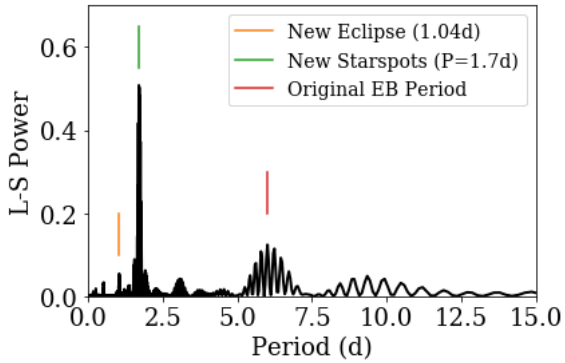

Figure 57: All three signals

I was able to, and it looked okay....some of the data points do lie 2 standard deviations away, which is probably not good for starspots. The residuals looked okay, but again, MORE DATA is required!

7.7 Future Scope

More data is required to analyze QX Cas. In theory, you should be able to determine what exactly caused these particular periodic pulses. TESS

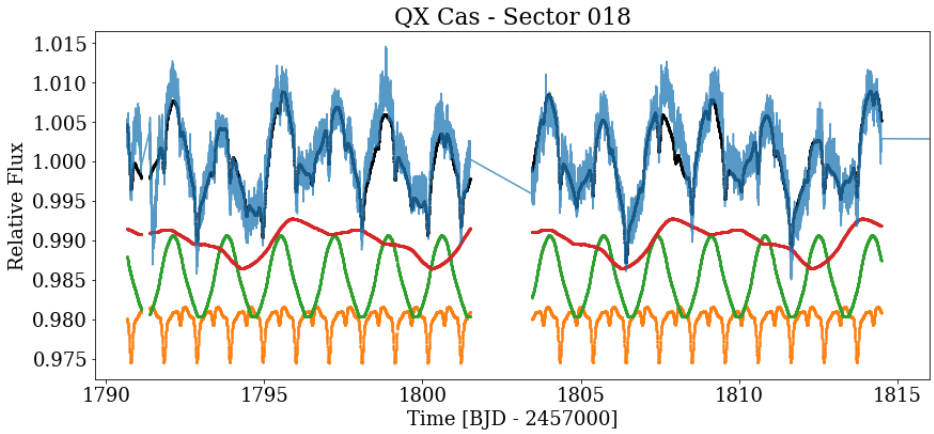

Figure 58: Empirical Model

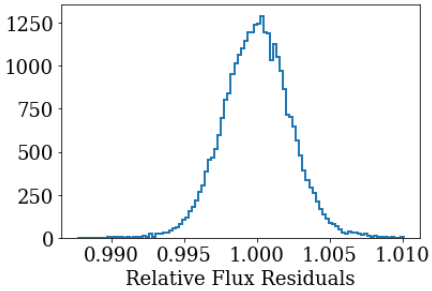

Figure 59: Decent residuals?

will eventually make more observations of QX Cas in another sector, but we'll have to wait a while for that data to be publicly available. As to simulating this in PHOEBE - you really don't want to. PHOEBE's issues with atmospheric tables are abundant. Hot stars simply cannot be analyzed smoothly with PHOEBE. The neural network PHOEBE was trained on does not like hot stars. Other, more classical tools, should be used in order to fit curves to this system. Some of the results which Zola et al arrived at show that the stars of QX Cas were not really far apart. A more probable cause for this variation is that the system became or is becoming a contact binary. Contact binaries are even more difficult to analyze in PHOEBE, so other methods will have to be looked at.

8 Final Thoughts

Binary stars are hard. I would have preferred a little more handholding at the beginning because the sheer variety and depth of binary star configurations lead to all sorts of light curves. Perhaps a way to solve an easy binary star could have been shown. I ended up liking the program, however, and even though my case study of a real-life star ended up not going anywhere, I still learned a lot of useful statistics, tricks, and tools that'll surely be helpful going forwards.