Development of An Eclipse Mapping Routine Using Python for Analysis of *Kepler* Data

NATHAN OLIVER SMITH

Supervisor: Dr. Richard Olenick

A thesis submitted in fulfilment of
the requirements for the degree of
Bachelor of Science

The Department of Physics
Constantin College
The University of Dallas
Irving, Texas

22 December 2019

Abstract

An extensive development and implementation of the eclipse mapping method using Python and its relevant libraries is discussed. The code utilizes Scipy's minimize function along with its various solving methods such as Nelder-Mead, Conjugate Gradient (CG), and Sequential Least Squares Programming (SLSQP). These methods are used to solve the maximum entropy equation with a chi-squared constraint to the observed photometric light curve. These methods are first evaluated on two dimensional Gaussian test data with no chi-squared constraint, and then used to image the accretion disk of Cataclysmic Variable KIC 201325107, revealing its Gaussian structure. Moreover, the structure of the code, along with potential design flaws, other errors, and parameter effects on computational time are examined. Factors such as the variance within the Gaussian weighting algorithm, the resolution of the disk image, the number of points within the observed light curve data, and the constraint level of the algorithm can drastically effect the quality of the image. These factors are analyzed and discussed in Chapter 4. The above methods and parameters are then considered as a whole and conclusions are drawn regarding the steps for further research. Finally, the the code's GitHub repository is discussed for version control and open source development.

Acknowledgements

I would like to thank the Donald A Cowan Institute at the University of Dallas for their financial support and Dr. Richard Olenick for his invaluable guidance and feedback. I would also like to thank my parents, Laura and Darrell for their incredible support and making my education at the University of Dallas possible.

Contents

List of Figures

CHAPTER 1

Introduction: Algol Binaries and Accretion Disk Formation

1.1 Binary Characteristics

Of the vast number of systems in the cosmos, roughly half contain multiple stars. Including Triple and Quadruple star systems. The most common, however, is the binary star system, which involves two stars orbiting around a common center of mass known as the barycenter. Within this class of star system, the characteristics can vary greatly. For example, the period of the orbit can range from a few minutes, to a few years. In turn, this disparity dramatically impacts orbital dynamics and stellar evolution, creating a variety of binary sub-classes and beautiful stellar fireworks.

1.1.1 Formation

Theoretically, binaries can form in two ways. The first of which occurs when two stars pass closely together and fall into each others gravitational pull, forming an orbit. While this is not impossible, it is unlikely due to the precise circumstances that need to exist in order for this type of formation to occur. For example, recall that a bound system has a negative total energy. In contrast, the total energy of an unbound system is positive. Gravitational capture is only possible if the system's total energy can shift from positive to negative. Hence, there must be some form of energy loss. Energy loss could have a range of sources, such as internal energy loss -as a result of tidal distortion- or loss due to a third body. For tidal distortion to occur, the stars must pass at precise distance; an extremely close approach can result in merging, while a distant approach will has no chance for the correct energy loss. These circumstances are very rare, and could only occur in the cores of globular clusters and other dense star fields.

Due to the scarcity of circumstance and plethora of observation, gravitational capture cannot be the primary formation process.

The second, and more likely case, occurs during proto-star formation within molecular clouds. These 'stellar nurseries' are massive clouds of gas that can spontaneously collapse in the right conditions, forming the cores of soon to be main sequence stars. Statistically, gas clouds of this size can have a range of densities at different points. Knowing this fact, consider the basic case for the minimum radius of gravitational collapse -Jeans Length- neglecting magnetic fields, rotation, and turbulence:

$$R_J \propto (T/\rho)^{1/2} \tag{1.1}$$

For the sake of argument, let the temperature " T " be fairly constant throughout the cloud. Given the likely variance of density " ρ " relative to volume, this stellar nursery could certainly satisfy Eq. 1.1 in various spaces and collapse into multiple points at once. This is known as cloud fragmentation. These cloud fragments must also collapse quickly to avoid re-merging. Furthermore, the cloud must be spherically asymmetric and not concentrated about the center. Cloud fragmentation likely occurs much more often than gravitational capture. Furthermore, many binaries contain pre-main-sequence stars which suggests that binary system and proto-star formation occur at relatively the same time.

1.1.2 Algol Type Eclipsing binaries

A special subclass of these systems is the Algol Type Eclipsing binary. Eclipsing means that the stars in the binary system are orbiting such that one star eclipses the other, blocking out a portion or all of its partner's light. Consider Figure 1.1 on the next page. Notice the orbital procession by which the smaller primary star and larger companion eclipse each other. This corresponds to the brightness measurement over time graph in the bottom section of the figure. The dips in the data correlate with the full eclipses of the binary. Furthermore, the term 'Algol Type' refers to two main-sequence stars which are semidetached. Notice in Figure 1.1 how the stars form a tear drop shape as they orbit. A system is 'semidetached' when only the larger

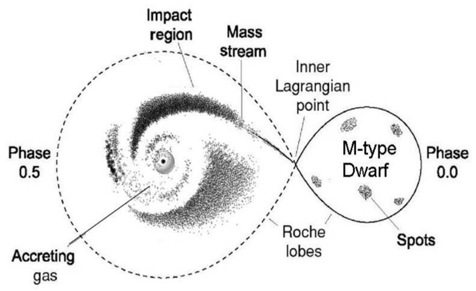

FIGURE 1.1: A diagram of a Cataclysmic Variable system displaying the accreted matter flowing from the secondary "M-type Dwarf" component two the primary component through the L_1 point.

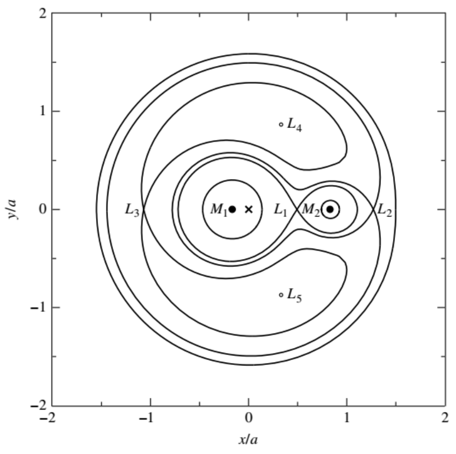

FIGURE 1.2: The Roche Lobe Equipotiental Lines for a binary system, which show where the gravitational potential is constant in space. (Carroll and Ostlie 2017)

companion star has this shape. This phenomena is due to the gravitational tidal forces from the system's close orbit. The specific shape of the tear drop is governed by the Roche Lobe Surface Equipotiental which is crucial to understanding the nature of these binaries and will be discussed in the next section.

1.1.3 Roche Lobe Equipotiental and Dwarf Novae

The Roche Lobe Equipotiental Surface is incredibly important when understanding Semi-detached Eclipsing binaries because it defines the effective gravitational potential at any point in the system. In Figure 1.2, the lines in the diagram represent where the gravitational potential is equal to some value. In particular, notice the point the 'L points'. These are known as *Lagrange Points*, which define special points in space where the effective gravitational potential is zero. If one was to place an object at one of these points, the object would not fall because the gravity of each star, combined with the centrifugal force of the orbit, act equally. The Roche Lobe Potential is defined as

$$\Phi = -G\left(\frac{M_1}{S_1} + \frac{M_2}{S_2}\right) - \frac{1}{2}\left[\frac{G(M_1 + M_2)r}{a^3}\right]^2 \tag{1.2}$$

Where S_1 and S_2 are an objects distance from the binary stars with masses M_1 and M_2. *a* is the semi-major axis, *G* is the gravitational constant, and *r* is the distance of the object from the center of mass of the binary. The characteristics of the first Lagrange point, denoted 'L1' in Figure 1.2, gives matter the ability to flow, transferring mass between stars. The distance from the first Lagrange point to the secondary star satisfies the equation

$$\frac{M_1}{(a-r)^2} = \frac{M_2}{r^2} + \frac{M_1}{a^2} - \frac{r(M_1 + M_2)}{a^3} \tag{1.3}$$

Where *r* is the distance from the secondary star to the first Lagrange point. Notice the tear drop shape around the larger star in Figures 1.2 and 1.3. This is known as the star's 'Roche Lobe' which defines the points where the effective gravitational potential is zero. In other words, where $\Phi = 0$. At certain stages of stellar evolution, this star will expand rapidly, filling its Roche Lobe. In consequence, the outer atmosphere of the star will begin to 'fall off', flowing through the L1 point towards the smaller star, 'M2', and spiral in the same direction of the orbit to form an accretion disk. This process is known as *Mass Transfer*. When a white dwarf acts as the primary star in a system of this type, a *dwarf nova* can occur. Dwarf novae belong to a broader class of binaries called *cataclysmic variables*, which are noted for their recurrent periods of outburst which increase their brightness by a up to a factor of 10. These white dwarfs have an average mass of 0.86 $M_\odot$, compared to 0.58 $M_\odot$ for their isolated brethren, and the less massive secondary star is usually a G spectral type.

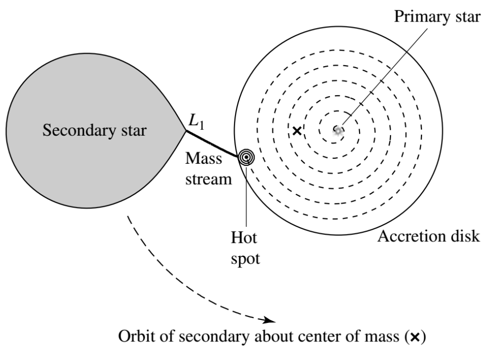

FIGURE 1.3: This mass transfer diagram shows the mass flow between the secondary and primary components and the spiraling nature of the gas trajectory as it spirals down onto the white dwarf. The mass stream forms a 'hot spot' on the disk due to collision with accretion disk. (Carroll and Ostlie 2017)

1.1.4 Mass Transfer and Accretion Disk Characteristics

Luminosity in the accretion disk due to mass transfer is the defining characteristic of dwarf novae. Essentially, mass transfer is the process by which matter becomes gravitationally unbound to a body and becomes bound to a more compact body such as a white dwarf. Due to the rotation of the system, as mass accretes around the white dwarf, a thin disk of hot gas forms within the plane of the orbit. In dwarf novae, this process takes on some surprising qualities. The first dwarf nova was observed in 1885, but it wasn't until 1974 that their precise characteristics were discovered, when astrophysicist Brian Warner argued that the observed periodic outbursts were due to the luminosity of the accretion disk, as opposed to the fusion of accreted hydrogen in classical novae. Modern theory, known as the Tidal Disruption Model (Shcherbakov et al. 2013), suggests that dwarf novae outbursts are due to instability in the accretion disk, where the gas reaches a critical temperature and effects the total viscosity, which is the internal friction that converts the directed orbital motion of the gas into random thermal motion. This, in turn, causes a greater influx of mass from the secondary star, compounding on the brightness and illuminating all of the gas orbiting the white dwarf. At certain stages of disk formation, enough orbital motion can be converted into heat to cause the gas to spiral down into the primary white dwarf star (see Figure 1.3). Little is known about the physical mechanism that causes accretion disk outbursts because the typical model of weak particle interactions does not account for the amount of heat generation. Other possibilities, such as turbulence from convection, and random gas motions can provide better models of dwarf novae outburst.

Like a star, a thin accretion disk can be treated as a black-body, emitting a continuous spectra at each radial distance. This provides useful information about the theoretical temperature profile and general structure. Because the disk's mass is substantially less then the mass of the primary or secondary star, the disk is only influenced by the primary white dwarf. Hence, the total kinetic energy can be estimated as:

$$E = -G\frac{M_{primary}M_{disk}}{2r} \tag{1.4}$$

Where *r* is the distance from the white dwarf. As stated before, the directed orbital motion of the disk is being converted into heat loss due to internal friction, causing the gas to fall to the white dwarf surface. If we assume the distance is *steady state*, where it is not changing with time, and that the mass transfer rate $\dot{M}$ is constant, we can use conservation of energy to say that the amount of energy radiated is equal to the amount of gas that moves in and out of an imaginary ring around the disk in some time interval:

$$dE = \frac{dE}{dr}dr = \frac{d}{dr}(-G\frac{M_{primary}M_{disk}}{2r})dr = G\frac{M_{primary}\dot{M}t}{2r^2}dr \tag{1.5}$$

Moreover, because luminosity times time is related to energy:

$$dL_{ring}t = dE = G\frac{M_{primary}\dot{M}t}{2r^2}dr \tag{1.6}$$

Using the Stephan-Boltzmann law we can also say that

$$dL_{ring} = 4r\pi\sigma T^4 dr = G\frac{M_{primary}\dot{M}t}{2r^2}dr \tag{1.7}$$

Solving for temperature, we can find the disk temperature at radius *r*.

$$T(r) = \left(\frac{GM_{primary}\dot{M}}{8\pi\sigma R^3}\right)^{\frac{1}{4}} \left(\frac{R}{r}\right)^{\frac{3}{4}} \tag{1.8}$$

For the purposes of accretion disk imaging, Equation 1.6 can be simplified with the form

$$T \propto \left(\frac{1}{r}\right)^{\frac{3}{4}} \tag{1.9}$$

Eq. 1.9 will be used to compare the temperature profiles of generated disk images. Comparing and contrasting the temperature profile of the reconstructed image gives valuable information about abnormal structures in the disk. On the other hand, consistent residuals could also reveal errors in the code and give insight to future improvements.

1.2 Statement of the Problem and the Usefulness of Eclipse Mapping

The goal of this particular method of accretion disk imaging is to gain insight into the structures and nature of accretion disks by taking advantage of the geometric orientation of dwarf novae to our planet. This method provides an alternative to Spectral Tomography which requires spectroscopic data that is less accessible to many who want to make progress in this field. The method discussed in this paper, Eclipse Mapping, merely requires photometric data, which is easily accessible via Kepler's MAST database. Access to this data makes it possible on Kepler objects similar to those from (Montgomery et al. 2017) and (Scaringi et al. 2013). However, the algorithm requires some a priori knowledge of the system's orbital mechanics which can also be determined through light curve analysis. Furthermore, with the use of python's versatile set of libraries, cloud computing, and big data applications such as Spark, this method can be scaled, automated, and sped up simultaneously to generate results for a vast variety of dwarf novae. These improvements also allow for faster testing of different parameters to generate optimal results. The problem of eclipse mapping can be expressed by asking three questions: What is the nature of eclipse mapping? What are the results of optimal and sub-optimal parameters? Finally, how can these processes be scaled and optimized with modern data applications and open source development?

CHAPTER 2

Formalism

2.1 Assumptions

The eclipse mapping method studied and developed by (Baptista and Steiner 1993), (Horne 1985), and others, is a computational approach that utilizes the eclipse geometry of the secondary star orbiting around the primary at any point in time. Each instant during the orbit has an individual eclipse profile which can be studied to determine the brightness distribution of the accretion disk around the primary white dwarf. However, some fundamental assumptions must be made about the physical characteristics of the disk: First, the surface of the secondary star is defined by the Roche Lobe Potential. Secondly, the matter contributing to the brightness distribution is constrained to the orbital plane. Finally, the emitted radiation is independent of orbital phase, which describes the specific rotational position of the orbit. The first two are basic assumptions about the qualities of dwarf novae and accretion disks. The third, however, may be problematic when performing analysis on anomalies such as outbursts and super outbursts. This is important to keep in mind during the future development of the Eclipse Mapping method.

2.2 General Characteristics

The problem of the Eclipse Mapping Method has two primary components: The eclipse map, and the maximum entropy equation, which are used in conjunction to reconstruct the accretion image. The eclipse map generates an eclipse geometry at every phase point, while the maximum entropy equation gives the most physically probable solution for the brightness distribution.

2.2.1 The Eclipse Map

The purpose of the eclipse map is to connect light curve data space with the physical space surrounding the white dwarf. This physical space is defined by a pixel grid that lies within the orbital plane of the orbit, where the area of each pixel corresponds to unique area in the accretion disk. Each pixel area is defined as

$$A_{pixel} = \frac{(\gamma R_{L1})^2}{N^2} \tag{2.1}$$

Where R_{L1} is the distance to the first Lagrange Point, *N* is the length of one side of the pixel gird and γ is a scaling factor that can be assigned arbitrarily. I left this parameter equal to one due to time constraints during testing, but experimentation with this parameter is greatly encouraged.

In essence, the eclipse map at some phase point ϕ denotes whether a particular pixel -and hence a particular area of the disk- is eclipsed. This gives valuable information about whether that area of the accretion disk is contributing to the light curve in that instance. The total flux of the disk, f_ϕ, can be calculated by taking the some of all visible pixels at phase ϕ:

$$f_\phi = \frac{\Theta^2}{4N^2} \sum_{j=1}^{N^2} I_v(j) V(j, \phi) \tag{2.2}$$

Where $I_v(j)$ is the is the intensity of pixel *j*, and $V(j, \phi)$ is the eclipse map visibility function at phase ϕ for pixel *j*. The output of this function is either 1 or 0, but a fraction can also be used to calculate a more detailed light curve. Increasing the resolution of the grid N^2 can achieve a similar effect at the expense of computational time. Θ is a constant defined as

$$\Theta^2 = \left(\frac{R_{L1}}{D}\right)^2 cos(i) \tag{2.3}$$

Where *i* is the orbital inclination, and *D* is the distance from the system to Earth. The distances to cataclysmic variables can be uncertain. Ideally, better values of Θ can be found through eclipse mapping analysis. (Baptista 2001) uses units of solar radii per kiloparsec for the value of Θ.

In order to determine whether a pixel is eclipsed at phase ϕ, consider a computational model of the orbit scaled by the the distance from the primary star to the R_{L1} point. Each pixel on the accretion disk grid, if visible, will direct line of sight from earth. Naturally, if it is eclipsed, this line of sight will be blocked. Therefore, a simple way to determine pixel visibility is to determine the orbital position at ϕ and check whether some pixel *j* is eclipsed by the secondary star with given radius R_2. If R_2 is uncertain, the Roche Lobe equation can be used determine the visibility of the pixel. (Horne 1985) uses an estimate radius which encloses the companion star. If the pixel's line of sight intersects with this radius, then the Roche Lobe equation is solved at short intervals to determine whether the pixel is eclipsed. Ideally, this method will be supported in an updated version of the code described in Chapter 3. Currently the code only supports use of a given radius R_2. In Kepler data, many orbital parameters are already known, which makes the construction of the orbital model possible to determine visibility. Details regarding code structure and implementation will be discussed in Chapter 3.

2.2.2 The Maximum Entropy Equation

The model used for the maximum entropy function is a variation of a metric known in information theory known as the Kullback–Leibler Divergence (Moulin and Johnstone 2014), or relative entropy information gain. This function is written as

$$D_{KL}(P||Q) = -\sum_{X \in x} P(x) log\left(\frac{P(x)}{Q(x)}\right) \tag{2.4}$$

which is equivalent to

$$D_{KL}(P||Q) = \sum_{X \in x} P(x) log \left(\frac{Q(x)}{P(x)} \right) \tag{2.5}$$

This equation determines the amount of information lost when a known function *Q* is used to approximate *P*. In other words, how many more bits of information are needed to accurately represent *P*? In the eclipse mapping problem, the KL Divergence is denoted as the entropy function *S*, representing the information entropy of a solution set of pixels to some weighted default image (Grünwald, Dawid et al. 2004).

$$S = -\sum_{j=1}^{N^2} p_j log \left(\frac{p_j}{q_j} \right) \tag{2.6}$$

Where

$$p_j = \frac{I_j}{\sum_k I_k} \qquad q_j = \frac{D_j}{\sum_k D_k} \tag{2.7}$$

In Eq. 2.7, I_j is the intensity of pixel *j*, and p_j normalizes this intensity over the sum of all pixel intensities. Similarly, q_j takes the weighted intensity from the default map D_j and normalizes over the sum of all weighted intensities. The default map is defined below as

$$D_j = \frac{\sum_k w_{jk} I_k}{\sum_k w_{jk}} \qquad w_{jk} = e^{-\frac{(R_j - R_k)^2}{2\Delta^2}} \tag{2.8}$$

Where R_j and R_k are the distances of pixels *j* and *k* from the center of the grid. There are a variety of weight functions for w_{jk} which emphasize different aspects of the disk image solution set *I*. The particular form of w_{jk} in Eq. 2.8 is, in essence, a convolution that emphasizes the radial structure of image for scales greater than Δ. Consequently, Δ acts like a smoothing value for the whole image. Other forms of w_{jk} can be found in the appendix.

With these definitions, Eq. 2.6 must be maximized with some constraint to observed data to find the most physically probable solution for *I*. Due to restrictions in the python library, Eq.

2.6 must be first made positive and then minimized. Hence, the solution for *I* corresponds to the minimum of the positive form of the entropy function S

$$I \mathrel{\widehat{=}} min\left[\sum_{j=1}^{N^2} p_j log\left(\frac{p_j}{q_j}\right)\right] \tag{2.9}$$

when constrained by the chi squared function,

$$\chi^2 = \frac{1}{M}\sum_{\phi=1}^{M}\frac{(f_\phi - d_\phi)^2}{\sigma_\phi^2} \tag{2.10}$$

where f_ϕ is the calculated flux defined by Eq. 2.2, d_ϕ is the observed flux, σ_ϕ is the uncertainty of d_ϕ, and *M* is the number of data points in the observed light curve. A common method of constraint is to set up the problem with Lagrangian constraints by defining a χ_{aim} value, which will determine how much the maximum entropy algorithm will constrain the solution for *I* to the data. The constraint function will then be defined as

$$0 = \chi^2 - \chi^2_{aim} \tag{2.11}$$

Where 0.6 as χ^2_{aim} per degree of freedom usually gives the smoothest solutions and accounts for noise (Baptista and Steiner 1993). A value of 1 will yield the highest constraint to the light curve, but can take significantly longer to solve and will not account for flickering. The algorithm may even fail to achieve this value in certain circumstances. In contrast, a smaller value will over-prioritize the entropy function and fail to produce and accurate image.

CHAPTER 3

Python Implementation and Methods

3.1 Overview

The primary goal regarding the structure of the eclipse mapping program is to be as versatile as possible, enabling others to automate different aspects of the code and test data with many different parameters and call complex analysis processes with single lines of code. The program is object oriented and consists of two primary classes. the *`Binary`* class and the *`Processes`* class. The `Binary` class is concerned with organizing and storing data regarding each binary system being analyzed. It contains data such as the orbital model used to construct the eclipse map for all phase points, as well as the Default map and w_{jk} values defined in Eq. 2.8. The `Processes` class deals with common analysis methods used across all objects in the `Binary` class, such as the the minimization process suggested in Eq. 2.9 and other analysis processes. Each class has its own discussion in the next sections. The current version of the `Processes` class contains mostly static methods and no parameters to instantiate in the `__init__` method. Later versions could incorporate definable meta parameters when initializing a `Processes` object, allowing for a more diverse analysis process. Moreover, the code currently uses methods from the *Phoebe* python package, which is only supported in Linux and Mac. Ideally, I would like to remove these dependencies to provide full support in Windows. Using this code in Windows currently requires the installation of python on a Linux Subsystem in Windows 10, which can then be routed as a remote interpreter in certain development environments.

3.2 The Binary Class

3.2.1 Inputs

The `Binary` class handles all data pertaining to a specific binary system. Multiple binary classes can be initialized and processed through looping and parallel processing methods. Each binary object has a corresponding data folder that stores the observed light curve data, called `light_curve_data.csv`, and a parameter file called *parameters.csv*. There is an example of each file on Github in the example outputs file. Below I will briefly define each of the input parameters as they are named in the code, discussing their units and how they are used.

radius_secondary_given. This is a Boolean value that indicates whether the program should attempt to estimate the radius of the star. The default for this value is True because estimation is not yet supported in the program.

ecc. The eccentricity of the orbit. The default is 0 since accreting dwarf novae have little if any eccentricity.

N. The pixel length of the accretion grid along one axis, making the pixel resolution equal to N^2.

`chi_AIM`. The value of chi squared per degree of freedom to constrain the algorithm. A typical value is 0.6.

period. The orbital period in days.

delta. The width of the Gaussian weight function, denoted as Δ in Eq. 2.8.

`system_name`. A string denoting the name of the system. After analysis, the program will save the outputs in its current working directory, within a sub-directory called `system_name`.

distance. The distance from the earth to the binary in kilo-parsecs.

incl. The inclination of the orbit in degrees, where 0 degrees would yield an orbital plane perpendicular to out point of view.

`radius_primary`. The radius of the primary star in solar radii.

`radius_secondary`. The radius of the secondary star in solar radii.

3.2.2 Lightcurve Data Inputs

The `light_curve_data.csv` contains three columns in the following order: the barycentric Julian Date of the observation, the observed brightness in units of flux, and the corresponding uncertainty, denoted in the file as 'JDtimes', 'observed fluxes', and 'observed uncertainties' respectively. Examples of the data files can be found on Github.

3.2.3 The `build_orbital_model` Method

Besides a basic normalization process, the `build_orbital_model()` method is the first process run when the analysis is initiated. This section of the code builds functions of the primary and secondary orbit that return the x-y-z coordinates in units of *RL1* for some phase point ϕ, and assigns them as attributes to the binary object. This design allows for easy automation and calling in most situations I anticipated in the code structure. The imports and variable declarations are defined below.

```python
def build_orbital_model(self):
    from numpy import sin, cos, deg2rad
    a1 = self.sma_primary # semi-major axis of primary orbit
```

```
        a2 = self.sma_secondary # semi-major axis of secondary orbit
        incl = self.incl # orbital inclination
        RL1_from_primary = self.RL1_from_primary
        phi0 = 0
        phi0 = deg2rad(phi0)

        def primary_orbit(phi):
            x = -a1 * cos(phi + phi0) * cos(incl) / RL1_from_primary
            y = -a1 * sin(phi + phi0) / RL1_from_primary
            z = -a1 * cos(phi + phi0) * sin(incl) / RL1_from_primary
            return [x, y, z]

        def secondary_orbit(phi):
            x = a2 * cos(phi + phi0) * cos(incl) / RL1_from_primary
            y = a2 * sin(phi + phi0) / RL1_from_primary
            z = a2 * cos(phi + phi0) * sin(incl) / RL1_from_primary
            return [x, y, z]

        self.primary_orbit = primary_orbit
        self.secondary_orbit = secondary_orbit
```

Where `a1`, and `a2` are in solar radii. `phi0` is in radians and can be arbitrarily assigned to modify the starting point of the visibility matrix. After these lines, I define a method which returns the x-y-z coordinates given some phase point and assign them as attributes to the binary object. It is important that `binary.build_orbital_model()` is called before any calculations are done. Otherwise, the orbit attributes will be empty and an error will occur. Usually, there is no need to call this method directly, since the `Processes` class calls these methods in the correct order automatically. This will be discussed in more detail in a later section.

3.2.4 The `construct_phasepoints` Method

Another important component to this algorithm is a consistent means to measure the phase of each data point in the observed light curve. The `construct_phasepoints` processes the light curve data creates a set of phase points for each observed data point. This set of phase points is then assigned as an attribute to the binary object for later use. Below is the method for this process.

```
1  def construct_phasepoints(self):
2      '''
3      :return: a list of phase points assigned to self.phase_points
4      '''
5      from numpy import amin, where, asarray
6      jdtimes = asarray(self.JDtimes)
7
8      phase0 = self.phi0 / 360
9      constructed_phases = []
10
11     for i in range(len(jdtimes)):
12         new_phase_point = phase0
13         constructed_phases.append(new_phase_point)
14         if i + 1 >= len(jdtimes):
15             break
16         delta_phase = (jdtimes[i + 1] - jdtimes[i]) / self.period
17         phase0 += delta_phase
18
19     self.constructed_phasepoints = asarray(constructed_phases)
```

In line 8, `phi0` is the assumed phase starting point of the light curve in degrees. Typically, this will be the secondary minimum, giving `phi0` value of 0.5 or 180 degrees. This value can be assigned in the parameter file discussed in 3.2.1. Line 11 initiates a for loop which measures the distance between the current phase point and the next phase point. This phase distance is then added to `phase0` in line 18. `phase0` is updated at the end of every loop and then added to the list of constructed phase points at the next interval in line 14. Finally, the list is assigned as an attribute to the binary object.

3.2.5 Accounting for Pixel Positions During Orbit

It is also critical to account for the motion of each pixel throughout the orbit. The accretion grid exists within the orbital plane, and is rotating in sync with the binary orbit such that the same side of the grid is always facing the secondary star. To account for these effects, the pixel coordinates must be converted to orbit space in units of *RL1*, rotated about the y-axis by orbital inclination θ, then rotated about the z-axis by phase $\phi' = \phi + \phi_0$ in radians. Because the the top left corner of a numpy grid has the coordinates [0,0], they must be transformed about the center of the grid:

$$x_c = x - \frac{N-1}{2} \qquad y_c = \frac{N-1}{2} - y \qquad z_c = 0 \tag{3.1}$$

The rotation matrices about the y and z axes are given by

$$R_y(\theta) = \begin{bmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{bmatrix} \qquad R_z(\phi') = \begin{bmatrix} cos(\phi') & -sin(\phi') & 0 \\ sin(\phi') & cos(\phi') & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(3.2)

The conversion factor from pixel to orbit space, β, is simply the square root of Eq. 2.2.

$$\beta = \frac{\sqrt{\gamma} R_{L1}}{N} \tag{3.3}$$

With these definitions, a transformation from pixel space to rotated orbit space can be defined as

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \left[\beta R_z(\phi') R_y(\theta) \begin{pmatrix} x_c \\ y_c \\ z_c \end{pmatrix} \right] + \begin{pmatrix} x_{wd} \\ y_{wd} \\ z_{wd} \end{pmatrix}$$

(3.4)

Where the coordinates with subscript *wd* are the coordinates of the primary white dwarf in units of *RL1*. In the code, the `pixel_orb_coords` method performs this transformation and can be found on the code's Github repository. This method is called at line 15 when building the eclipse maps in the next section.

3.2.6 The `build_visibility_matrix` Method

After an orbital model is assigned and phase points are constructed, `build_visibility_matrix()` can be called on the binary object. Similar to the orbital model method, the final output will be assigned as an attribute to the binary object. Namely, the eclipse map for every phase point in the observed data, called the `visibility_matrix`. This name is fitting because the data structure is a three dimensional matrix with dimensions *N* by *N* by *M*. Each two dimensional slice is an eclipse map consisting of ones and zeroes, where the first two axes are the x and y coordinates respectively, and the third axis is the `phi` axis. In short, the data structure is a stack of individual eclipse maps for each phase point ϕ. Consider the following method.

```
1  def build_visibility_matrix(self):
2      from numpy import pi, ndarray, sqrt
3      visibility_matrix = ndarray(shape=(self.N, self.N,
           len(self.constructed_phasepoints)))
```

```
      to_RL1 = self.to_RL1
      for i in range(len(self.constructed_phasepoints)):
         matrix_frame = ndarray(shape=(self.N, self.N))
         # pulling phase point from list of phase points and converting to radians
         phi = self.constructed_phasepoints[i] * 2 * pi
         x_2nd, y_2nd, z_2nd = self.secondary_orbit(phi)
         # self.secondary_contributions.append(0)
         # contrib = self.get_secondary_contributionv2(y_2nd)
         self.secondary_contributions.append(0)
         for x in range(self.N):
            for y in range(self.N):
               x_px, y_px, z_px = self.pixel_orb_coords(x, y, phi)

               # getting the y-z projection of the distance between the orbit
               y_z_distance = sqrt((y_2nd - y_px) ** 2 + (z_2nd - z_px) ** 2)

               secondary_radius_in_RL1 = to_RL1(self.radius_secondary)
               if y_z_distance <= secondary_radius_in_RL1:
                  if phi >= 0.5 * pi and phi <= 1.5 * pi:
                     matrix_frame[x, y] = 1
                  if phi >= 1.5 * pi or phi <= 0.5 * pi:
                     matrix_frame[x, y] = 0
               else:
                  matrix_frame[x, y] = 1
         visibility_matrix[:, :, i] = matrix_frame
      self.visibility_matrix = visibility_matrix
```

Line 3 declares `visibility_matrix` as a 3 dimensional numpy array with the appropriate dimensions. The contents of the array are irrelevant since they will be replaced. Line 5 initiates a loop through the list of constructed phase points from 3.2.4, creating a two dimensional slice called `matrix_frame` (see lines 21 - 26). This matrix frame is assigned to a slice inside `visibility_matrix` which is assigned as an attribute of the binary object after the loop is finished (line 29).

3.2.7 The `get_wjks` Method

Before constructing the entropy equation to minimize, the w_{jk} weights must be defined from Eq. 2.8. Because each *jth* pixel has a unique position relative to all other *k* pixels, each *j* will have its own two dimensional matrix of weights for each coordinate in the accretion grid. Hence, the data structure for the w_{jk} weight matrix is an array with dimensions *N*, by *N*, by N^2, which can be reshaped and iterated through when building the entropy function discussed in the next section.

```python
def get_wjks(self):
    from tqdm import tqdm
    from numpy import ndarray, exp
    N = self.N
    delta = self.delta
    wjk_matrix = ndarray(shape=(N, N, int(N ** 2)))
    frame_number = 0
    # these i and j loops define the jth pixel for the weight function
    for i in tqdm(range(N), ’’.join(["Building weight matrix for ", self.system_name])):
        for j in range(N):
            # centering grid around coordinates [0,0]
            xj = i - ((N - 1) / 2)
            yj = ((N - 1) / 2) - j
            jframe = ndarray(shape=(N, N))
```

```
15            Rj = self.dist_from_center([xj, yj])
16            # these k and l loops define the kth pixel for the weight function
17            # because these loops are inside the i and j loops, k and l will complete a full cycle for every i and j
18            for k in range(N):
19                for l in range(N):
20                    xk = k - ((N - 1) / 2)
21                    yk = ((N - 1) / 2) - l
22                    Rk = self.dist_from_center([xk, yk])
23                    wjk = exp(-(Rj - Rk) ** 2 / (2 * delta ** 2))
24                    jframe[k, l] = wjk
25            wjk_matrix[:, :, frame_number] = jframe
26            frame_number += 1
27        self.wjks = wjk_matrix
28        return wjk_matrix
```

Similar to the visibility matrix, line 6 initiates a numpy array which will be edited with a loop below. The structure in lines 9-26 consists of four nested loops, the top two (lines 9 and 10) loop through the coordinates for each *jth* pixel, where the variables `i` and `j` are the coordinates for pixel *j*. Variables `k` and `l` govern the coordinates for the *kth* pixel. Because the `k` and `l` loops are nested, they complete for every interval of loop `j`. With every completion of a loop, a new two dimensional set of weights is added to `wjk_matrix`, which is assigned as an attribute to the binary object for future use in the construction of the entropy equation.

3.2.8 Constructing the Entropy Equation

The final step before solving for the accretion disk image is to create a function of *S* for the accretion image explicitly in terms of the a the brightness grid *I*.

```
def get_entropy_function(self):
    from numpy import asarray
```

```
3       N = self.N
4       wjks = self.get_wjks()
5       wjks = wjks.reshape((int(N ** 2), int(N ** 2)))
6
7       def entropy_function(I):
8           from numpy import log, nan, inf, nan_to_num
9           D = asarray([sum(a * b for a, b in zip(I, wjks[j, :])) /
                sum(wjks[j, :]) for j in range(len(I))])
10
11          q = asarray([i / sum(D) for i in D])
12          p = asarray([i / sum(I) for i in I])
13          # actual function is - sum(.... but im minimizing the
                positive version to find the maximum of its opposite
14          S = sum([pj * log(pj / qj) for pj, qj in zip(p, q)])
15
16          if S == inf:
17              S = nan_to_num(inf)
18          if S == nan:
19              S = nan_to_num(nan)
20          return S
21      return entropy_function
```

This method defines the function `entropy_function` and returns it as a result upon call. Notice lines 4 and 5 call the `get_wjks` and reshape the resultant matrix for easy iteration in line 9, which implements the default map equation (Eq. 2.8). Line 14 defines Eq. 2.9, though it is not yet minimized. An important note: the variable `I` corresponds to the brightness distribution which will be solved for the final disk image. However, in its data structure is a one dimensional numpy array which will later be reshaped for the final two dimensional image. See the docs on `scipy.optimize.minimize` for details (Kiusalaas 2013).

3.2.9 Solving for the Maximum Entropy Image

The final important step in this process is to minimize the entropy equation using the scipy library. Similar to the other methods discussed in this chapter, the `get_image_solution` function solves for the maximum entropy solution for *I* and assigns it as an attribute to the binary object (Vrielmann et al. 2002), (Skilling 1986).

```python
def get_image_solution(self):
    from scipy.optimize import minimize, NonlinearConstraint
    from numpy import amax, asarray, inf
    import time
    N = self.N
    S = self.get_entropy_function()
    guess = self.gkern().reshape((int(N) ** 2))
    chi_sqrd = self.build_chi_constraint()

    method = self.method
    cons = [{'type': 'eq', 'fun': chi_sqrd}]

    print"Solving entropy equation with resolution: ", self.N, '
       by ', self.N
    start = time.time()
    # method usually = method, but im changing to 'trust-constr'
       for testing, since i think it might use better
       coonstraints


    res = minimize(S, x0=guess, method=self.method, constraints=
       cons)
    stop = time.time()

    solve_time = (stop - start ) /(60**2)
    time_text = ''.join([str(solve_time), ' hrs'])
```

```
23        self.all_params['solve time'] = time_text
24
25        image_entropy = S(res.x)
26        image_solution = res.x
27        sol_max = amax(image_solution)
28        normalized_solution = map(lambda x: x / sol_max,
              image_solution)
29
30        self.image_entropy = image_entropy
31        self.image_solution =
              asarray(normalized_solution).reshape((N, N))
```

Lines 6 and 8 instantiate the entropy function and the chi squared constraint which will be minimized in line 18. Line 6 creates a two dimensional Gaussian distribution which is flattened into a one dimensional array. This will be the initial starting solutions for the minimization process. Line 11 wraps the chi squared function into a constraint dictionary according the scipy docs. Finally, the solution is normalized and reshaped to a two dimensional image, where it is assigned as an attribute to the binary object (lines 25 - 31).

3.3 The Processes Class

The primary purpose of the `Processes` class is to call the binary object methods in the correct order, condensing the analysis process into a few simple lines of code, allowing one to load, compute, and save a binary object data in its own directory.

```
binary_dir = #some directory that houses the binary parameter file
    and light curve data
b = Processes.load_binary(binary_dir)
Processes.build_compute_save(b)
```

Where line 2 creates a binary object out of the parameter file and light curve data, and line 3 calls all of the methods discussed in section 3.2. The results are saved in the working directory of the python file, under `Outputs/Results`. The methods in the above lines can be found on Github.

CHAPTER 4

Results

4.1 Testing the Solving Methods Without Constraints

The Scipy's minimize function has a variety of solving methods for multivariate equations such as SLSQP, Conjugate Gradient, and Others. Each of these methods solves the maximum entropy equation in different ways. Hence, it is important to see them in action when not constrained to the chi squared function because different methods could give different results. Ideally, there should be a method that naturally returns a disk-like solution when the starting point for the minimization is a two dimensional Gaussian, and is not constrained to chi squared. I tested these methods by writing a loop in python which iterates through different resolutions and methods. The figure below shows a series of solutions for the methods I tested. I tested for N values of 10, 15, and 25. For the sake of brevity, I am only displaying the results for N = 15. All results can be found on the code's Git Hub repository, which will be discussed in the final section of this paper.

The method shown in Figure 4.3 does not seem to minimize the image, since a Gaussian point distribution was the starting guess of the algorithm. Figure 4.2 is a good candidate and should be considered in further research. However, the method I chose to implement was SLSQP (Figure 4.1), since it had the fastest solving time and consistently produced the most disk like results. It is important to look at the relative structure of 4.1 instead of the actual values of each pixel. Notice that the image is inverted in the center, which is correlated with pixel dimension. This problem vanishes when a chi squared constraint is applied.

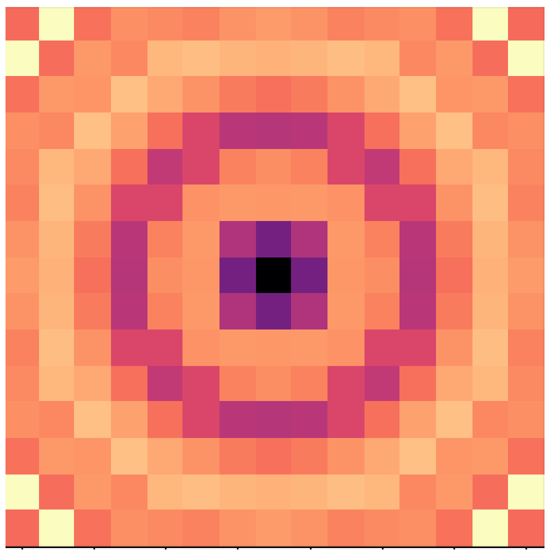

FIGURE 4.1: A solution to the Maximum Entropy Equation without constraints to data, which reveals this solving method's natural bias. Method: Sequential Least Squares Programming (SLSQP), Resolution (N) = 15

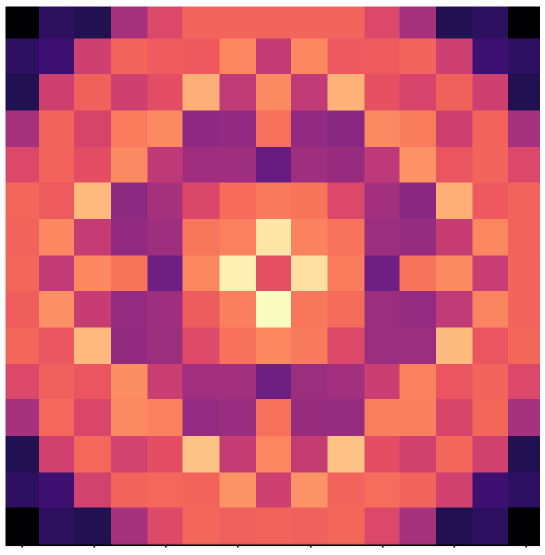

FIGURE 4.2: Unconstrained solution (see Figure 4.1 for description) Method: Truncated Newton (TNC), N = 15

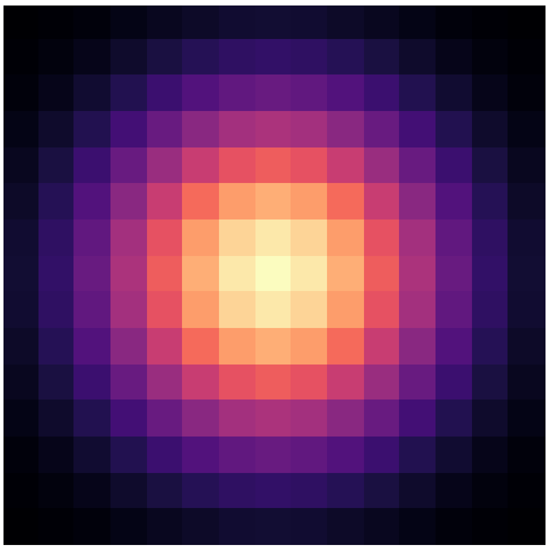

FIGURE 4.3: Unconstrained solution (see Figure 4.1 for description) Method: Conjugate Gradient (CG) , N = 15

4.2 Lightcurve Synthesis and Chi-Square Constraint Results

The algorithm was run over normalized lightcurve data from binary star KIC 201325107, whose parameters were found via Phoebe modelling (Jones et al. 2017). The minimization process used the SLSQP method with resolution values of N = 20 and N = 30. The analysis also produced a temperature profile, which was found my applying the Stephan-Boltzmann law at each pixel brightness, as well as a synthetic light curve plot. Other various parameters were tested such as the Δ from Eq. 2.8. Where a value of 3.0 was used to produce the smoothest images. The algorithm was also run with N = 35, which seemed to be the limit resolution because the disk image resembled the diffraction pattern of a spherical object. Furthermore, a lower resolution image was run over an anomaly-like epoch to assess how the algorithm will solve the maximum entropy equation in this situation. Finally, the algorithm was run over 70 days of data to asses how the accretion disk is smeared over multiple epochs.

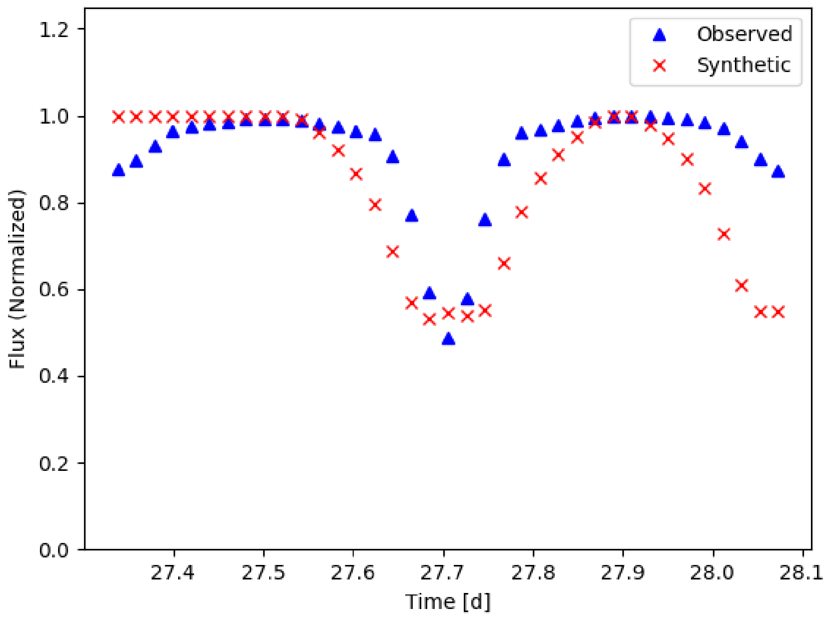

FIGURE 4.4: Observed Versus Synthetic Lightcurve for a single epoch of data of normalized flux for KIC 201325107
Pixel Resolution (N) = 30, Method: SLSQP

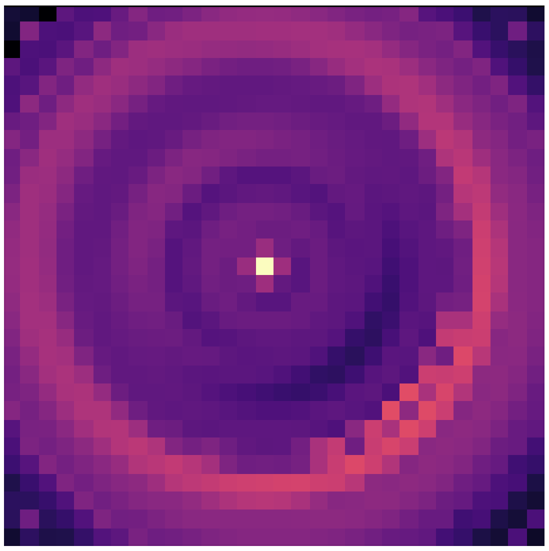

FIGURE 4.5: A maximum entropy solution map corresponding to the synthetic lightcurve in figure 4.4 with a normalized brightness distribution. for KIC 201325107
N = 30, Method: SLSQP

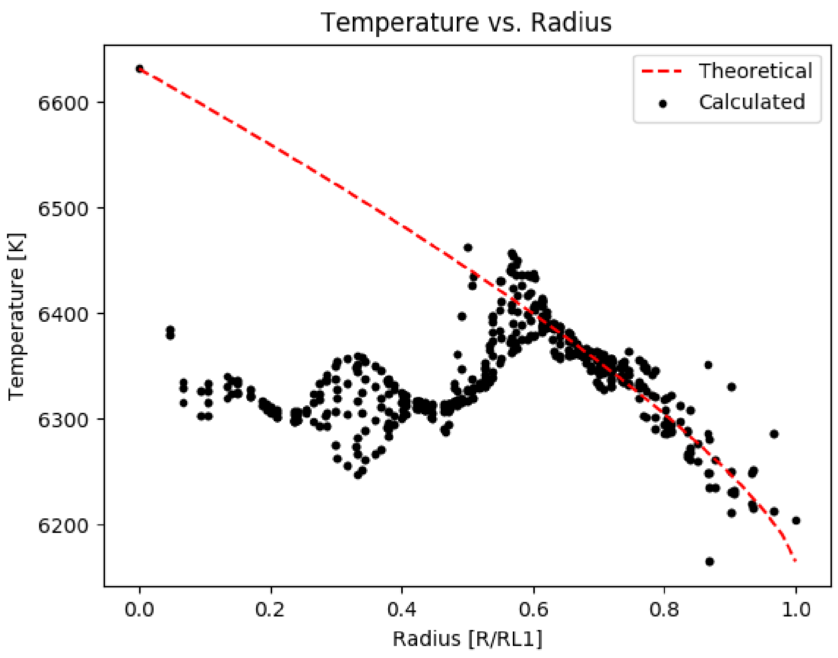

FIGURE 4.6: Temperature Profile calculated at each pixel compared to the theoretical temperature profile for an accretion disk corresponding to Figure 4.5. for KIC 201325107.
N = 30, Method: SLSQP

4.3 Discussion

The Maximum Entropy Algorithm produced a wide range of results which can be found on Github. Generally speaking, it was difficult to consistently produce results. This is likely due to the instability of the code and needs further development, but there were also some very encouraging results. Consider figure 4.5, showing the brightness distribution of the accretion disk. The algorithm does produce the Gaussian hot-spot in the bottom left corner in a similar fashion as (Baptista and Steiner 1993), which is very promising. This also suggests that the conjugate gradient algorithm is one method of many that can be used to solve these computational problems. I encountered many errors with the conjugate gradient method, but this may have been to due to my code design. In this particular case, SLSQP yielded favorable results with a value of N = 30. Figure 4.6 displays the corresponding temperature profile for Figure 4.5, which suggests a reasonable correlation with a typical accretion disk (Eq. 1.9). The discrepancy from 0 to 0.6 should be investigated further. I

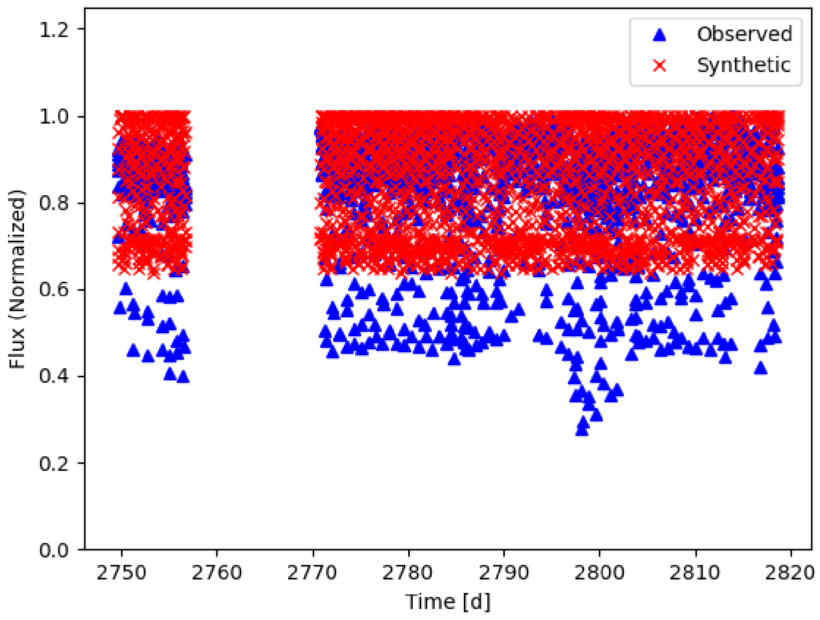

FIGURE 4.7: Observed Versus Synthetic Lightcurve for KIC 201325107 (Smeared)
N = 20, Method: SLSQP

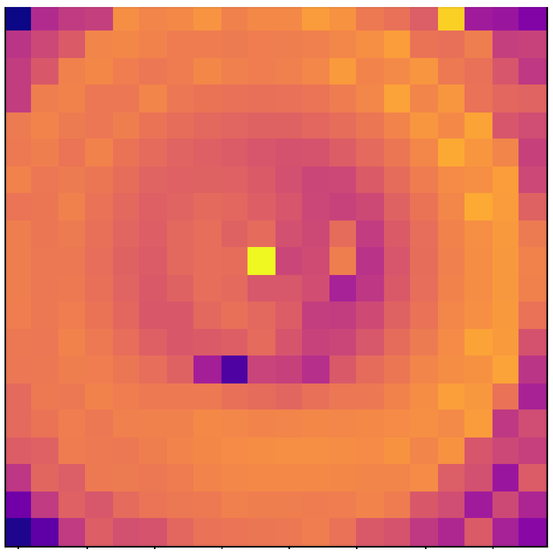

FIGURE 4.8: Smeared Accretion Image for KIC 201325107
N = 20, Method: SLSQP

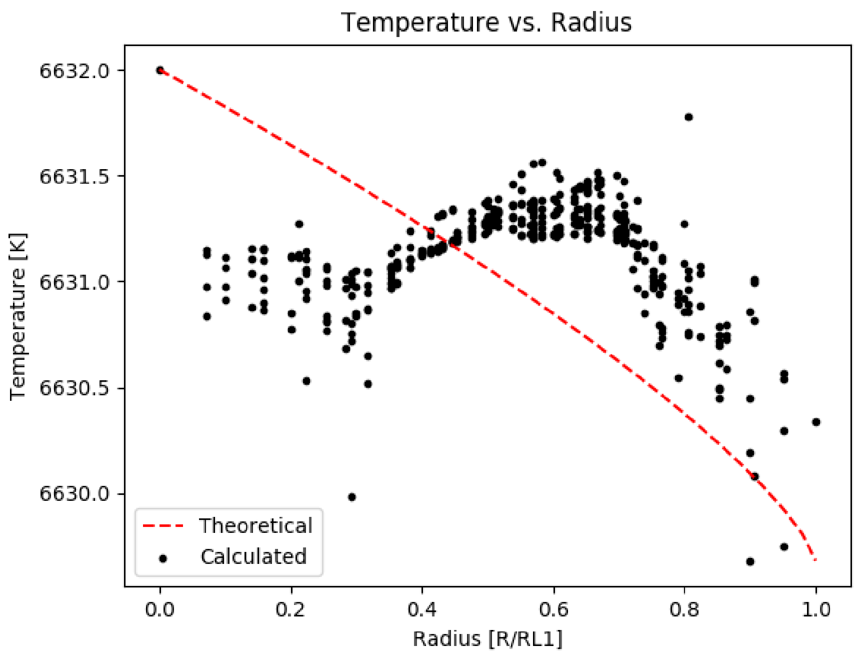

FIGURE 4.9: Smeared Temperature Profile for KIC 201325107
N = 20, Method: SLSQP

believe this is due to the algorithm prioritizing entropy maximization over the chi-squared constraint, which is reasonable given the synthetic light curve divergence in Figure 4.4. I attempted to account for the flux of the secondary lightcurve by adding fractions of the flux proportional to the eclipsing area, but there was no discernible difference in the synthetic light curve after this was accounted for, this may have been due to a programming error, or a mathematical issue. See the `get_secondary_contributionv2` method on Github to view exactly how the secondary contributions were accounted for. None the less, using a single epoch certainly produces the clearest image, as opposed to the multi-epoch image in Figure 4.8. Notice how the Gaussian hot-spot is smeared across the right portion of the disk, compared to the better defined distributions in Figures 4.5 and 4.11. This smearing is also evident in 4.7, where the synthetic light curve does not conform to the shape of the observed data in consequence of the 'averaging' effect at each pixel across all epochs, producing a smeared image. The corresponding temperature profile (Figure 4.9) has little correlation to the standard temperature profile, with a significantly steeper initial gradient than the theoretical model. Similarly, the temperature profile for the anomaly-like solution (Figure 4.12) diverges,

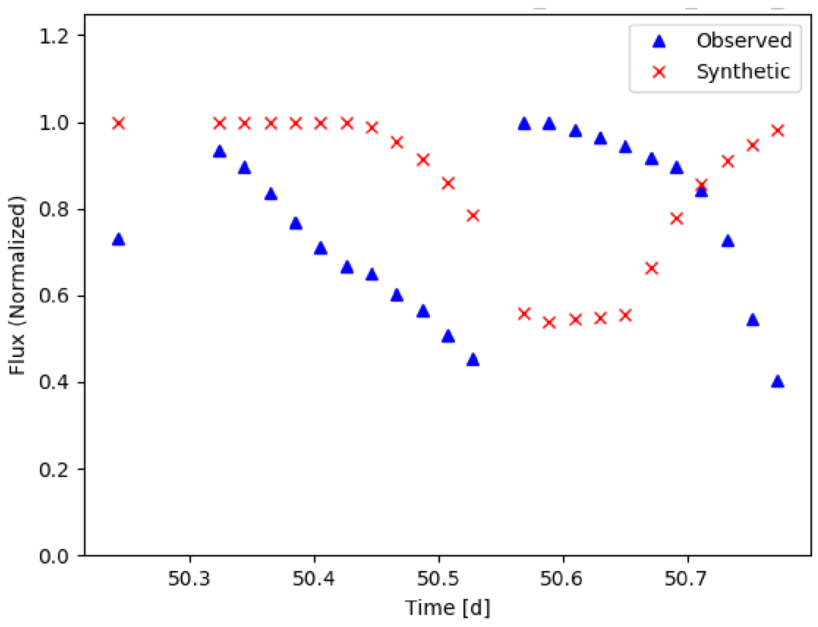

FIGURE 4.10: Observed Versus Synthetic Lightcurve for KIC 201325107 (anomaly-like)
N = 20, Method: SLSQP

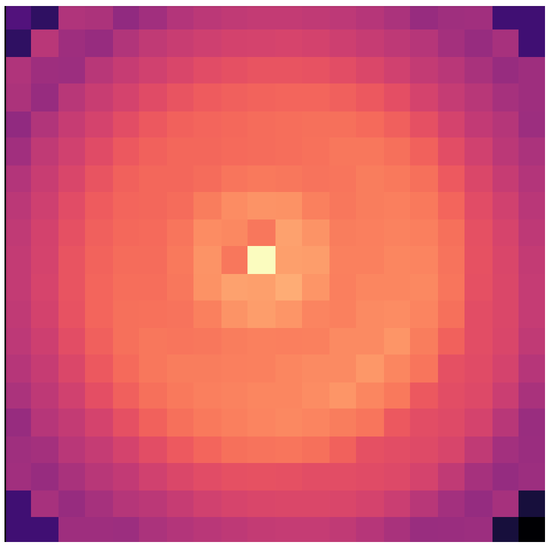

FIGURE 4.11: Accretion Image for KIC 201325107 (anomaly-like)
N = 20, Method: SLSQP

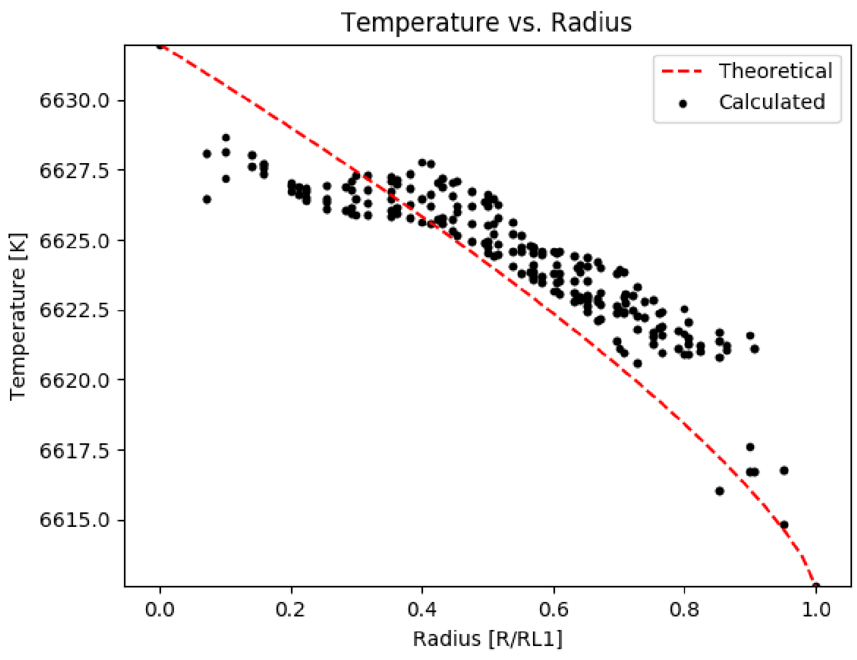

FIGURE 4.12: Temperature Profile for KIC 201325107 (anomaly-like)
N = 20, Method: SLSQP

but less so than Figure 4.9. Despite the lack conformity to the observed data, Figure 4.10 attempts to form a minima-like solution which yields an disk-like image with a Gaussian hot-spot in 4.11, revealing the bias introduced by the radial weighting algorithm (Eq. 2.8). The radial bias is very useful when performing a basic signal analysis, but other biases may be useful to extract anomalous disk structures. A possible innovation of the eclipse mapping method could involve multiple layers of analysis to reveal phenomena such as super-humps in the accretion disk image.

CHAPTER 5

Conclusion

5.1 Areas for Improvement and Future Work

One of the major challenges with the maximum entropy method and eclipse mapping algorithm is managing the bias toward solutions that are products of the weight function and solving method. Consider Figures 4.5 and 4.6. In 4.6, the residuals in the temperature profile from Radius 0.1 to 0.55 diverge from the theoretical model in a clean, oscillatory pattern which corresponds with the natural bias of the SLSQP method in Figure 4.1. Learning to better manage this bias is a high priority. However the particular bias for SLSQP may also help us map super-humps given the solving method's natural resonances (Wood et al. 2011). Furthermore, additional data can be gathered about the thickness of the disk, leading to 3-dimensional models (Rutten 1998). Finally, the eclipse mapping method can be used a different bandwidths, giving more detailed accretion disk maps at different ranges of wavelengths (Baptista et al. 1995). Overall, this project was incredibly insightful for me from both a physics and computer science point of view. This research challenged me to build code from ground up, which allowed me to develop my skills in object oriented programming and code design. These two things alone were extremely valuable. I have also become very comfortable cleaning data and a applying numerical method to develop a model. I believe this skill is very important in physics and any data analytic position that I might pursue.

Bibliography

Baptista, R and JE Steiner (1993). ‘Improving the eclipse mapping method’. In: *Astronomy and Astrophysics* 277, p. 331.

Baptista, Raymundo (2001). ‘Eclipse mapping of accretion discs’. In: *Astrotomography*. Springer, pp. 307–331.

Baptista, R et al. (1995). ‘Hubble space telescope and R-band eclipse maps of the UX Ursae Majoris accretion disk’. In: *The Astrophysical Journal* 448, p. 395.

Carroll, Bradley W and Dale A Ostlie (2017). *An introduction to modern astrophysics*. Cambridge University Press.

Grünwald, Peter D, A Philip Dawid et al. (2004). ‘Game theory, maximum entropy, minimum discrepancy and robust Bayesian decision theory’. In: *the Annals of Statistics* 32.4, pp. 1367–1433.

Horne, Keith (1985). ‘Images of accretion discs–I. The eclipse mapping method’. In: *Monthly Notices of the Royal Astronomical Society* 213.2, pp. 129–141.

Jones, John, Richard Olenick and Arthur Sweeney (2017). ‘Kepler K2 Observations and Modelling of Algol-type binary KIC201325017’. In: *Bulletin of the American Physical Society* 62.

Kiusalaas, Jaan (2013). *Numerical methods in engineering with Python 3*. Cambridge university press.

Montgomery, MM et al. (2017). ‘Photometric observations and Numerical modeling of SDSS J162520. 29+ 120308.7, an SU UMa in the CV period gap’. In: *New Astronomy* 50, pp. 43–51.

Moulin, Pierre and Patrick R Johnstone (2014). ‘Kullback-Leibler Divergence and the Central Limit Theorem’. In: *UCSD Information Theory and Applications Workshop*. Citeseer.

Rutten, RGM (1998). '3D eclipse mapping'. In: *Astronomy and Astrophysics Supplement Series* 127.3, pp. 581–586.

Scaringi, S, PJ Groot and M Still (2013). 'Kepler observations of the eclipsing cataclysmic variable KIS J192748. 53+ 444724.5'. In: *Monthly Notices of the Royal Astronomical Society: Letters* 435.1, pp. L68–L72.

Shcherbakov, Roman V et al. (2013). 'GRB060218 as a Tidal Disruption of a White Dwarf by an Intermediate-mass Black Hole'. In: *The Astrophysical Journal* 769.2, p. 85.

Skilling, J (1986). *Maximum Entropy and Bayesian Methods in Applied Statistics, edited by JH Justice*.

Vrielmann, Sonja, Rae F Stiening and Warren Offutt (2002). 'Physical parameter eclipse mapping of the quiescent disc in V2051 Ophiuchi'. In: *Monthly Notices of the Royal Astronomical Society* 334.3, pp. 608–620.

Wood, Matt A et al. (2011). 'V344 Lyrae: A touchstone Su UMa cataclysmic variable in the Kepler field'. In: *The Astrophysical Journal* 741.2, p. 105.

1 Appendix A

1.1 Other weight functions

As mentioned, there are many options for the weight function w_{jk}, which emphasize various aspects of the accretion disk structure. See the list of equations below.

Most uniform map:

$$w_{jk} = 1 \tag{.1}$$

Smoothest map:

$$w_{jk} = exp(-\frac{d_{jk}^2}{2\Delta^2}) \tag{.2}$$

Most asymmetric map with full azimuthal smearing:

$$w_{jk} = exp\left[-\frac{(R_j - R_k)^2}{2\Delta^2}\right] \tag{.3}$$

Limited azimuthal smearing for a constant angle θ :

$$w_{jk} = exp\left[-\frac{1}{2}\left(\frac{(R_j - R_k)^2}{\Delta_R^2} + \frac{\theta_{jk}^2}{\Delta_\theta^2}\right)\right] \tag{.4}$$

Limited azimuthal smearing for a constant arc length s:

$$w_{jk} = exp\left[-\frac{1}{2}\left(\frac{(R_j - R_k)^2}{\Delta_R^2} + \frac{s_{jk}^2}{\Delta_s^2}\right)\right] \tag{.5}$$