DEVELOPING AN OPEN SOURCE PREDICTIVE SPOT MODEL FOR USE IN BROWN DWARF PHOTOMETRY

by

Kyle Dettman

A thesis submitted to the Faculty of the University of Delaware in partial fulfillment of the requirements for the degree of Honors Bachelor of Science in Physics and Astronomy with Distinction

Spring 2015

DEVELOPING AN OPEN SOURCE PREDICTIVE SPOT MODEL FOR USE IN BROWN DWARF PHOTOMETRY

by

Kyle Dettman

Approved: _____

John Gizis, Ph.D.
Professor in charge of thesis on behalf of the Advisory Committee
Department of Physics and Astronomy

Approved: _____

James MacDonald, Ph.D.
Committee member from the Department of Physics and Astronomy

Approved: _____

Matthew DeCamp, Ph.D.
Committee member from the Board of Senior Thesis Readers
Department of Physics and Astronomy

Approved: _____

Michael Arnold, Ph.D.
Director, University Honors Program

ACKNOWLEDGMENTS

I would like to take this time to thank the people who supported me during this entire process.

First and foremost, I would like to thank my parents for without them I could not have had the college career that I was lucky enough enough to experience. Their constant love and support has buoyed me through these past 4 years and especially these last 2. I would also like to thank my advisor, Dr. John Gizis, for taking a chance on an untested undergraduate to do work for him. Obviously, without this research experience I would not have found a topic for my thesis, but more importantly, his sponsorship of me allowed me to develop valuable skills that I otherwise would not have had the resources to develop.

Finally, I would like to thank my friends in the Society of Physics students for helping me survive both mentally and academically.

TABLE OF CONTENTS

LIST OF FIGURES . . . . . **vi**
ABSTRACT . . . . . **vii**

Chapter

1 INTRODUCTION **1**

1.1 The Kepler Mission 1
1.2 An Overview of Brown Dwarfs 2

1.2.1 Distinction between Planets 3
1.2.2 Evolution 4
1.2.3 Magnetic Properties 5

1.3 History of Spots and Spot Modeling 6

2 METHODS **8**

2.1 Modeling the Spot 8

2.1.1 Intent 8
2.1.2 The Model 8
2.1.3 Edge Conditions 10
2.1.4 Limb-Darkening 13

2.2 Analysis 13
2.3 Data 15

3 STARSPOTS AND SUNSPOTS **17**

3.1 Sunspots 17
3.2 Weather-Related Features 20

4 CONCLUSION **22**

4.1 Results 22

BIBLIOGRAPHY **24**

Appendix

A SPOTMODEL.PY **25**
B ANALYSIS.PY **42**

LIST OF FIGURES

1.1 A table containing some values for the magnetic field strength of various low spectral type stars[9] 6

2.1 A single triangle within the model. Note, this triangle is enlarged for detail, triangles in the actual model are much smaller with approx. 100 to a circle. 11

2.2 A triangle traveling over the edge. The portion over the edge is subtracted from the total area. 11

2.3 An example of what a circle created as a collection of triangles . . . 12

2.4 An example of data collected by the Kepler Space Telescope for the L1 dwarf WISEP J190648.47+401106.8 16

2.5 An example of the poor results that are obtained from a CCD being switched on after a long period of inactivity 16

3.1 Low and High resolution pictures of Jupiter. Note both the prominent spot and stripe features. 21

4.1 Output curve of spot, centered on the star with an area of 0.01 . . 23

4.2 Output curve of a stripe, centered on the star with a width of 0.02 23

ABSTRACT

When studying ultracool stars such as brown dwarfs some of the most informative data comes photometric observations. These observations, though limited in its accuracy due to the limitations of the collector, provide insight into features of the atmosphere that produce fluctuations in the light curve. It is commonly thought that the fluctuations are caused by traditional spots in the star's atmosphere, though analyzing the data can be difficult due to the sometimes sporadic data. This paper seeks to create an open source program which can analyze this photometric data and provide suggestions as to the structure of the spot to produce such a curve. It will also look at the typical features of spots and determine if the features that are seen on these stars are actually consistent with these features or if it some new phenomena such as clouds or other weather related structures.

Chapter 1

INTRODUCTION

The study of Brown Dwarf stars and their features is difficult by its very nature. A combination of both dim objects and still somewhat imperfect equipment creates data that is very noisy and thus hard to analyze. This and the naturally large data sets makes for problems that are very difficult to analyze by hand. This stalls research and interferes with doing actual physics as there is currently no widely available program or method to handle this data. Therefore, a program or model must be developed to facilitate in processing and analyzing the data. Using this model, a rough estimate as to the properties of the features of the star can be determined and work as to the actual properties of the feature and star can begin. The purpose of this paper, therefore, is to present, at least on the most basic level, a model of a spot or some other feature traveling along a star. Hopefully, this can be used as a basis for others to use and make progressively better models and analysis tools so that they may be used by the community to further their research endeavors. It will be created using the high-level language python since that seems to the standard among the astronomy community and its high readability, making editing the code by others much easier.

1.1 The Kepler Mission

Launched in March of 2009, the Kepler space telescope was designed to find Earth-like planets in orbit around their star in what is known as the "Habitable Zone", or distance from the host star where water can exist in liquid form. There are two main ways in which an orbiting planet can be detected, measurements of the radial velocity of the star to find orbital wobble, or measurements in the brightness of the star to find a transiting planet. Of these two methods, Kepler was designed around transits.

Kepler is equipped with 42 charged coupled devices (CCDs) each with a resolution of 2200 X 1024 pixels. These CCDs take in light and convert it into an electrical signal that is read out from each pixel every three seconds. Because each pixel is slightly different in how it reads data, the incoming light is defocused by about 10 arcseconds so that light from any given star can either land on more than one pixel or completely fill one pixel. In total, the array covers 105 square degrees of the sky and, as of its initial launch, is constantly pointed at a patch of sky located in the Cygnus-Lyra region. The sensitivity of the photometer is set such that it can detect the transit of an Earth-like planet across a Sun-like star to 4 sigma precision. Additionally, the filter of the photometer allows in light from the spectral band ranging from 400nm to 850nm, or approximately visible to near infrared light. In order to effectively observe transiting planets, the spacecraft was designed to point at one section of the sky for a very long period of time, approximately three and a half years, the mission's initial lifetime.

Because of the ability to see into the near infrared and the stability of the field of view, the Kepler space telescope has become popular among those who study ultracool stars as well as those searching for exoplanets. Ultracool stars, such as brown dwarfs, emit mainly in the infrared due to their temperature and are difficult to observe because of their low luminosities. These problems are solved thanks to the sensitivity and spectral range of Kepler and these systems can be studied in great depth thanks to the long viewing periods[8].

1.2 An Overview of Brown Dwarfs

Brown Dwarfs are stellar objects. They cannot be classified as stars because they are to low in mass to facilitate the fusion of hydrogen in their core. Since the heat that leads to the temperatures required to start fusion comes from the gravitational contraction of the initial gas cloud, if there is not sufficient material, the collapsing material becomes electron degenerate before adequate temperatures are reached, stopping contraction. This inability to start the process of fusion leads to very dim objects

as the only radiating heat is supplied from the relatively cool contraction energy. Because these objects, classified as M, L, and Y spectral type stars, are so dim it took very powerful telescopes to view them with any certainty, with the first L-Dwarf being confirmed in 1997. Being such a recent edition to the astronomical catalog, much less is known about these types of objects than almost any other stellar object[9].

1.2.1 Distinction between Planets

The extremely low temperatures of Brown Dwarfs lead to some very interesting characteristics not found in other stellar objects. One of these oddities is the presence of molecules in the photosphere of the star such as titanium oxide (TiO), calcium hydride (CaH), iron hydride (FeH), and water[9]. With low mass, temperature, and the presence of heavy elements in the atmosphere, it can be hard to distinguish Brown Dwarfs from gas giants such as Jupiter. The similarities between these two objects are strikingly many. Both primarily generate heat from their cores, both may have complicated systems orbiting them such as moons or planets, and since the size of a Brown Dwarf is set by its electron degeneracy, both can have comparable sizes[1]. Officially, there is no strict dividing line between a Brown Dwarf and a planet, though the International Astronomical Union (IAU) does tentatively place the division at 13 Jupiter masses as below this point fusion cannot occur at all even with the easily fusible deuterium[3]. Of course, if one is observing a Brown Dwarf in infrared light, the distinction between these two is obvious as planets are normally around 100K while the larger dwarfs are on order 1000K, though very low mass dwarfs may approach the temperature of a very large gas giant. There is one other observational way to see a Brown Dwarf, using x-rays. Brown Dwarfs' lack of a source of nuclear energy causes them to have highly convective interiors much like Hayashi track main sequence stars and their very small size give them a very high angular velocity. These two factors produce very kinked magnetic fields near the surface that conduct other highly magnetized material from just below the surface to the surface, creating a sub-surface flare which, in turn, allows electric currents in the star to flow, creating an x-ray flare.

These x-ray flares can be easily detected by telescopes such as Chandra. However, these kinds of flares are, at least to out knowledge, relatively rare, with Chandra only recording one, though this may be an artifact of the sample size as the study of Brown Dwarfs is fairly new and there are not many x-ray telescopes actively looking for such flares[5].

1.2.2 Evolution

Like every other star, Brown Dwarfs begin as a large cloud of loosely associated dust. This dust then collapses in on itself, becoming denser and hotter as potential energy is converted into thermal energy. During this process, the star travels down what is known as the Hayashi track, gradually becoming smaller and, due to decreasing surface area, dimmer. Eventually, most stars reach a point where they have accumulated enough thermal energy that their core temperature is sufficient to start the process of hydrogen fusion. However, since the gravitational energy, and thus subsequent thermal energy, of the collapsing mass goes like $\frac{M^2}{R}$, some of these clouds do not have enough mass, approximately $0.09R_{\odot}$, to reach the critical temperature required for fusion and instead, their cores become electron degenerate. Because of this degeneracy and the lack of fusion, no part of the star transitions from convective, energy transport via bulk movement of warm and cold material, to radiative, energy transport via movement of photons, unlike other main sequence stars which have radiative cores. Additionally, their low mass and luminosity prevent them from reaching the main sequence of stars and relegate them to the spectral types M, L, T, and Y.

Electron degeneracy occurs when the electrons in the plasma become so close together that the force caused by quantum phenomenon, such as the Pauli Exclusion Principle and uncertainty, is greater than the pressure generated by gravity. At this point, we can no longer use an ideal gas equation to model the core of the star.

The degeneracy can be quantified with the parameter α_E according to the equation:

$$\alpha_E = \frac{N_e h^3}{2(2\pi M - EkT)^{2/3}} \sim \frac{electronchemicalenergy}{kT}$$

Where N_e is the number of electron, M the mass of the star, E the Coulomb energy of the star, and h, k, and T are Planck's constant, Boltzmann's constant and temperature respectively. The electron chemical energy is the total potential energy of the electrons in the system, also known as the Fermi level. If this parameter is less than -4 then the plasma can be treated as a perfect gas and if it is over 20 then the plasma is fully degenerate. As the parameter increase, less of the energy from gravitational contraction goes to heating the gas, rather it decreases the separation between electrons. Eventually, if the contraction continues, the star will reach a minimum size of approximately $0.1R_{\odot}$. However, the star can contract to smaller radii as contraction continues, but as it continues to shrink the degeneracy pressure increases, slowing the rate that it contracts. Most Brown Dwarfs bottom out at $0.1R_{\odot}$ because this is the point when degeneracy pressure dominates[9].

1.2.3 Magnetic Properties

In this paper, perhaps the most pertinent feature of Brown Dwarfs is their magnetic properties. As suggested by Reid et. al., the most attractive model for the magnetic structure of a Brown Dwarf is a turbulent dynamo model. A turbulent dynamo creates magnetic fields through the random motion of fluid in the convection zone. These random motions create bundles of field which then escape to the surface of the star[9]. Though some, such as Morin et. al., have observed fully convective stars with stable poloidal fields, they admit that their observations break down below about $0.2R_{\odot}$ where they begin to observe significant non-axisymmetric components and toroidal fields[7].

The Sun uses a similar mechanism to generate its magnetic field, though because it has a radiative core and thus a radiative-convective, it can store and generate fields at this boundary. However, since Brown Dwarfs do not have this boundary, it is generally expected that they have no way to store fields nor have any cyclic phenomena. As Reid et. al. mention, field in this model is generated, moved, and destroyed quickly[9].

Name	Spectral type	B (Gauss)	f(%)
Gl 171.2a	K2e	2,800	60
EQ Vir	K5e	2,500	80
DT Vir	M0.5e	3,000	50
AD Leo	M3e	3,800	73
		4,000	60
Gl 729	M3.5e	2,600	50
EV Lac	M3.5e	3,800	50
YZ CMi	M4.5e	4,200	67

Figure 1.1: A table containing some values for the magnetic field strength of various low spectral type stars[9]

The strength of the magnetic fields in stars can be measured by observing line splitting that occurs due to the Zeeman effect. These effects are easy to observe on solar type stars due to their relatively high luminosity and the availability of the Sun for study. Unfortunately, because Brown Dwarfs are both new and dim, there is not much information regarding their magnetic fields. However, as can be seen in Figure 1.1, there is a general trend toward increasing magnetic fields with decreasing spectral type[9].

1.3 History of Spots and Spot Modeling

The problem of modeling spots on cool stars is not a new one. At least as far back as the 1980s, people have been trying to model the movement of spots across an unresolved star and the resulting light curve. Surprisingly, even then, authors such as Dorren[4], though observing much hotter F-type stars, struggled with the issues of spot size, persistence, and even latitudinal location of spot. Dorren also notes that though Sunspots generally have non-uniform brightness and temperature across their

surface, it is much simpler to simply model uniform spots without umbras as the extra parameters reveal little about the actual structure of the spot itself.

In Dorren's time, computers were not as powerful as they are today of course and as such, this is reflected in the way Dorren creates his model. He uses a straightforward double integral to integrate over the area of an arbitrary spot, whose shape is chosen explicitly to simplify the integral, on a star's surface and subtracts it from the total area of the star (a unit sphere). Unlike in the model presented here, this method is far more analytical and can be solved by hand. Today though, we can use computation to better handle these types of problems through numerical methods and iterations. By doing this we are not constrained as Dorren was to only choose easy shapes, using the methods presented here, we can make almost any shape imaginable[4].

Today, the modeling of spots has become far more sophisticated, being used to measure the properties of the star such as its inclination angle, radius, and differential rotation rate. All of this is because we are able to more accurately model the spots using computational methods. Spots have also taken on a slightly more important role in Astronomy as the search for extraterrestrial life intensifies. Because spots are very similar to planets as they move across the surface of a star, it becomes very important to find a way to distinguish the two so that false positives can be avoided. In Dorren's time, spots were mainly studied in an effort to determine the properties of the spots such as their temperature and how similar these spots behaved to the ones on the sun. In his paper, Kipping prefers to use an analytical model because though numerical models can produce many different shapes, analytical models are much faster for the limited number of shapes that they use since they do not need to perform iteratively[6]. While this may be true, in this paper we would like to analyze, somewhat, the effects that other shapes have so we will use a numerical method. Additionally, a numerical method is easier to approach conceptually than an analytical model.

Chapter 2

METHODS

2.1 Modeling the Spot

2.1.1 Intent

Before we discuss how we actually go about modeling the spot, let us discuss what we want to get out of the model first and, conceptually, how we will set up the model. We know that we want to have some feature rotate around a sphere, but what shape will it take and how will we deal with it passing over the edge of the sphere? To start, let us consider the simplest case with the simplest shape, a single triangle. A triangle contains three lines and three vertices so we must decide if we want to represent this triangle in terms of its edges or vertices. If we want this triangle to pass easily over the edge of the sphere it seems that using the vertices would be the easiest implementation as we can simply make a new triangle that is not over edge by adding additional points. Therefore, if a triangle goes over the edge we can still find the total area covered by the triangle by adding extra points at the boundary to create an appropriate triangle.

2.1.2 The Model

The first thing that this code must do is effectively model some feature (the spot) on a sphere, have it rotate around the sphere, (into and out of view of the observer) and from this, produce a light curve. To represent the sphere on which we will place the spot we will simply use a circle. Some similar models will use a square, randomly sample points within that square and define the inside and outside of a circle by their distance from the center. This is too complicated and unnecessary for this model. Instead, I will model the sphere through an essentially three-dimensional coordinate

system. If we place polygons on the sphere, these polygons are defined by the points of their vertices. These points will be represented by three points, x, y, and z. The y and z coordinates will define the location of the point within the plane and the x coordinate representing how far into or out of the plane the point is. Using the x coordinate, we can determine if the point is on the front of the sphere (in view) or on the back of the sphere (out of view).

We run into another problem after we have decided this though. Since we would like to accurately represent a shape moving upon a sphere we must account for the apparent warping of the shape as it approaches the edges. To do this instead of starting with the Cartesian coordinates we start with spherical coordinates and then use spherical trigonometry to transform them into Cartesian coordinates. Therefore, when a user inputs a feature, they will specify the polar angle ("phi" in the code) and the azimuthal angle ("lamda" in the code because lambda is a reserved name in Python) in degrees which the program will convert into polar. However, we can make an assumption at this stage. If we assume that whatever feature we decide to make using these points will be very small compared to the overall sphere on which it sits, we can use simple Euclidean transformations instead of the messy spherical ones to obtain the Cartesian coordinates from the spherical. Therefore, we have:

$$x = \sin(\phi) * \cos(\lambda), y = \sin(\phi) * \sin(\lambda), z = \cos(\lambda)$$

Using these coordinates along with simple geometry allows us to get the area of the shape we have created, giving us all the information we need to know about it.

The reason we only need to know the area of the shape is because we will assume for this model that the spot is completely dark, stopping all light that would have been produced by that area of the star. The reason for this is to simply keep the computation time down. A spot that is slightly brighter will produce essentially the same light curve as a slightly smaller, completely dark spot. Using this, we have a fairly straightforward way of computing the light curve; we simply subtract from the

total area of the sphere in view (the area of a circle) the total area covered by some kind of spot.

Finally, we must deal with exactly what shape we want to make with the program. Since we are modeling spots on a star, a circle seems like the best option. However, circles are not made of points which our model currently requires. Therefore, we must look for other ways to construct a circle using other shapes. The best way to do this is by using triangles, from which a circle can easily be constructed by creating a small wedge and rotating it around the center, copying it each time.

Using triangles is good fit for two reasons: 1) We can make them very small and adhere to our previously made assumption; and 2) We can make almost any other polygon using triangles. This second reason will become helpful later when we try to model stripes as well as spots.

2.1.3 Edge Conditions

Perhaps the most important case to deal with during the simulation process is what happens when a triangle reaches the edge of the circle and travels over it onto the other side. In order to determine how the triangle is traveling over the edge we look at the three points that make up the triangle, tracking whether each point is on the front of the circle or on the back. We can do this by tracking the sign of the x-value of a point. If the x-value is positive or zero it is on the front of the circle and if it is negative it is on the back of the circle. To account for the loss of area as the spot moves over the side we must make another triangle equal either to the area of the triangle over the edge or the area of the triangle still on the front of the circle. To do this we must create additional "false" points on the edge of the circle.

To correctly position these false points let us assume that the triangle of points extends past the two-dimensional projection of the sphere. Where this triangle intersects the circle is where the false point will be placed.

To find the slope of the line connecting two points we use simple rise over run.

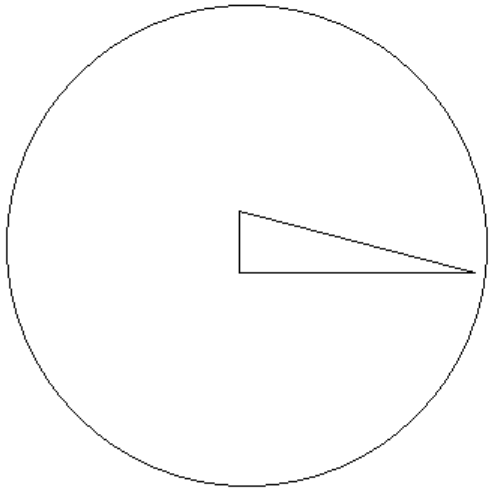

Figure 2.1: A single triangle within the model. Note, this triangle is enlarged for detail, triangles in the actual model are much smaller with approx. 100 to a circle.

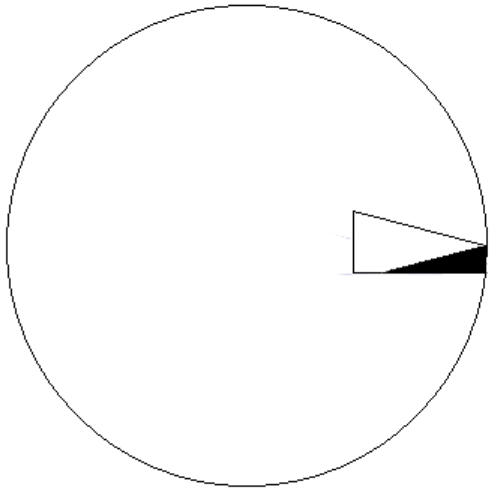

Figure 2.2: A triangle traveling over the edge. The portion over the edge is subtracted from the total area.

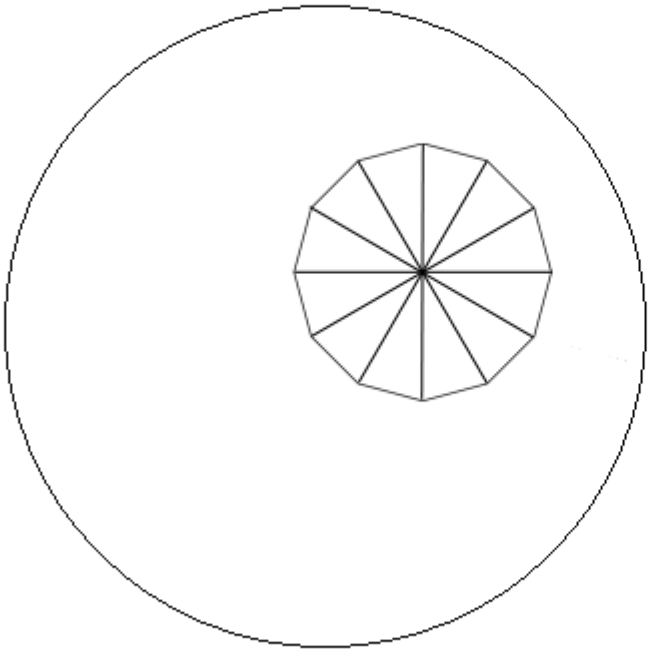

Figure 2.3: An example of what a circle created as a collection of triangles

$$slope = \frac{y_2^{'} - y_1}{z_2 - z_1}$$

2.1.4 Limb-Darkening

One of the more complex features of a star is its limb-darkening. It is very difficult to determine exactly how limb-darkening will affect the light profile of a star or by what function it will be determined. Usually though, limb-darkening is described by a linear function, something like $1 - \mu * x$ where x is the radial distance from the center of the star. This effect is of course different at every point inside an extended object such as our triangles, however, we will use the small triangle approximation again to simplify the process slightly. Instead of calculating the limb-darkening at each point we instead calculate the limb-darkening at the average center of the triangle. Since the limb-darkening is linear the total decrease of the brightness of the spot would just be the average at either end since the triangle is very small and line-like. We will now simply modify the area of the spot by the limb-darkening parameter. The constant μ can be input by the user.

2.2 Analysis

Originally, this program was to be a completely self-contained analysis tool, unfortunately though I was unable to get the built-in functions in the scipy package of python to properly analyze data. Instead, this program will format given data in such a way that the user can use an outside data analysis tool such as Origin to produce a least-squares curve fit.

The first step of the analysis process is for the user to input a data set of photometric counts and the times the counts were taken. The input file should be a ".txt" file with two columns, the left being the times the counts were taken and the right being the photometric counts normalized to 1. As an additional parameter, the user will also input the step time in days between one unit of time in the input file. The program then runs through each row and finds the average time step over the

data set and then runs through the data again, dividing the data into separate files based on the average time step. If there is a time step within the data larger than the average time step the program will break the data and put the preceding data into a new file. This step is to ensure that the user has a data set free from interruptions in data collection and also identifies areas around which there may be low data fidelity as a CCD comes back online. The user may then take these data files and run a least squares analysis using whatever tool they wish.

Once the user has run their own analysis they will be prompted to input the results into the program, specifically, the amplitude, period, and phase shift. There are also additional parameters that may be input such as inclination angle of the star.

The program will then attempt to model the given data using the following steps:

1. Determine how much area is covered by spots.

   Assuming the input data was normalized it will use the amplitude to find the percent area covered by $percentarea = 1 - Amplitude$. The radius of the circle will be then found by $r = (percentarea)$ since the circle in the model is a unit circle. This program will not try to run an analysis with stripes as the total area covered by stripes is highly dependent on the latitude they are at.

2. Create a reference file.

   The program will take the fit parameters and run through them one period to create a reference file of points for use later.

3. Iterate through one cycle.

   The program will center the spot at a point such that the entire spot can be out of view. During each iteration, it will subtract the output file from the reference file and take the average. If during an iteration the average is lower it will replace the current one. The iterations will continue at some interval until it reaches an angle ϕ of 90. Once all the iterations are complete it will output the parameters of the best run.

It should be noted that while the analysis tool is important, the main focus of this thesis is to develop the spot model. Therefore, some features of the analysis program may not work properly or be fully implemented.

2.3 Data

The data that is most commonly used for these problems, that is to say observing features on Brown Dwarfs, is photometric data. Photometric data is a set of information containing the number times a photon, of any energy allowed by the filters, strikes a pixel on the camera. From this we can determine how much light the observed star is emitting at any given time since the number of photons emitted is directly proportional to the star's luminosity and flux at the observation location.

In most every case, the camera collecting the data is a Charge-Coupled Device or CCD. CCD cameras rely on the incoming photons exciting the surface of the device and generating an electric current that registers with the software as one count. In the data that we used as a template to design the model, the counts were normalized to 1 as can be seen in Figure 2.4. However, CCDs are not perfect. Each individual pixel has its own sensitivity on different locations of the pixel itself, leading to misleading data if the problem is not addressed. This is remedied by slightly defocussing the camera lens so that an individual point of light does not hit one spot in a pixel, but rather the image is smeared over the pixel and it neighbors.

Additionally, since Brown Dwarfs are very cool, these measurements are done in the infrared and therefore the measurements are susceptible to interference from thermal background noise. This combined with the already low signal-to-noise, because of the low luminosity of these stars, creates very messy data that is hard to evaluate by hand or eye. As can be seen in figure 2.5, when a CCD becomes active again after a long period of inactivity, the results that it obtains are dubious at best. Due to the nature of the CCD chip, it must first establish an equilibrium with the environment before it can provide useful data, much like how its is difficult to see in a bright area after you have been exposed to the dark for a period of time.

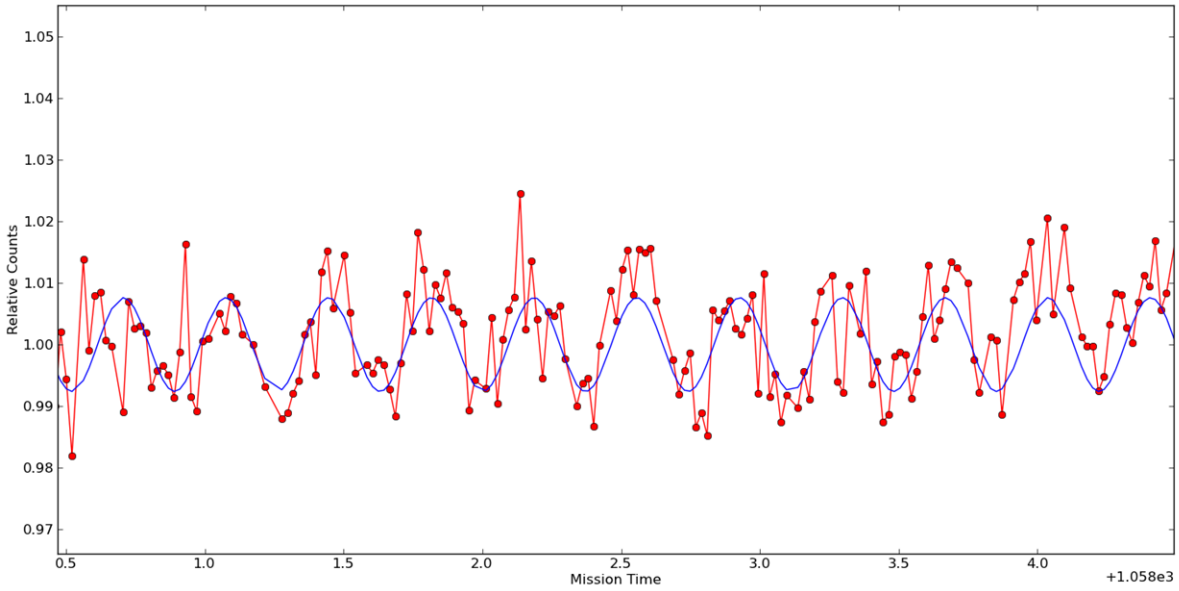

Figure 2.4: An example of data collected by the Kepler Space Telescope for the L1 dwarf WISEP J190648.47+401106.8

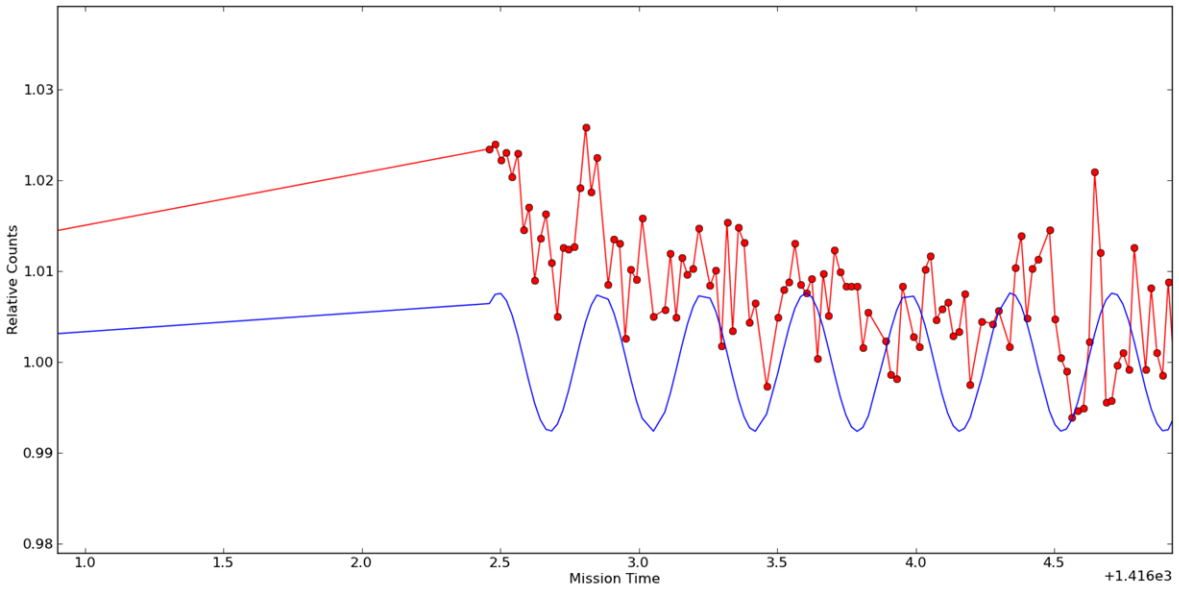

Figure 2.5: An example of the poor results that are obtained from a CCD being switched on after a long period of inactivity

Chapter 3

STARSPOTS AND SUNSPOTS

Spots are an interesting phenomena because they are easily viewable on the Sun, but very difficult to observe on distant stars. Because we cannot resolve the disk on distant stars, its hard to determine if a decrease in observed luminosity is due to a spot or some other phenomenological process. This puts us in an interesting position since we can only determine the properties of a potential spot from basic observables such as luminosity or spectroscopy. This leaves our understanding fairly limited and based mainly on what we know of sunspots.

3.1 Sunspots

Sunspots are the most well understood spot phenomenon since they are directly observable. Using our understanding of these we can devise a template to study other spots on distant stars with our limited information.

Sunspots are grouped into several classifications, most of which involve bipolar spots, or spots which have a twin spot on the other side of the star. One classification, H, is known as a theoretician's spot since it is the easiest to model, not containing a multitude of smaller spots or having a large umbra/penumbra distribution. These spots have been shown empirically to decay at a slow, constant rate of:

$$\dot{A} \approx -1.5 \times 10^8 m^2/s$$

though there is some evidence to suggest a parabolic decay rather than a linear one, a linear model still provides usable results.

When we consider modeling sunspots we must start by idealizing the umbra of the spot. Because these umbra are very inhomogeneous it is much easier to consider a

mean spot rather than the actual spot. Additionally, it has been theorized and observed that sunspots represent a physical depression in the sun due to their lower opacity. This means that when the spot reaches the edge of the sun the disc-side penumbra appears to be narrower than the limb-side penumbra, an important consequence that must be remembered when modeling the transit of a spot, especially later when we deal with spots on Brown Dwarfs. There are also considerations that must be made for the magnetohydrostatics of the problem since there are some dynamical elements. These elements are the decay of the spot and the moat, an area encircling the spot where the surface of the sun flows away from the spot to remove heat from underneath it. We simplify this though by simply observing that both of these processes are very slow and do not greatly affect the static problem. Having the static problem, we use the magnetohydrostatic equilibrium equation:

$$-\nabla P + \rho \mathbf{g} + \mathbf{j} \times \mathbf{B} = 0$$

to determine the magnetic properties of the spot and its underlying structure where P is the pressure of the gas, ρ is the density of the gas, g is the gravitational acceleration, $\mathbf{j}$ is the current density, and $\mathbf{B}$ is the magnetic field. The most basic models of this approach effectively determine the strength of the field and the spots depression by using temperature as a free parameter to find the eigenvalue corresponding to mixing length over scale height. However, these models do not effectively explain the sharp transitions between umbra, penumbra, and surrounding area. This can be solved by introducing current sheets into the model. These current sheets separate the magnetic fields, by running tangent to the magnetic field, of the umbra and penumbra and the whole system from the quiet outside. These two sheets are known as the magnetopause, which surrounds the entire system, and the peripatopause, which surrounds the umbra.

Earlier, we had broached the subject of spot decay as an essentially constant process. Now we will look at spot decay in terms of the magnetic field of the spot. To start, we assume that the spot and its associated field decay diffusively according to the turbulent diffusivity η_t caused by the random motion of the gas in deeper levels of

the sun. By using the magnetic diffusion equation $\dot{B} = \frac{1}{s}\frac{\partial}{\partial s}(s\eta_t \frac{\partial B}{\partial s})$ where s is the spot radius, we can solve for B to obtain:

$$B = \frac{\Phi}{4\pi\eta_t t e^{\frac{s^2}{4\eta_t t}}}$$

Where Φ is the total flux. This equation states that for small t, B is very large, though we know that the magnetic field is limited by the magnetohydrostatic equilibrium. Therefore, a correction term is introduced, Φ_*, which represents the amount of flux that is actually observable through the spot. Physically, Φ_* represents the amount of the magnetic field greater than some critical value that allows it to actually affect convective energy transport, the source of the visibility of the spot. Therefore, the expression for this new visible flux is:

$$\Phi_* = \Phi - 4\pi\eta_t B_c t$$

and from this we can determine the approximate decay of the area of the spot as:

$$\dot{A} \cong -10\pi \frac{B_c}{B_m}$$

with B_m the portion of the field greater than the critical field. As we can see, this decay is related to the overall size of the spot, with larger spots having longer lifetimes.

The total number of spots on the surface of the sun vary over an approximately eleven year cycle. At the height of the cycle, the Sun may have as many 200 spots while at the depth there are almost none. Additionally, the spots also seem to migrate uniformly from latitudes of 40 degrees down to the equator where they disappear. When the cycle restarts, the spots reappear around 40 degree latitude. It is difficult to say how the number of spots equate to total area of the Sun covered by said spots. Spots come in many classifications ranging from large homogeneous spots to collections of small, irregular spots. Additionally, these spots, while generally black in terms of brightness, can have complex umbras and penumbras which do not lend themselves to efficient models. However, the largest of these spots are usually very black and have minimal penumbras that can be effectively ignored with respect to the overall area of the spot [10].

3.2 Weather-Related Features

As we will discuss later, it may be possible to explain the brightness curves of Brown Dwarfs by weather-related effects. To examine these we will look to Jupiter as it is well studied and has both large spots and stripes in its atmosphere. Firstly, the Great Red Spot on Jupiter is very stable, having lasted for over 100 years as can be seen in Figure 3.1. As noted by Beebe, maintaining a large spot like the one on Jupiter requires a lot of internal energy, a quantity that a star, albeit a very weak star, has in spades. There have also been observations over the course of many decades that show an occasionally fast but usually slow longitudinal drift. These drifts, in their slow periods, can be as slow as 20 degrees over 10 years.

The brightness contribution of the spot changes over time. At times, ice crystals are funneled into the main body of the spot, making it largely homogeneous with the surrounding cloud layer. At other times, ice crystals are evacuated from the spot, making the spot well defined and dark with respect to the surrounding cloud layer. Unfortunately, because Jupiter is so close and so easily resolved there is no data directly measuring the brightness of Jupiter, especially since Jupiter must rely on the Sun for its luminosity.

In 1939, new features were observed on the surface of Jupiter, notably ellipsoid flat clouds. These clouds are noteworthy because they seems to have similar properties to the Great Red Spot, such as color, and are very long-lived and stable in terms of size. They have lasted from 1939 when they were discovered until at least 1989 when they were still being observed [2].

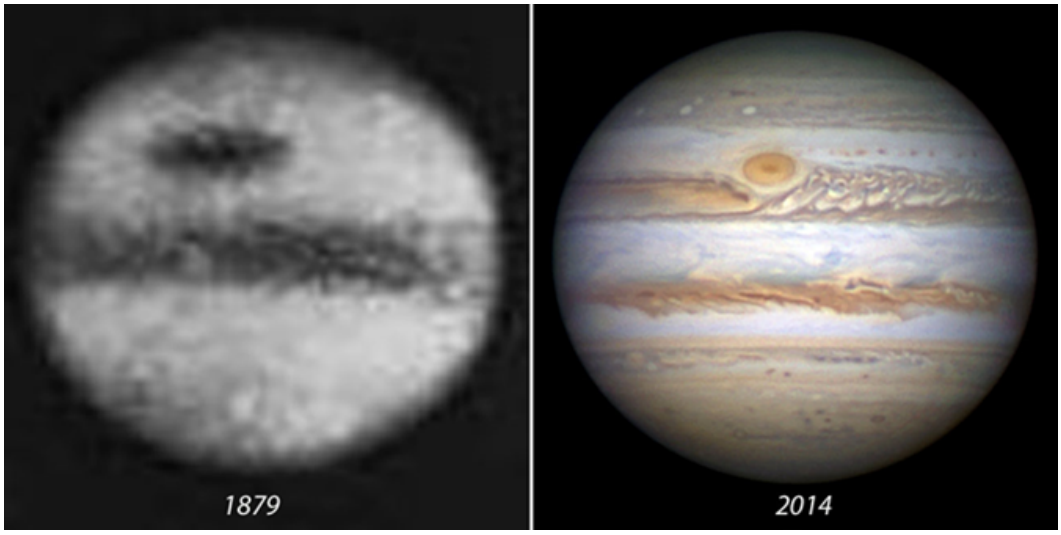

Figure 3.1: Low and High resolution pictures of Jupiter. Note both the prominent spot and stripe features.

Chapter 4

CONCLUSION

Though the model and analysis tools do not work perfectly, they provide a solid basis to start processing astronomical data and a solid basis from which others can build and improve. Overall, things that need to be improved are

1. Fully encompassed data analysis package that can run a least squares analysis without additional user input.
2. Faster model or more mathematically correct model that relies less on assumptions
3. A model that accurately includes an inclination angle correction
4. More efficient model runs

4.1 Results

From what we know about both star spots and the data we collect, see Figure 2.4, we can make the assumption that the features on Brown Dwarfs are not spots in the typical sense. The features that are observed are far to long lived, especially for the large magnetic fields that are present in such stars. Running the model as both a spot and stripe of similar areas produce similar curves. As can be seen in figures 4.1 and 4.2 the two different features produce very similar curves. Because the curves for spots and stripes are so similar, this suggests that the feature could be a stripe, which is to say that the feature decreasing the luminosity is a striped cloud, similar to what we see on Jupiter (see figure 3.1). Conceptually, this does make sense as ultracool stars like Brown Dwarfs are cool enough to contain heavy molecules just as dust and water vapor. This may lead to the condensation of clouds in the cool outer photosphere creating a blanketing layer that reduces the overall luminosity of the star. This fact has been speculated by Gizis, et. al. as a way to accurately explain the incredibly long lived features found on Brown Dwarfs.

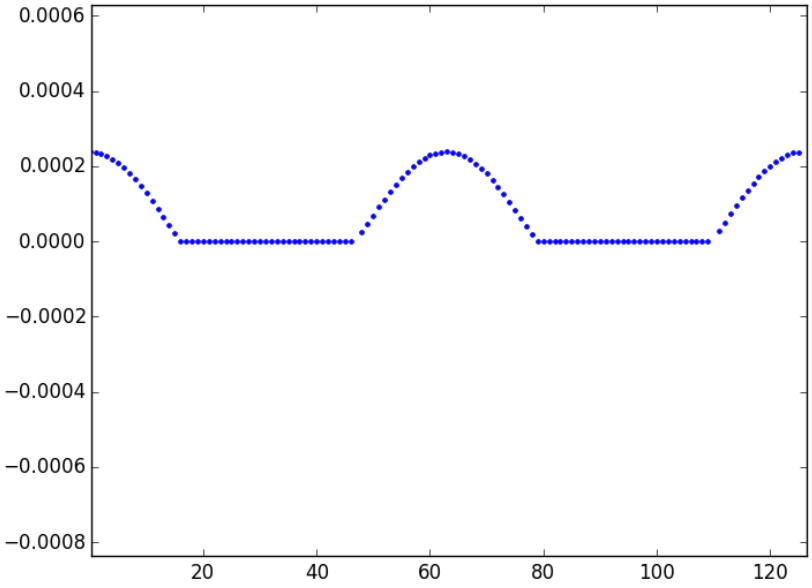

Figure 4.1: Output curve of spot, centered on the star with an area of 0.01

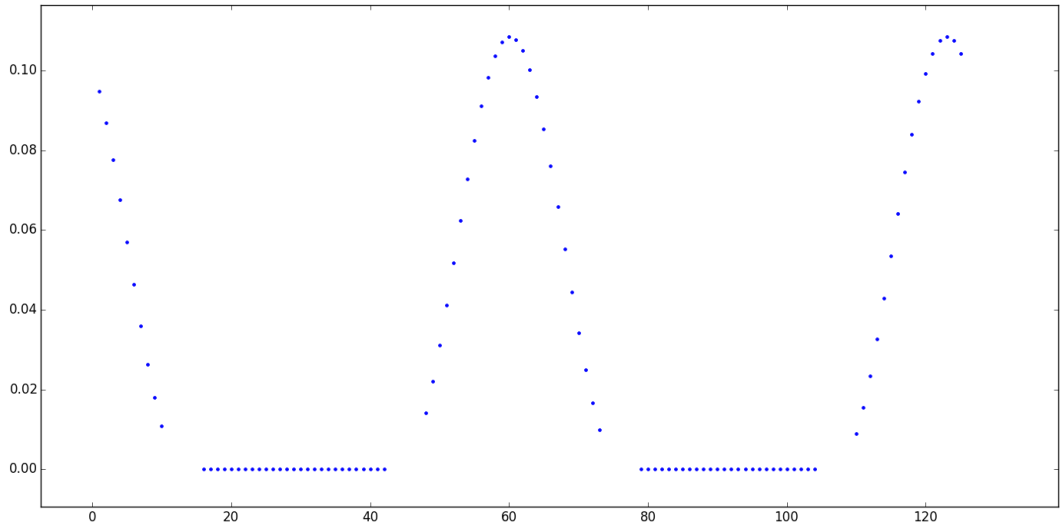

Figure 4.2: Output curve of a stripe, centered on the star with a width of 0.02

BIBLIOGRAPHY

[1] Gibor Basri and Michael E. Brown. PLANETESIMALS TO BROWN DWARFS: What is a Planet? *Annual Review of Earth and Planetary Sciences*, 34.1:193–216, 2006.

[2] Reta Beebe. *Jupiter: The Giant Planet Second Edition*. Smithsonian Institution Press, 1997.

[3] Lars L. Christensen, Helen Sim, Raquel Y. Shida, Nadja Wolf, and Lars H. Nielsen. The Public Comunication at the IAU GA 2006. *Proceedings of the International Astronomical Union*, 2.S240, 2006.

[4] J. D. Dorren. A New Formulation of the Starspot Model, and the Consequences of Starspot Structure. *Astrophysical Journal*, 320:756–767, September 1987.

[5] Harvard. Chandra :: Field Guide to X-ray Sources :: Brown Dwarfs, June 2012.

[6] David M. Kipping. An Analytic Model for Rotational Modulations in the Photometry of Spotted Stars. *Monthly Notices of the Royal Astronomical Society*, pages 1–30, November 2012.

[7] J. Morin, J. F. Donati, P. Petit, L. Albert, M. Auriere, R. Cabanac, C. Catala, X. Delfosse, B. Dintrans, R. Fares, T. Forveille, T. Gastine, M. Jardine, R. Konstantinova-Antova, J. Lanoux, F. Lingires, A. Morgenthaler, F. Paletou, J. C. Ramirez Velez, S. K. Solanki, S. Thado, and V. Van Grootel. Exploring the magnetic topologies of cool stars. *The Physics of Sun and Star Spots Proceedings IAU Symposium*, 2010.

[8] NASA. Kepler and K2, April 2015.

[9] I. Neil Reid and Suzanne L. Hawley. *New Light on Dark Stars: Red Dwarfs, Low-mass Stars, Brown Dwarfs*. Springer, 2000.

[10] Michael Stix. *The Sun: An Introduction Second Edition*. Springer, 2002.

Appendix A

SPOTMODEL.PY

```
#Spot Model Program
#Kyle Dettman 2015

import math
from math import sin
from math import cos
from math import tan
from math import pi
import numpy
import matplotlib.pyplot as plt
#Making Spot Object
class Spot:
    """A spot is made up of three points definded by their inclination a
    phi and polar angle lamda"""


    spotList = []

    def __init__(self, phi1, lamda1, phi2, lamda2, phi3, lamda3, incl=No
        #do not use the inclination parameter, it is not properly implem
        if (incl is not None):
            self.phi1 = self.phi1+incl
```

```
            self.phi2 = self.phi2+incl
            self.phi3 = self.phi3+incl
        self.phi1 = (phi1*pi)/180
        self.lamda1 = (lamda1*pi)/180
        self.phi2 = (phi2*pi)/180
        self.lamda2 = (lamda2*pi)/180
        self.phi3 = (phi3*pi)/180
        self.lamda3 = (lamda3*pi)/180
        #The following attributes keep track of how many points have pas
        self.onePoint = False
        self.twoPoint = False
        #The following attributes keep track of which point is OTE
        self.OTE1 = False
        self.OTE2 = False
        self.OTE3 = False
        #To deal with OTE points we need false points
        self.falsePoints = False
        self.fp1 = None
        self.fp2 = None
        #By convention I would like to make false point 1 (fp1) to be th
        #Interior Angles
        self.angleA = self.calcCenAngle(phi1, phi2, lamda1-lamda2)
        self.angleB = self.calcCenAngle(phi2, phi3, lamda2-lamda3)
        self.angleC = self.calcCenAngle(phi3, phi1, lamda3-lamda1)
        #Surface Angles
        self.a = (math.cos(self.angleA)-math.cos(self.angleB)*math.cos(s
        self.b = (math.cos(self.angleB)-math.cos(self.angleC)*math.cos(s
        self.c = (math.cos(self.angleC)-math.cos(self.angleA)*math.cos(s
```

```
#Additionally I could convert to cartessian coordinates
#x is defined to be out of the screen
#y is the axis going left to right
#z is the axis going up and down
self.x1 = sin(phi1)*cos(lamda1)
self.y1 = sin(phi1)*sin(lamda1)
self.z1 = cos(phi1)
self.x2 = sin(phi2)*cos(lamda2)
self.y2 = sin(phi2)*sin(lamda2)
self.z2 = cos(phi2)
self.x3 = sin(phi3)*cos(lamda3)
self.y3 = sin(phi3)*sin(lamda3)
self.z3 = cos(phi3)
#To account for limb-darkening we need to find the average x of
self.averageX = (self.x1+self.x2+self.x3)/3
#When x is less than zero discount the point
if (self.x1 >= 0 and self.x2 >= 0 and self.x3 >= 0): #Messy appr
    self.z1 = cos(self.phi1)
    self.y1 = sin(self.phi1)*sin(self.lamda1)
    self.z2 = cos(self.phi2)
    self.y2 = sin(self.phi2)*sin(self.lamda2)
    self.z3 = cos(self.phi3)
    self.y3 = sin(self.phi3)*sin(self.lamda3)
    self.onePoint = False
    self.twoPoint = False
    self.OTE1 = False
    self.OTE2 = False
    self.OTE3 = False
elif (self.x1 >= 0 and self.x2 >= 0 and self.x3 < 0):
```

```
            self.z1 = cos(self.phi1)
            self.y1 = sin(self.phi1)*sin(self.lamda1)
            self.z2 = cos(self.phi2)
            self.y2 = sin(self.phi2)*sin(self.lamda2)
            self.z3 = 0
            self.y3 = 0
            self.onePoint = True
            self.twoPoint = False
            self.OTE1 = False
            self.OTE2 = False
            self.OTE3 = True
        elif (self.x1 >= 0 and self.x2 < 0 and self.x3 >= 0):
            self.z1 = cos(self.phi1)
            self.y1 = sin(self.phi1)*sin(self.lamda1)
            self.z3 = cos(self.phi3)
            self.y3 = sin(self.phi3)*sin(self.lamda3)
            self.z2 = 0
            self.y2 = 0
            self.onePoint = True
            self.twoPoint = False
            self.OTE1 = False
            self.OTE2 = True
            self.OTE3 = False
        elif (self.x1 < 0 and self.x2 >= 0 and self.x3 >= 0):
            self.z3 = cos(self.phi3)
            self.y3 = sin(self.phi3)*sin(self.lamda3)
            self.z2 = cos(self.phi2)
            self.y2 = sin(self.phi2)*sin(self.lamda2)
            self.z1 = 0
```

```
            self.y1 = 0
            self.onePoint = True
            self.twoPoint = False
            self.OTE1 = True
            self.OTE2 = False
            self.OTE3 = False
        elif (self.x1 >= 0 and self.x2 < 0 and self.x3 < 0):
            self.z1 = cos(self.phi1)
            self.y1 = sin(self.phi1)*sin(self.lamda1)
            self.z2 = 0
            self.y2 = 0
            self.z3 = 0
            self.y3 = 0
            self.onePoint = True
            self.twoPoint = True
            self.OTE1 = False
            self.OTE2 = True
            self.OTE3 = True
        elif (self.x1 < 0 and self.x2 >= 0 and self.x3 < 0):
            self.z2 = cos(self.phi2)
            self.y2 = sin(self.phi2)*sin(self.lamda2)
            self.z1 = 0
            self.y1 = 0
            self.z3 = 0
            self.y3 = 0
            self.onePoint = True
            self.twoPoint = True
            self.OTE1 = True
            self.OTE2 = False
```

```
            self.OTE3 = True
        elif (self.x1 < 0 and self.x2 < 0 and self.x3 >= 0):
            self.z3 = cos(self.phi3)
            self.y3 = sin(self.phi3)*sin(self.lamda3)
            self.z1 = 0
            self.y1 = 0
            self.z2 = 0
            self.y2 = 0
            self.onePoint = True
            self.twoPoint = True
            self.OTE1 = True
            self.OTE2 = True
            self.OTE3 = False
        else:
            self.z1 = 0
            self.y1 = 0
            self.z2 = 0
            self.y2 = 0
            self.z3 = 0
            self.y3 = 0
            self.onePoint = True
            self.twoPoint = True
            self.OTE1 = True
            self.OTE2 = True
            self.OTE3 = True


        self.spotList.append(self)
```

```
#Calculates the central angle between two angles
def calcCenAngle(self, phi1, phi2, delLamda):
    return numpy.arccos((math.sin(phi1)*math.sin(phi2))+(math.cos(ph
    #TODO: Add if statement to check for small angles

#Currently not implemented as it is not cartessian
def calcArea(self):
    #Assuming unit sphere
    return self.a + self.b + self.c - math.pi

def calcAreaCart(self):
    if (not self.onePoint):#No points OTE
        #need to multiply by a limb-darkening coefficient
        #return (1-(self.averageX+.01))*(0.5*(self.distance(self.z1,
        return (0.5*(self.distance(self.z1, self.y1, self.z2, self.y2
    elif (not self.twoPoint):#One point OTE
        self.falsePoints = True
        #With one point OTE we need to make two false points at the
        #To do this we need to know the slope of the line between th
        #area = (1-(self.averageX+.01))*(0.5*(self.distance(self.z1,
        area = (0.5*(self.distance(self.z1, self.y1, self.z2, self.y2
        if (self.OTE1):#Point 1 OTE
            yOTE1 = 2-self.y2-self.y1
            yOTE2 = 2-self.y3-self.y1
            diff = 1 - self.y1
            if (yOTE1-self.y2 != 0):#Check to eliminate divide by ze
                slope1 = (self.z1-self.z2)/(yOTE1-self.y2)
                self.fp1 = self.z1 + slope1*diff
            else:
```

```
            self.fp1 = self.z1 #TODO: check that this is correct
        if (yOTE2-self.y3 != 0):
            slope2 = (self.z1-self.z3)/(yOTE2-self.y3)
            self.fp2 = self.z2 + slope2*diff
        else:
            self.fp2 = self.z2
        #Need to calculate area of OTE triangle with new points
        #negArea = (1-(self.averageX+.01))*(0.5*(self.distance(s
        negArea = (0.5*(self.distance(self.z1, self.y1, self.fp1
        return area-negArea
    elif (self.OTE2):#Point 2 OTE
        yOTE1 = 2-self.y1-self.y2
        yOTE2 = 2-self.y3-self.y2
        diff = 1-self.y2
        if (yOTE1-self.y1 != 0):
            slope1 = (self.z2-self.z1)/(yOTE1-self.y1)
            self.fp1 = self.z2 + slope1*diff
        else:
            self.fp1 = self.z2
        if (yOTE2-self.y3 != 0):
            slope2 = (self.z2-self.z3)/(yOTE2-self.y3)
            self.fp2 = self.z3 + slope2*diff
        else:
            self.fp2 = self.z3
        #negArea = (1-(self.averageX+.01))*(0.5*(self.distance(s
        negArea = (0.5*(self.distance(self.z2, self.y2, self.fp1
        return area-negArea
    else:#Point 3 OTE
        yOTE1 = 2-self.y1-self.y3
```

```
            yOTE2 = 2-self.y2-self.y3
            diff = 1-self.y3
            if (yOTE1-self.y1 != 0):
                slope1 = (self.z3-self.z1)/(yOTE1-self.y1)
                self.fp1 = self.z3 + slope1*diff
            else:
                self.fp1 = self.z3
            if (yOTE2-self.y2 != 0):
                slope2 = (self.z3-self.z2)/(yOTE2-self.y2)
                self.fp2 = self.z1 + slope2*diff
            else:
                self.fp2 = self.z1
            #negArea = (1-(self.averageX+.01))*(0.5*(self.distance(s
            negArea = (0.5*(self.distance(self.z3, self.y3, self.fp1
            return area-negArea
        elif (self.OTE1 and self.OTE2 and self.OTE3):#Three points OTE
            return 0
        else:#Two points OTE
            if (self.OTE1 and self.OTE2):
                yOTE1 = 2-self.y3-self.y1
                yOTE2 = 2-self.y3-self.y2
                diff1 = 1-self.y1
                diff2 = 1-self.y2
                if(yOTE1-self.y3 != 0):
                    slope1 = (self.z1-self.z3)/(yOTE1-self.y3)
                    self.fp1 = self.z1 + slope1*diff1
                else:
                    self.fp1 = self.z1
                if(yOTE2-self.y3 != 0):
```

```
            slope2 = (self.z2-self.z3)/(yOTE2-self.y3)
            self.fp2 = self.z2 + slope2*diff2
        else:
            self.fp2 = self.z2
        #area = (1-(self.averageX+.01))*(0.5*(self.distance(self
        area = (0.5*(self.distance(self.z1, self.y1, self.fp1, 1)
        return area
    elif (self.OTE1 and self.OTE3):
        yOTE1 = 2-self.y2-self.y1
        yOTE2 = 2-self.y2-self.y3
        diff1 = 1-self.y1
        diff2 = 1-self.y3
        if (yOTE1-self.y2 != 0):
            slope1 = (self.z1-self.z2)/(yOTE1-self.y2)
            self.fp1 = self.z1 + slope1*diff1
        else:
            self.fp1 = self.z1
        if (yOTE2-self.y2 != 0):
            slope2 = (self.z3-self.z2)/(yOTE2-self.y2)
            self.fp2 = self.z3 + slope2*diff2
        else:
            self.fp2 = self.z3
        #area = (1-(self.averageX+.01))*(0.5*(self.distance(self
        area = (0.5*(self.distance(self.z1, self.y1, self.fp1, 1)
        return area
    else:#TODO check the areas to make sure they do the thing rig
        yOTE1 = 2-self.y1-self.y3
        yOTE2 = 2-self.y1-self.y2
        diff1 = 1-self.y3
```

```
                diff2 = 1-self.y2
                if (yOTE1-self.y1 != 0):
                    slope1 = (self.z3-self.z1)/(yOTE1-self.y1)
                    self.fp1 = self.z3 + slope1*diff1
                else:
                    self.fp1 = self.z3
                if (yOTE2-self.y1 != 0):
                    slope2 = (self.z2-self.z1)/(yOTE2-self.y1)
                    self.fp2 = self.z2 + slope2*diff2
                else:
                    self.fp2 = self.z2
                #area = (1-(self.averageX+.01))*(0.5*(self.distance(self
                area = (0.5*(self.distance(self.z1, self.y1, self.fp1, 1)
                return area

    def distance(self, x1, y1, x2, y2):
        if (x1 == x2 and y1 == y2):
            return 0
        #return 10**math.sqrt((x1-x2)**2 + (y1-y2)**2)
        return math.sqrt((x1-x2)**2 + (y1-y2)**2)

    #Recalculates Cart coords
    def redo(self):
        self.x1 = sin(self.phi1)*cos(self.lamda1)
        self.x2 = sin(self.phi2)*cos(self.lamda2)
        self.x3 = sin(self.phi3)*cos(self.lamda3)
        #To account for limb-darkening we need to find the average x of
        self.averageX = (self.x1+self.x2+self.x3)/3
        if (self.x1 >= 0 and self.x2 >= 0 and self.x3 >= 0): #Messy appr
```

```
            self.z1 = cos(self.phi1)
            self.y1 = sin(self.phi1)*sin(self.lamda1)
            self.z2 = cos(self.phi2)
            self.y2 = sin(self.phi2)*sin(self.lamda2)
            self.z3 = cos(self.phi3)
            self.y3 = sin(self.phi3)*sin(self.lamda3)
            self.onePoint = False
            self.twoPoint = False
            self.OTE1 = False
            self.OTE2 = False
            self.OTE3 = False
        elif (self.x1 >= 0 and self.x2 >= 0 and self.x3 < 0):
            self.z1 = cos(self.phi1)
            self.y1 = sin(self.phi1)*sin(self.lamda1)
            self.z2 = cos(self.phi2)
            self.y2 = sin(self.phi2)*sin(self.lamda2)
            self.z3 = 0
            self.y3 = 0
            self.onePoint = True
            self.twoPoint = False
            self.OTE1 = False
            self.OTE2 = False
            self.OTE3 = True
        elif (self.x1 >= 0 and self.x2 < 0 and self.x3 >= 0):
            self.z1 = cos(self.phi1)
            self.y1 = sin(self.phi1)*sin(self.lamda1)
            self.z3 = cos(self.phi3)
            self.y3 = sin(self.phi3)*sin(self.lamda3)
            self.z2 = 0
```

```
            self.y2 = 0
            self.onePoint = True
            self.twoPoint = False
            self.OTE1 = False
            self.OTE2 = True
            self.OTE3 = False
        elif (self.x1 < 0 and self.x2 >= 0 and self.x3 >= 0):
            self.z3 = cos(self.phi3)
            self.y3 = sin(self.phi3)*sin(self.lamda3)
            self.z2 = cos(self.phi2)
            self.y2 = sin(self.phi2)*sin(self.lamda2)
            self.z1 = 0
            self.y1 = 0
            self.onePoint = True
            self.twoPoint = False
            self.OTE1 = True
            self.OTE2 = False
            self.OTE3 = False
        elif (self.x1 >= 0 and self.x2 < 0 and self.x3 < 0):
            self.z1 = cos(self.phi1)
            self.y1 = sin(self.phi1)*sin(self.lamda1)
            self.z2 = 0
            self.y2 = 0
            self.z3 = 0
            self.y3 = 0
            self.onePoint = True
            self.twoPoint = True
            self.OTE1 = False
            self.OTE2 = True
```

```
            self.OTE3 = True
        elif (self.x1 < 0 and self.x2 >= 0 and self.x3 < 0):
            self.z2 = cos(self.phi2)
            self.y2 = sin(self.phi2)*sin(self.lamda2)
            self.z1 = 0
            self.y1 = 0
            self.z3 = 0
            self.y3 = 0
            self.onePoint = True
            self.twoPoint = True
            self.OTE1 = True
            self.OTE2 = False
            self.OTE3 = True
        elif (self.x1 < 0 and self.x2 < 0 and self.x3 >= 0):
            self.z3 = cos(self.phi3)
            self.y3 = sin(self.phi3)*sin(self.lamda3)
            self.z1 = 0
            self.y1 = 0
            self.z2 = 0
            self.y2 = 0
            self.onePoint = True
            self.twoPoint = True
            self.OTE1 = True
            self.OTE2 = True
            self.OTE3 = False
        else:
            self.z1 = 0
            self.y1 = 0
            self.z2 = 0
```

```
            self.y2 = 0
            self.z3 = 0
            self.y3 = 0
            self.onePoint = True
            self.twoPoint = True
            self.OTE1 = True
            self.OTE2 = True
            self.OTE3 = True

    @staticmethod
    def run():
        areaSpot = 0
        radialSweep = 0
        areaList = []
        x = []
        count = 0
        while (radialSweep < 2*2*pi):
            x.append(count)
            for s in Spot.spotList:
                #calculate total area of spot
                a = s.calcAreaCart()
                areaSpot+= a
                areaSpotCor = (areaSpot) #adding a correction because the
            areaList.append(areaSpotCor)
            print (areaSpot)
            for s in Spot.spotList:
                #increase all radial angles by 0.1 radians
                s.lamda1 += 0.1
                s.lamda2 += 0.1
```

```
                s.lamda3 += 0.1
                s.redo()
                #areaSpot += s.calcArea
            areaSpot = 0
            radialSweep += 0.1
            count+=1
        plt.plot(x, areaList, 'b.')
        plt.ylim(-0.01,0.01)
        plt.show() #supressing plot show so model can run
        return areaList

    def plotCir(self):
        points = numpy.linspace(-1,1)
        y = []
        yy = []
        for x in points:
            y.append(math.sqrt(1-x**2))
        for x in points:
            yy.append(-math.sqrt(1-x**2))
        plt.plot(points, y, "b.", points, yy, "b.")
        plt.show()

    @staticmethod
    def makeCir(radius, phiCen, lamdaCen):
        x = 0
        delPhi = 0.01
        while (delPhi < 2*radius):
            delLamda = (cos(radius)/(cos(phiCen)*cos(phiCen-radius+delPh
            s = Spot(phiCen, lamdaCen, phiCen-radius, lamdaCen, phiCen-ra
```

```
            p = Spot(phiCen, lamdaCen, phiCen-radius, lamdaCen, phiCen-ra
            delPhi+=0.01

    @staticmethod
    def makeStr(length, width, phiStart, lamStart): #creates a stripe wh
        lenCount = 0
        delLamda = 0.001 #this defines the resolution of the stripe
        while(lenCount < length):
            delLamda += lenCount
            s = Spot(phiStart, lamStart+delLamda, phiStart+width, lamSta
            p = Spot(phiStart, lamStart+delLamda, phiStart, lamStart+(2*
            delLamda += 0.01
            lenCount += delLamda*math.pi/180

    @staticmethod
    def clear():
        Spot.spotList = []
```

Appendix B

ANALYSIS.PY

```
#Analysis Program for use with Spot Model Program
#Kyle Dettman 2015

import numpy
import SpotModel as sm
from math import pi
from math import sin

def partition(fileIn, outPre):
    fr = open(fileIn, 'r')
    listTime = []
    listCount = []

    #Separate file into readable lists
    for line in fr:
        hold = line.split()
        x=0
        for x in range(2):
            if (x == 0):
                listTime.append(hold[x])
            elif (x == 1):
                listCount.append(hold[x])
```

```
#find average time separation
count = 0
total = 0
x=0
for x in range(len(listTime)):
    diff = float(listTime[x])-float(listTime[x-1])
    if (diff < 0):
        diff = diff*(-1)
    count += 1
    total += diff
ave = total/count
print (str(total) + '  ' + str(count) + '\n')
print (str(ave) + '\n')
#partition data into files based on average time separation
x=1
listTimePrime = [listTime[0]]
listCountPrime = [listCount[0]]
ver = 0
for x in range(len(listTime)):
    diff = float(listTime[x])-float(listTime[x-1])
    if (diff < 0):
        diff = diff*(-1)

    if (diff <= ave):
        listTimePrime.append(listTime[x])
        listCountPrime.append(listCount[x])
    else: #If the time difference is greater than the average create
        name = outPre + '_' + str(ver) + '.txt'
```

```
            ver += 1
            fw = open(name, 'w')
            for y in range(len(listTimePrime)):
                fw.write(listTimePrime[y] + ' ' + listCountPrime[y] + '\n
            fw.close()
            listTimePrime = []
            listCountPrime = []
            print ('Created file ' + name + '\n')

def modelRun(amp, period, phase, incl=None, numSpot=None, spotVstripe=No
    if (incl is None):
        incl = 0
    if (numSpot is None):
        numspot = 1
    if (spotVstripe is None):
        spotVstripe = True
    #need to generate list of values for the fit parameters
    listFit = []
    x = 0
    while (x <= (2*2*pi)):
        point = float(amp)*sin(2*pi*float(period)*float(x) + float(phase
        listFit.append(point)
        x += 0.1 #same increment as in SpotModel.py


    x = 0
    s = sm.Spot(0,1,2,3,4,5)#making s a valid Spot object
    rad = amp
    phi = 0
    lam = 0
```

```
minum = -1
notComplete = True
while (x <= pi): #first sweep of model
    while (notComplete):
        s.clear()#clearing initialization data
        s.makeCir(rad, phi, lam)
        areaList = s.run()
        #initialize variables to track difference from input data
        count = 0
        total = 0
        for y in range (len(areaList)):
            if (listFit[y] is not None): #ensuring no length errors
                diff = float(areaList[y] - listFit[y])
                if (diff < 0):
                    diff = -1*diff
            total += diff
            count += 1
        ave = total/count
        if (minum == -1):
            minum = ave
            radMin = rad
            phiMin = phi
            lamMin = lam
        elif (ave < minum):
            minum = ave
            radMin = rad
            phiMin = phi
            lamMin = lam
        lam += 0.01
```

```
        if (lam > 2*pi):
            notComplete = False
    x+=0.01
    phi += x
    lam = 0
    notComplete = True


s.makeCir(0.01,0,0)#making a circle on the object s
```