
spiderman Documentation

Release 0.2.2

Tom Louden

Oct 18, 2018

Contents

1	Installation	3
1.1	pip	3
1.2	From source	3
1.3	Tests	3
2	Quickstart	5
3	Plotting	7
3.1	Visualising the planet	7
3.2	Generating a simple spectrum	14
4	Brightness maps	19
4.1	Spherical Harmonics	19
4.2	Zhang and Showman 2017	24
4.3	Offset hotspot	27
4.4	Two sided planet	28
5	Stellar models	37
5.1	Using the spidermanrc file	37
5.2	Precalculating grids of models	38
6	Instrument response	39
7	Acknowledgements	41
8	Release Notes	43
9	0.3.1 (2016/08/25)	45
10	0.2.2 (2016/08/10)	47

Contents:

CHAPTER 1

Installation

1.1 pip

The fastest way to get the latest stable version of `spiderman` up and running is through `pip`:

```
$ pip install spiderman-package
```

1.2 From source

To get the most up to date version of `spider` install directly from source, though stability is not guaranteed. The [development version](#) can be found on GitHub.

Installing from GitHub is straightforward, simply type

```
$ git clone https://github.com/tomlouden/SPIDERMAN
```

To ‘clone’ the repository, then `cd` into the folder and install using the setup script:

```
$ cd SPIDERMAN
$ sudo python setup.py install
```

And that’s it! (Note that you’ll need to `cd` out of the source directory **before** you can import `spiderman`.)

1.3 Tests

An Ipython notebook is included with a few examples, I recommend running it to test everything is working correctly. (built in test functions coming soon)

CHAPTER 2

Quickstart

Making a lightcurve — This is likely the basic task that you need SPIDERMAN to perform, to, for example, form the basis for a likelihood function being fed into an mcmc. Here is how to return a simulated lightcurve from SPIDERMAN, the fast way.

First, import spiderman into the namespace:

```
import spiderman as sp
```

Now, it is necessary to generate a parameter object, this will be the main way of interacting with the code. Before doing so, you must specify which brightness model ([link to brightness model page](#)) you wish to use. In this example we will use the analytical formula from Zhang & Showman 2016, but many other options are available (If you have a brightness model that is not represented, please contact me, and I will include it in the next release of the package).

```
spider_params = sp.ModelParams(brightness_model="zhang")
```

Now that the parameter object has been created, we can define the parameters as so:

```
spider_params.n_layers= 5
```

This parameter refers to the number of layers in the 2d “web” used to define the integration grid. 5 is typically acceptable for most available data qualities. Next we will define the system parameters, using values suitable for WASP-43b

```
spider_params.t0= 200          # Central time of PRIMARY transit [days]
spider_params.per= 0.81347753  # Period [days]
spider_params.a_abs= 0.01526   # The absolute value of the semi-major axis [AU]
spider_params.inc= 82.33      # Inclination [degrees]
spider_params.ecc= 0.0        # Eccentricity
spider_params.w= 90           # Argument of periastron
spider_params.rp= 0.1594      # Planet to star radius ratio
spider_params.a= 4.855        # Semi-major axis scaled by stellar radius
spider_params.p_u1= 0         # Planetary limb darkening parameter
spider_params.p_u2= 0         # Planetary limb darkening parameter
```

Note: these definitions are compatible with Batman (Kreidberg et al 2015)

Now set the parameters specific to the brightness model that we defined earlier:

```
spider_params.xi= 0.3      # Ratio of radiative to advective timescale
spider_params.T_n= 1128    # Temperature of nightside
spider_params.delta_T= 942 # Day-night temperature contrast
spider_params.T_s = 4500   # Temperature of the star
```

Since this uses model spectra, it is necessary to specify the bandpass with these parameters:

```
spider_params.l1 = 1.1e-6   # The starting wavelength in meters
spider_params.l2 = 1.7e-6   # The ending wavelength in meters
```

Now, define the times you wish the model to be evaluated at, let's do a single full orbit:

```
t= spider_params.t0 + np.linspace(0, + spider_params.per,100)
```

Finally, a lightcurve can be generated simply by using the “lightcurve” method:

```
lc = spider_params.lightcurve(t)
plt.plot(t,lc)
```

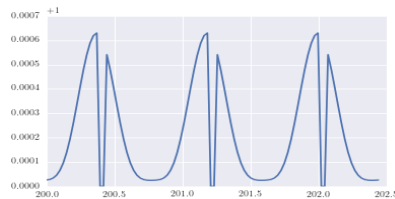


Fig. 1: The resulting lightcurve

Warning: SPIDERMAN currently only produces secondary eclipses and phase-curves - the primary eclipse will not be modelled! To model the primary eclipse an additional code, such as BATMAN (Kreidberg et al 2015) will be required.

It's that simple!

3.1 Visualising the planet

Spiderman provides a few different ways to visualise the resulting brightness distribution on the planet, below are some examples, after the planet has been specified with “spider_params”

The “plot_planet” method outputs the visible face of the planet at the specified time (t), using the same code that is used internally for calculating the model output. This can be a useful sanity check of what the model is doing.

By default the plot will be in a standard white theme suitable for inclusion in papers or printouts, alternatively a snazzy black theme more suited to powerpoint presentations can be selected for all spiderman plots by setting the “theme” keyword to black:

Note that to save figures like this, you will have to let matplotlib know that you want to output the figure with a black background, e.g.

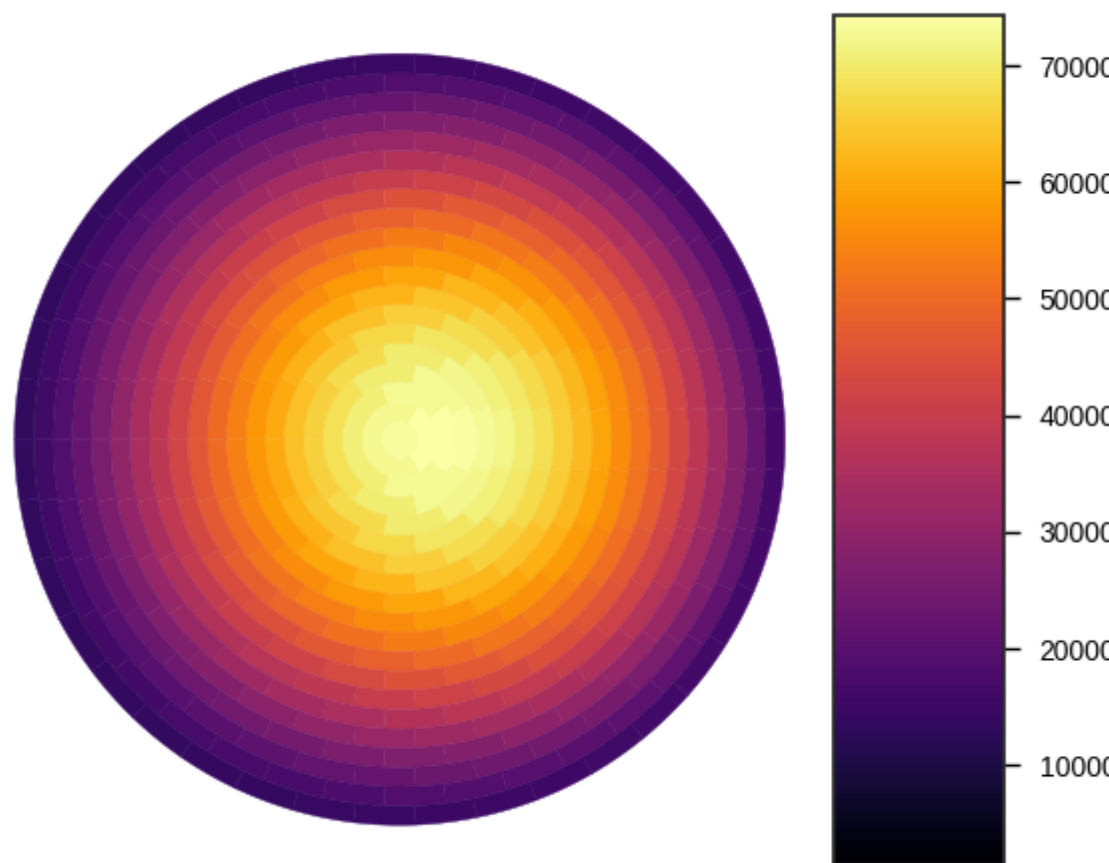
For brightness temperature based models like the Zhang model, you can also plot in temperature by setting “use_temp” to true

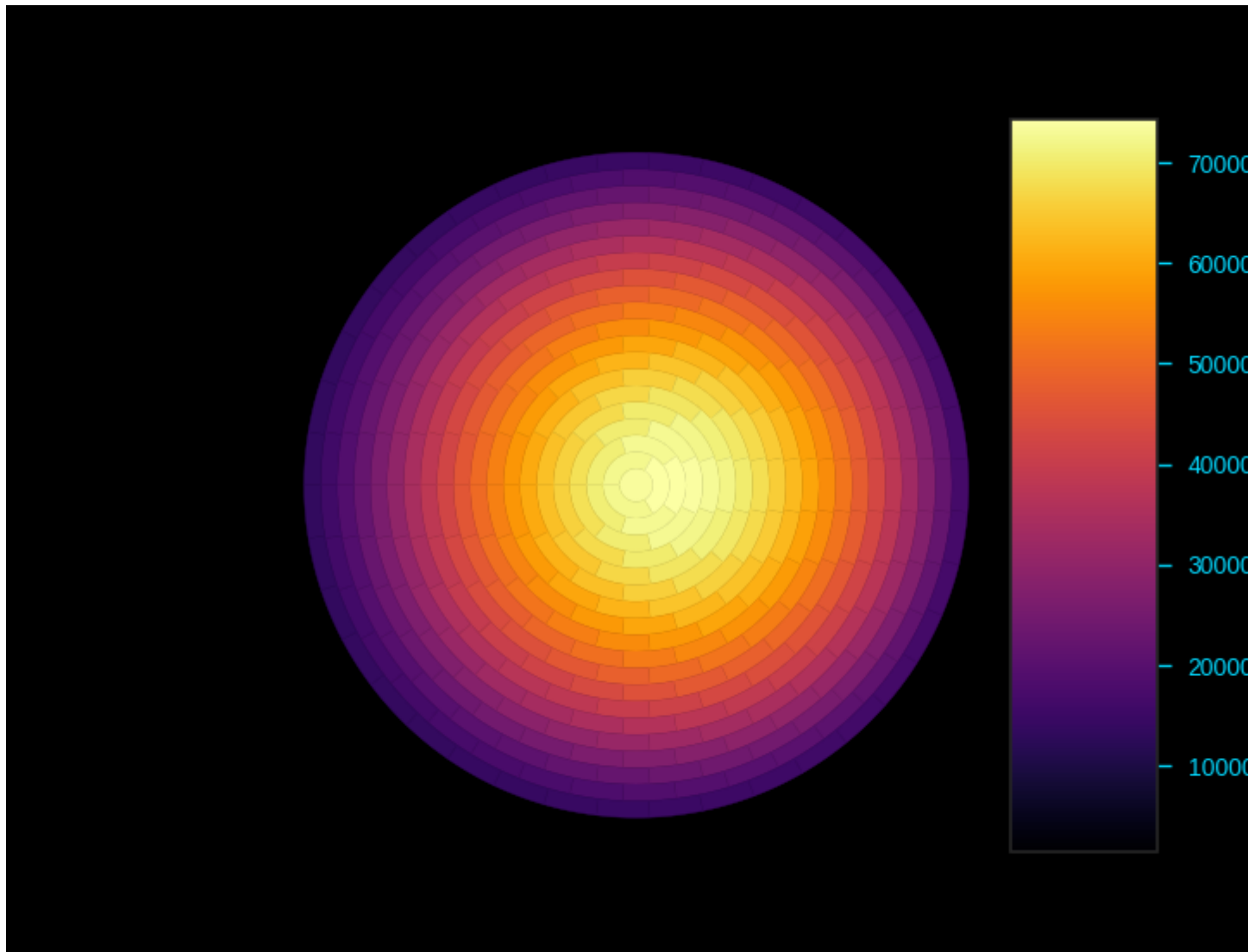
To see the system in context, you can also output a “system plot” which shows the projected position of the planet to scale with the star. Again, this plot is constructed from the same code used internally for model calculation, so is a good way to check what’s going on. Both plot_planet and plot_system can accept a phase instead of a time when the use_phase keyword is set to True

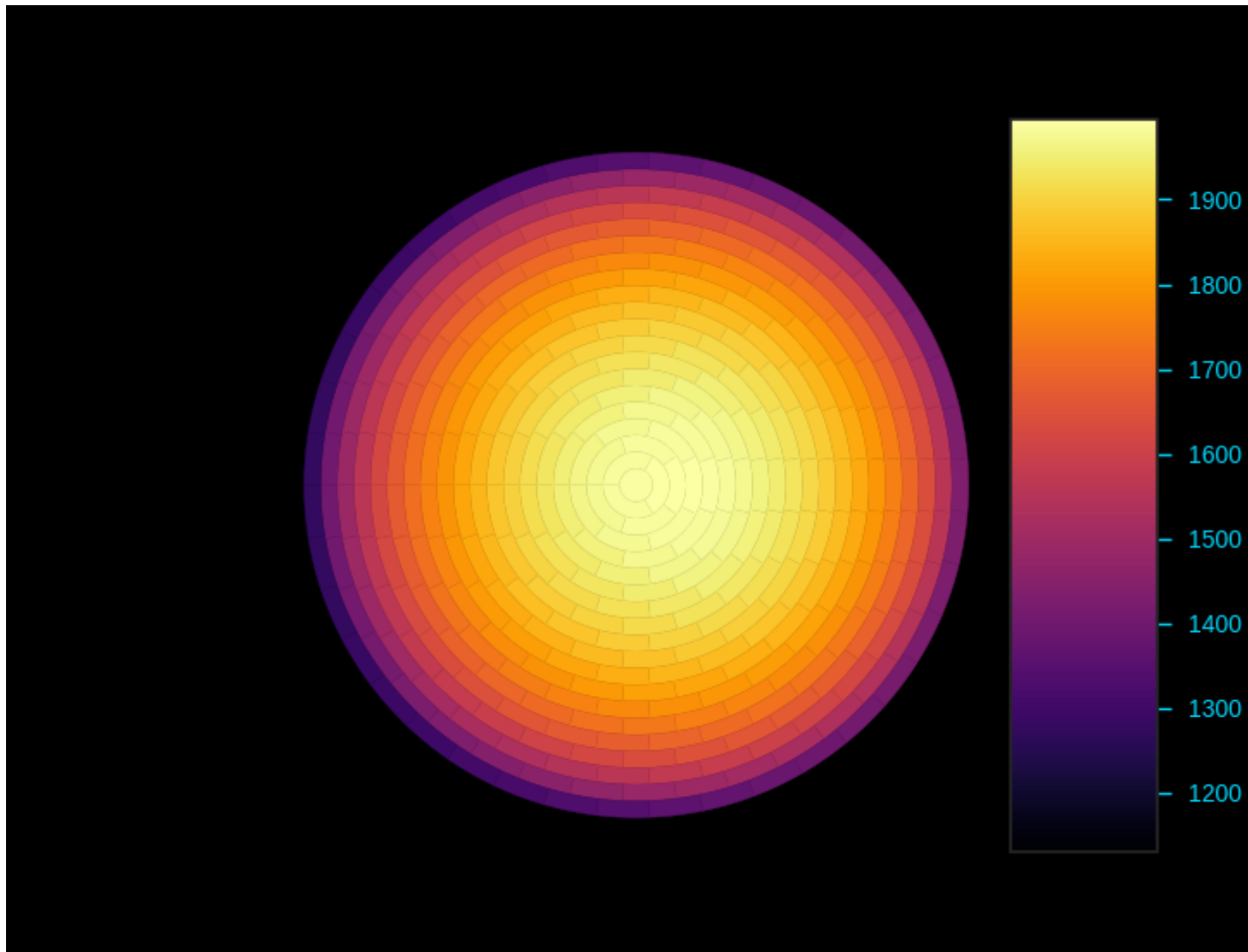
“Quad_plot” gives you a quick way to see the projected appearance of the planet at primary and secondary eclipses, and at phase 0.25 and 0.75. It has the same theming options and is able to plot either flux or temperature.

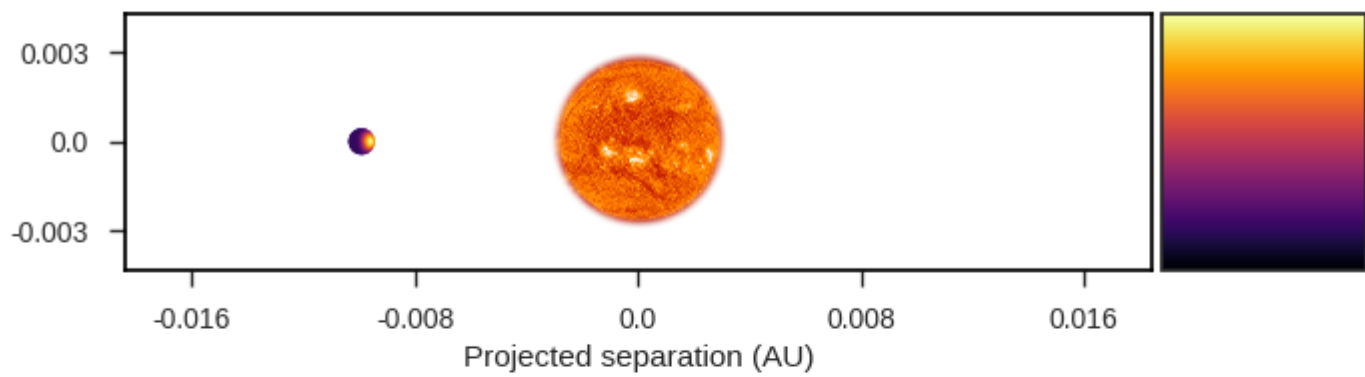
Sometimes you just want a straightforward global map - in spiderman this is generated by the “square_plot” command. This function does not use the typical spiderman segment scheme, and instead evaluates the brightness function on an evenly spaced grid in longitude and latitude. The number of grid points in the latitude and longitude direction are specified by nla and nlo keywords, respectively.

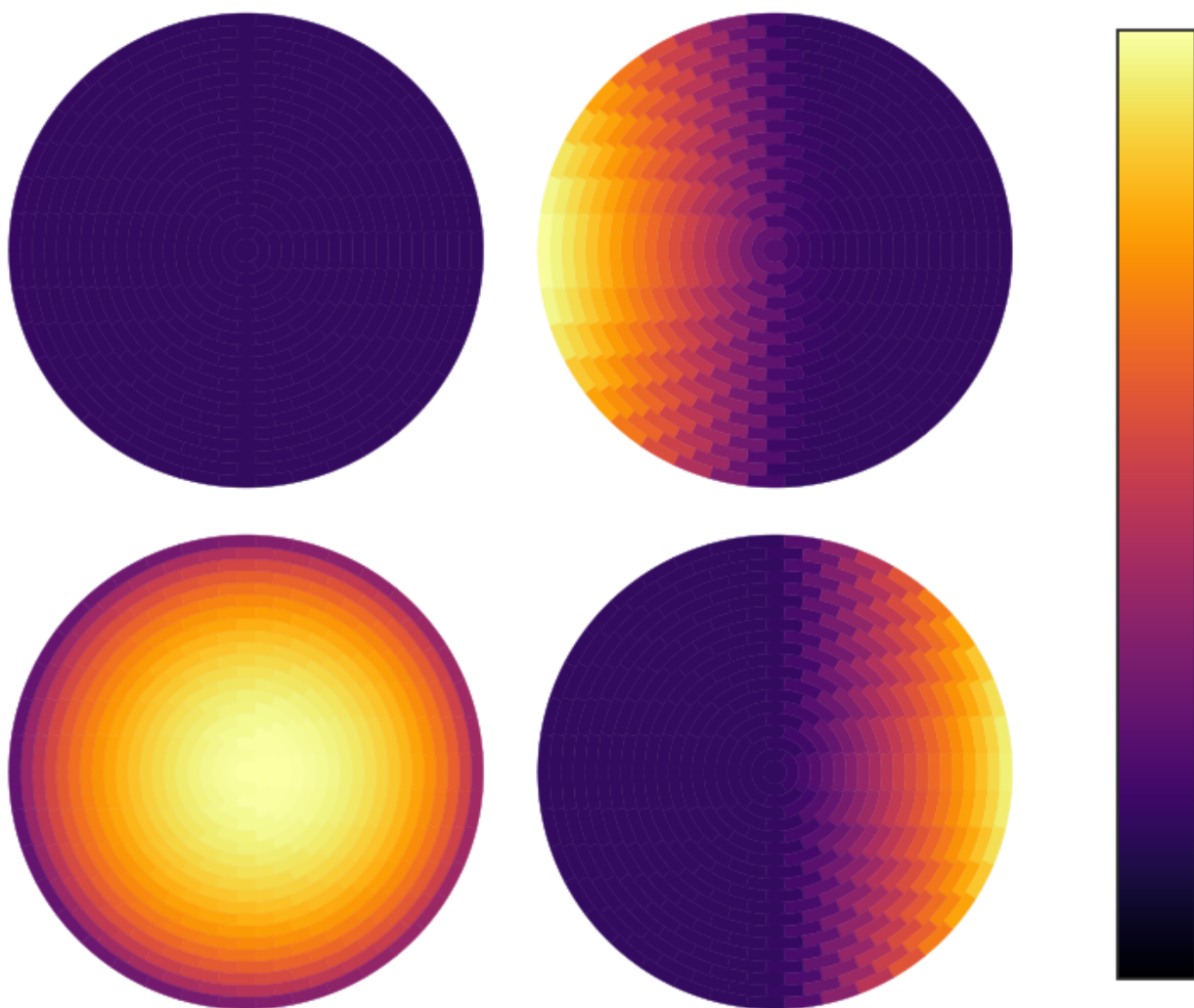
end

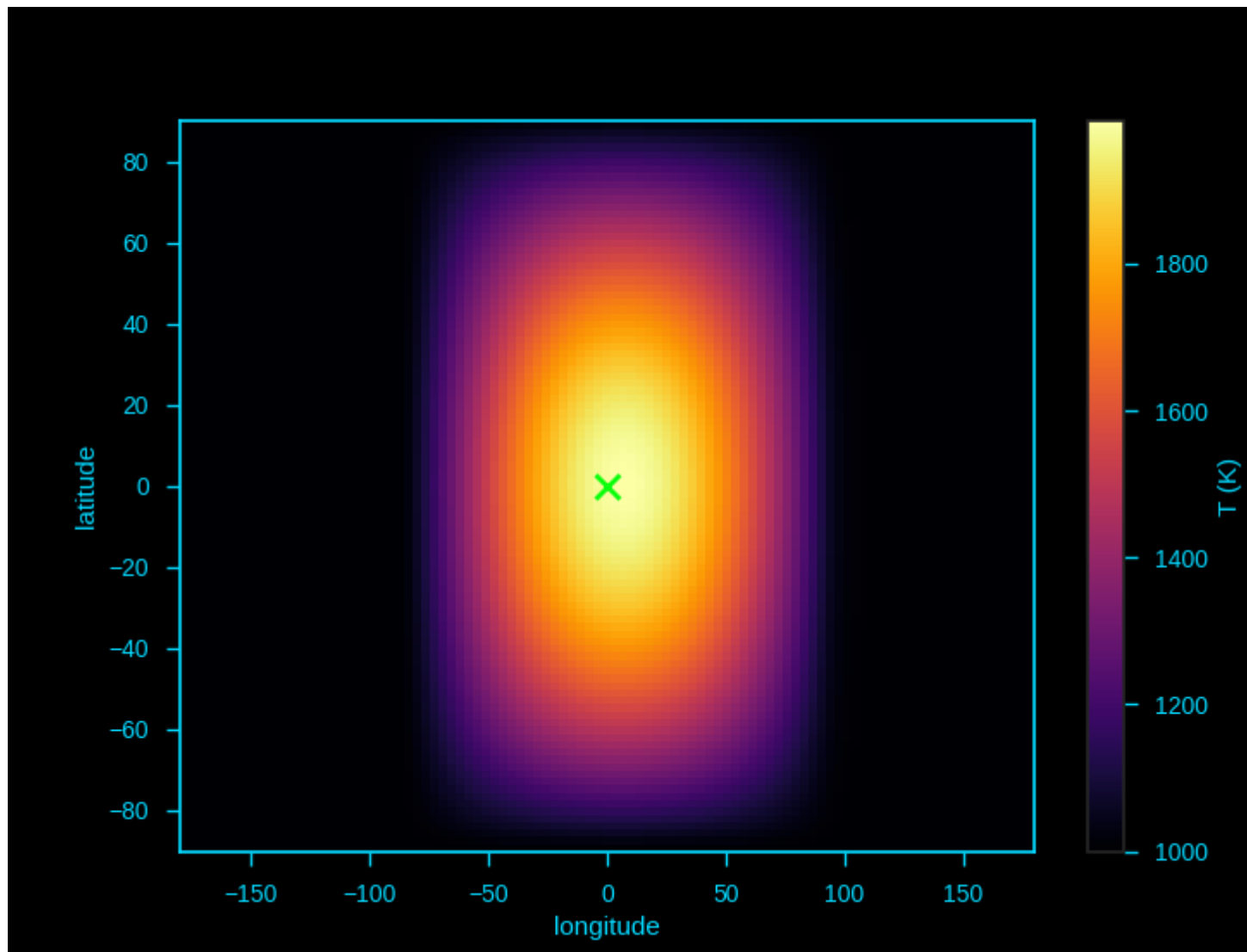












3.2 Generating a simple spectrum

Sometimes you don't want to bother with running a full orbital model, and just want a quick estimate of the eclipse depth of system. Spiderman has a couple of techniques to allow you to do this.

If all you want is the eclipsed depth, you can use the “eclipse_depth” method, like so:

```
import spiderman as sp
import numpy as np
import matplotlib.pyplot as plt

spider_params = sp.ModelParams(brightness_model='zhang')
spider_params.n_layers = 5
```

for this example we'll use a Zhang and Showman type model with 5 layers, next the relevant model parameters are entered -

```
spider_params.l1 = 1.1e-6      # The starting wavelength in meters
spider_params.l2 = 1.7e-6      # The ending wavelength in meters

spider_params.T_s = 4520
spider_params.rp = 0.159692

spider_params.xi = 0.1
spider_params.T_n = 1000
spider_params.delta_T = 1000
```

Note that if all you want is a simple eclipse depth, there's no need to enter the orbital parameters. Spiderman will assume a circular orbit and an inclination of 90 degrees unless you tell it otherwise. Now, you can call the eclipse_depth:

```
d = spider_params.eclipse_depth()
print(d)
>> 0.00045781826310942186
```

This method can be used to quickly generate an occultation spectrum of the depth as a function of wavelength, like so:

```
min_wvl = 1.1e-6
max_wvl = 1.7e-6
steps = 10
wvl_step = (max_wvl-min_wvl)/steps

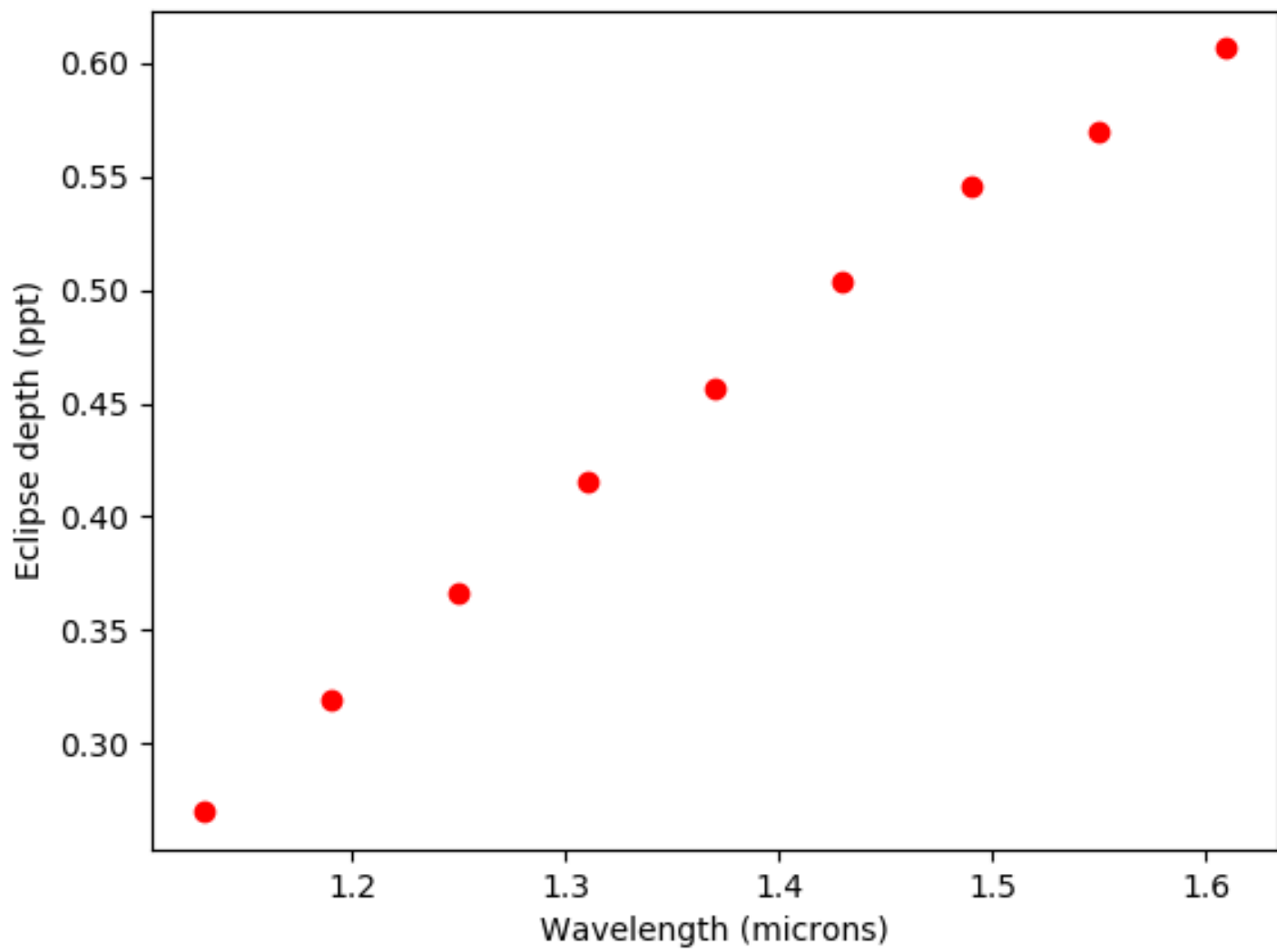
for i in range(0,steps-1):
    spider_params.l1 = min_wvl + wvl_step*i
    spider_params.l2 = min_wvl + wvl_step*(i+1)

    mid_wvl = min_wvl + wvl_step*(i+0.5)
    d = spider_params.eclipse_depth()
    plt.plot(mid_wvl*1e6,d*1000,'ro')

plt.xlabel('Wavelength (microns)')
plt.ylabel('Eclipse depth (ppt)')
```

Some caution must be used with this method, as it only returns the *blocked light* relative to the stellar brightness at the specified phase - so for an example, if you were to specify a grazing transit you would not receive the total flux of the dayside.

If you do want the total flux of the planet from a specific phase, you can instead use the “phase_brightness” method. Using this method you can calculate the emission spectrum of the planet in *physical units* at the phase of your choosing,



it is called in a similar way to `eclipse_depth`, but has an optional phase argument which can accept either a single phase value or a list. You can provide a planet radius to receive the total band luminosity of the visible hemisphere in Watts, if this is not given then an average surface intensity will be returned.

```
mid_wvls = []
p1s = []
p2s = []
p3s = []
p4s = []

for i in range(0, steps-1):
    spider_params.l1 = min_wvl + wvl_step*i
    spider_params.l2 = min_wvl + wvl_step*(i+1)

    mid_wvl = min_wvl + wvl_step*(i+0.5)
    mid_wvls += [mid_wvl*1e6]

    p1, p2, p3, p4 = spider_params.phase_brightness([0.0, 0.25, 0.5, 0.75], planet_
    ↪radius=6.9911e7)

    p1s += [p1]
    p2s += [p2]
    p3s += [p3]
    p4s += [p4]

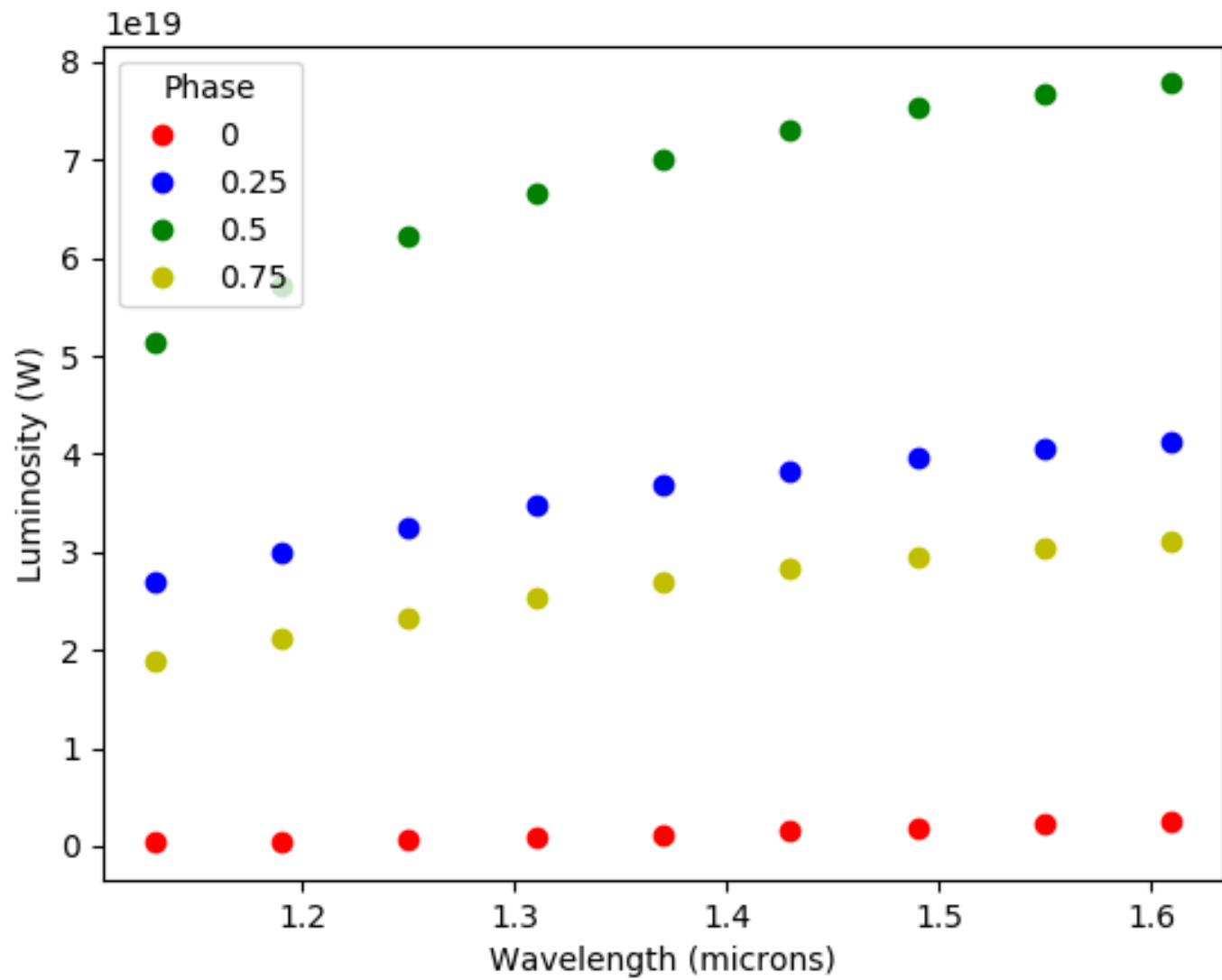
plt.plot(mid_wvls, p1s, 'ro', label = '0')
plt.plot(mid_wvls, p2s, 'bo', label = '0.25')
plt.plot(mid_wvls, p3s, 'go', label = '0.5')
plt.plot(mid_wvls, p4s, 'yo', label = '0.75')

plt.legend(title='Phase')
plt.xlabel('Wavelength (microns)')
plt.ylabel('Luminosity (W / sr)')
```

Finally, you can use the `total_luminosity` method to return the total band luminosity of the planet in the model in Watts, a planet radius in meters is required:

```
spider_params.l1 = 1.1e-6
spider_params.l2 = 1.7e-6
lum = spider_params.total_luminosity(10*6.9911e7)
print(lum)
>> 7.03802421799e+20
```

end.



4.1 Spherical Harmonics

A spherical harmonics model, the user can specify the co-efficients for as many terms as desired. Spherical harmonics models are useful, since they do not make any physical assumptions about the distribution you wish to recover. It can be useful to check the results of a physically motivated model against the results of a spherical harmonic fit.

main parameters:

- degree** The maximum degree of Harmonic you want to consider (You won't typically want more than 2)
- la0** Offset of the center of the co-ordinate centre from the substellar point in the latitude direction (unit: Degrees)
- lo0** Offset of the center of the co-ordinate centre from the substellar point in the longitude direction (unit: Degrees)
- sph** A list of the co-efficients for the harmonic terms, there *must* be the appropriate number (degree squared), and arranged in the correct order: [l0, l1 m-1, l1 m0, l1 m1, l2 m-2, l2 m-1, l2 m0, l2 m1, l2 m2. etc]. These parameters are scaled to be relative to the stellar flux, so will typically be of order $1e-3$ - $1e-4$.

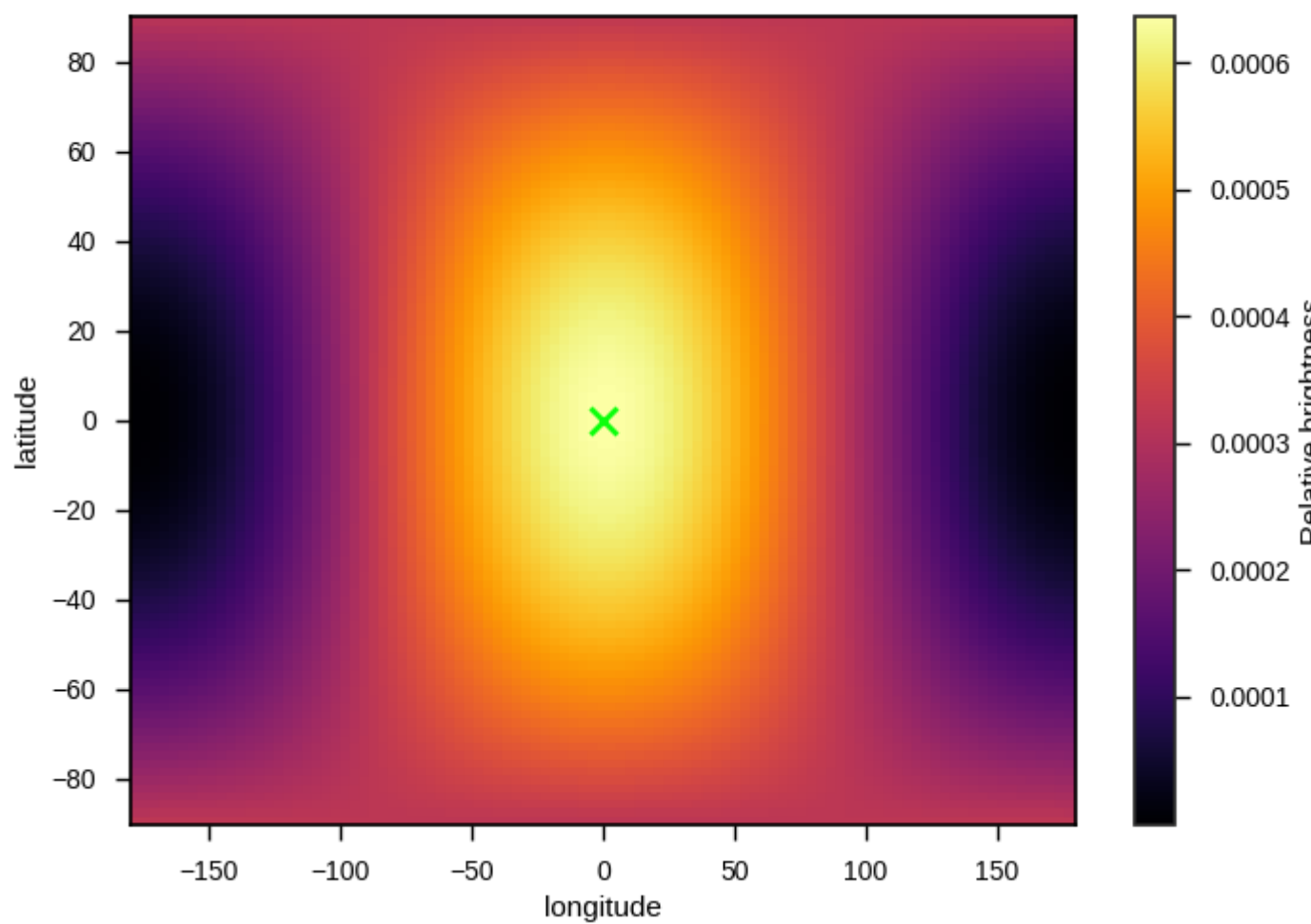
Warning: There is nothing implicit in this spherical harmonics implementation to prevent negative surface fluxes! It is suggested that care is taken when specifying priors to prevent unphysical results.

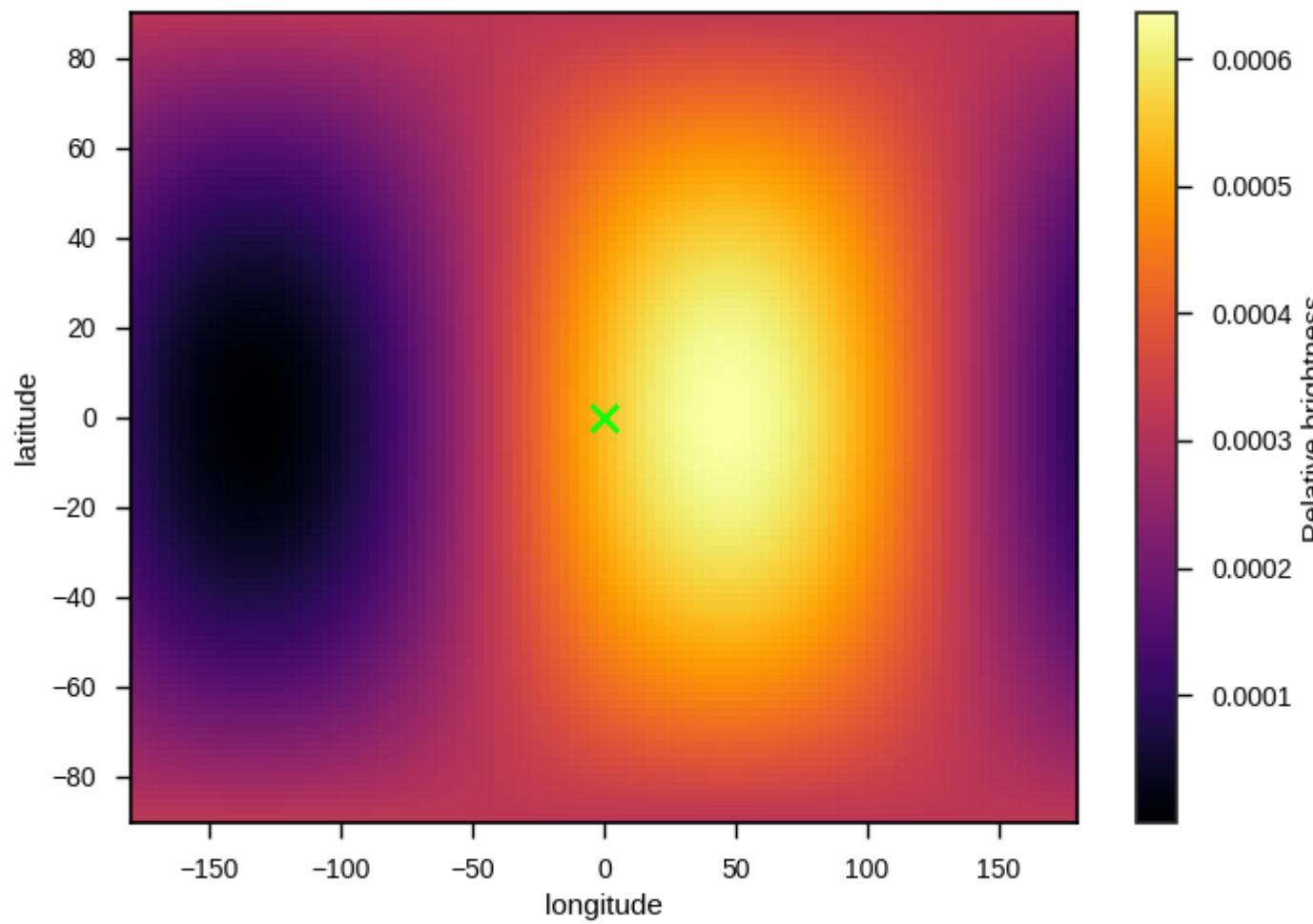
An example square plot using a two degree spherical harmonic using the l0 m0 and l1 m1 terms only - this is a simple dipole, and can represent a day/night side difference:

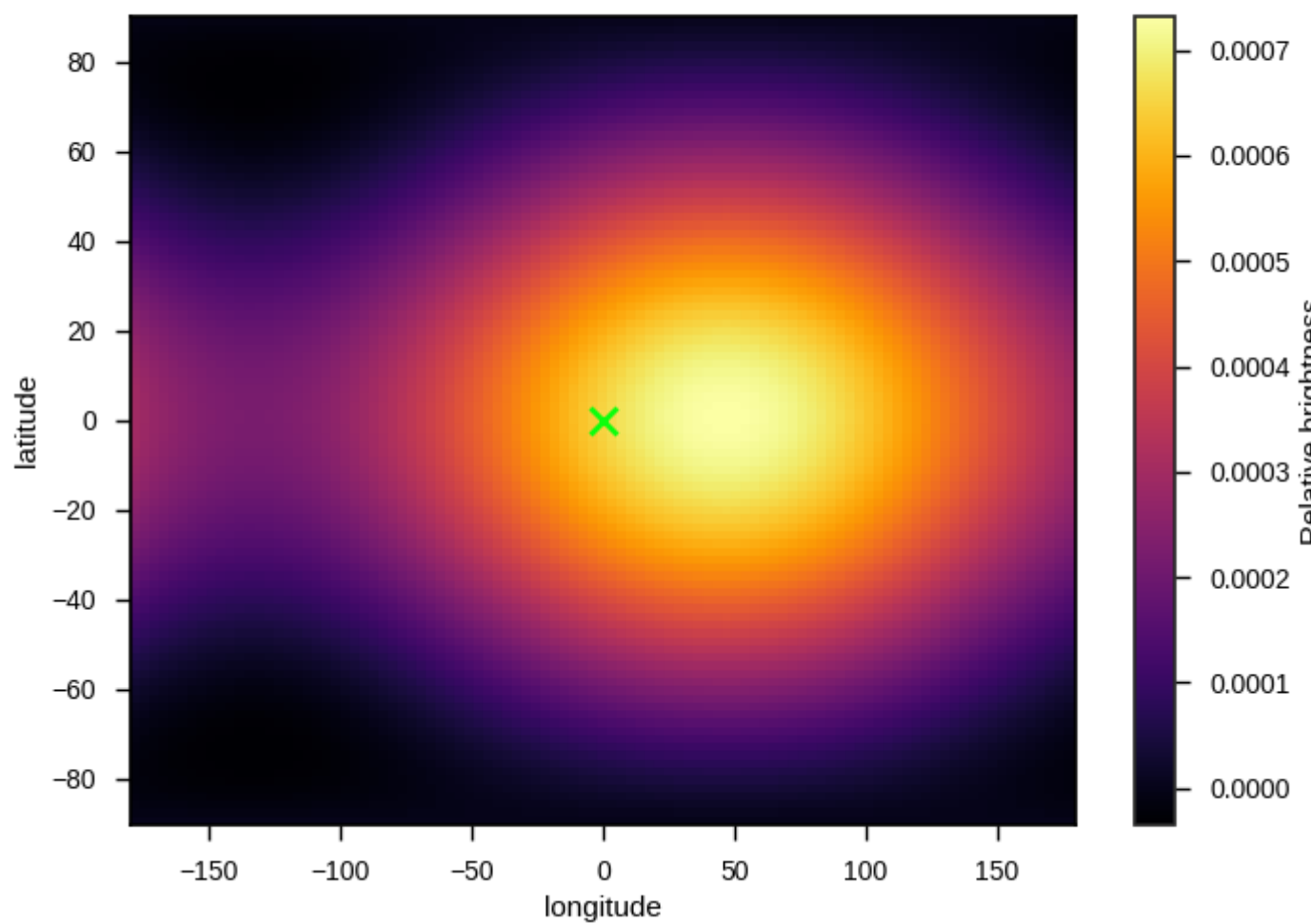
This time the hotspot is offset by adding the l1 m-1 term. (the same effect can also be achieved by changing the la0 and lo0 parameters, but never try to fit for both simultaneously, as it's degenerate!):

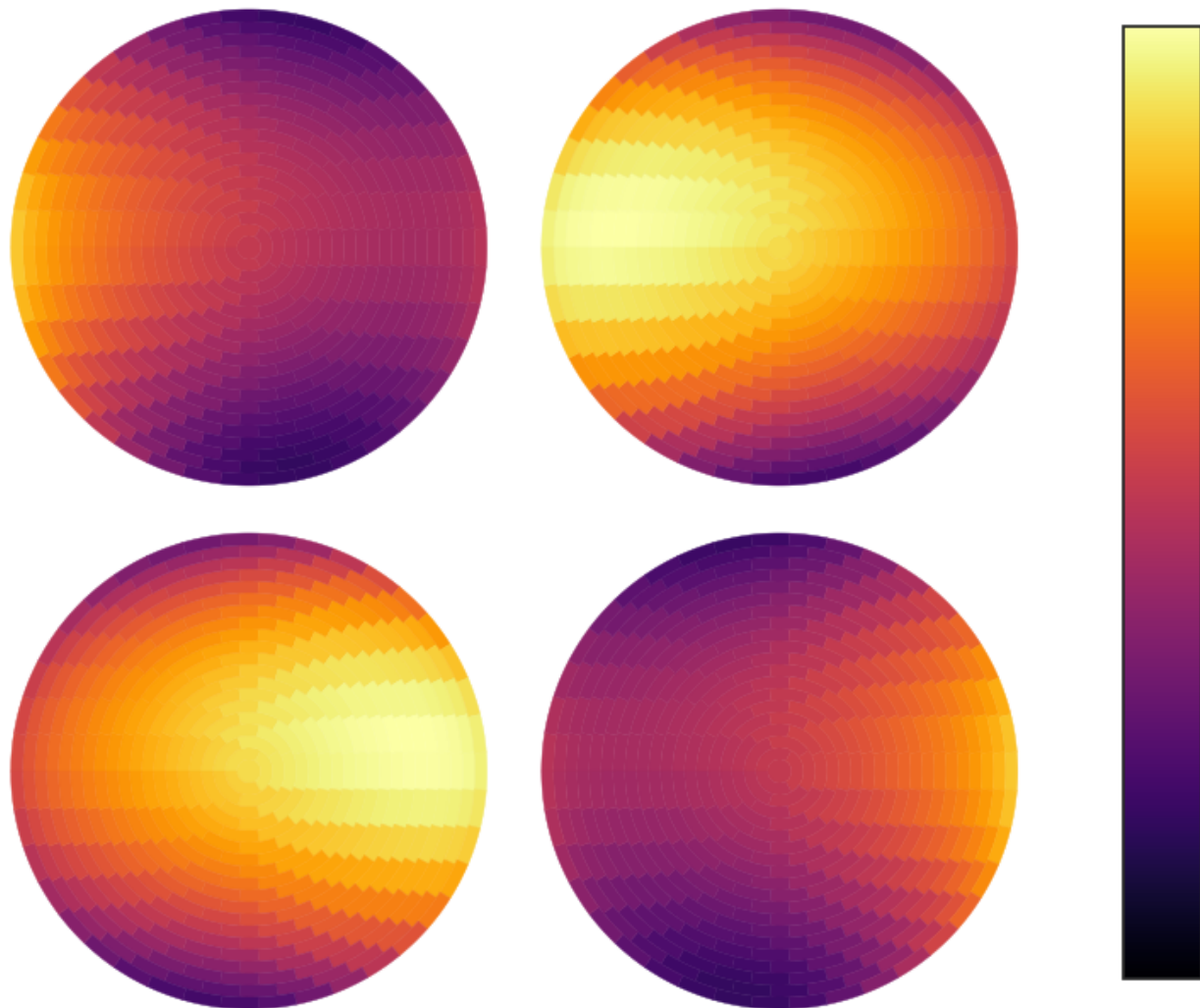
Now, with a higher order term added, l2 m0, to concentrate flux towards the equator.

An example four phase plot with this distribution:

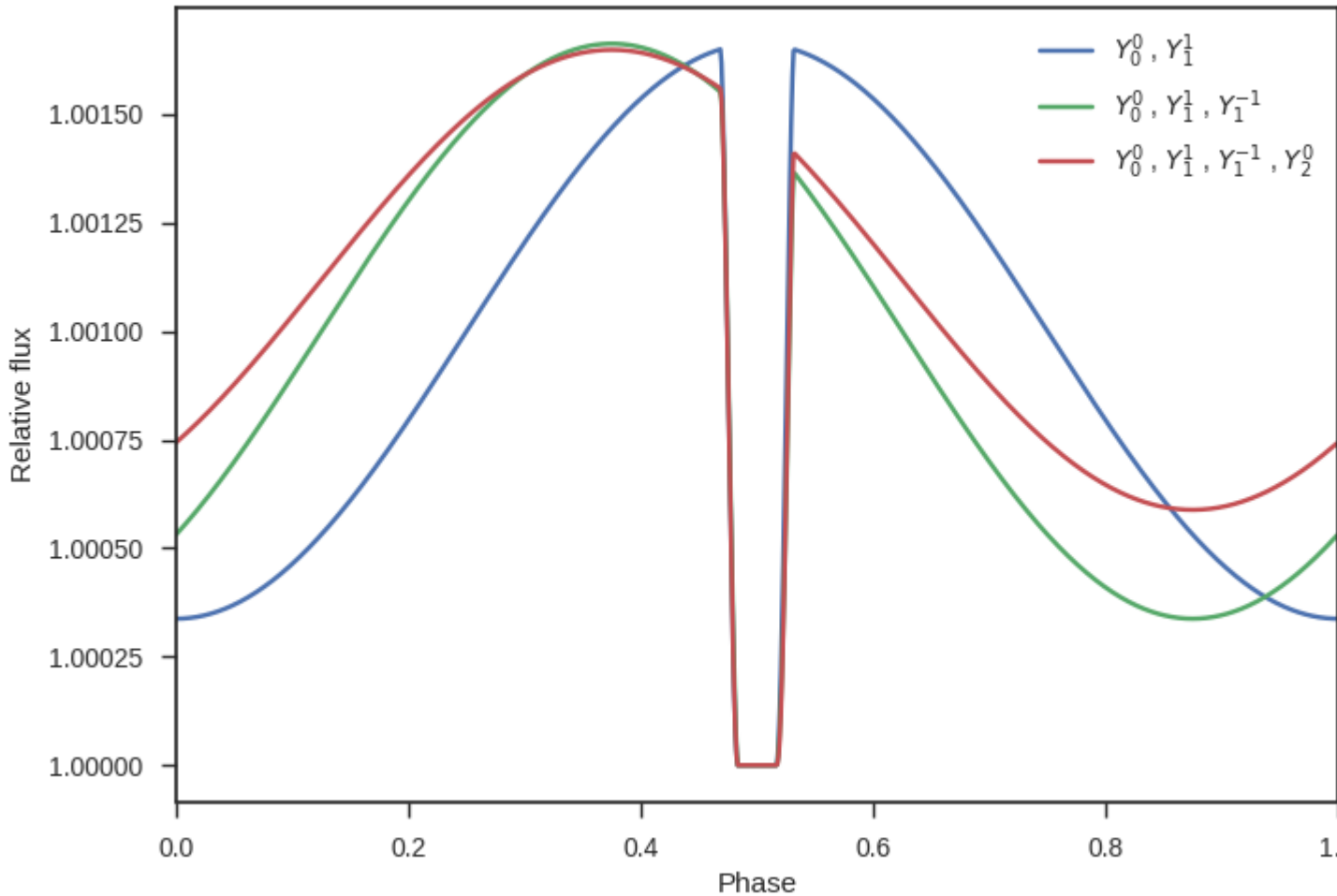








The resulting lightcurves for the three example distributions:



4.2 Zhang and Showman 2017

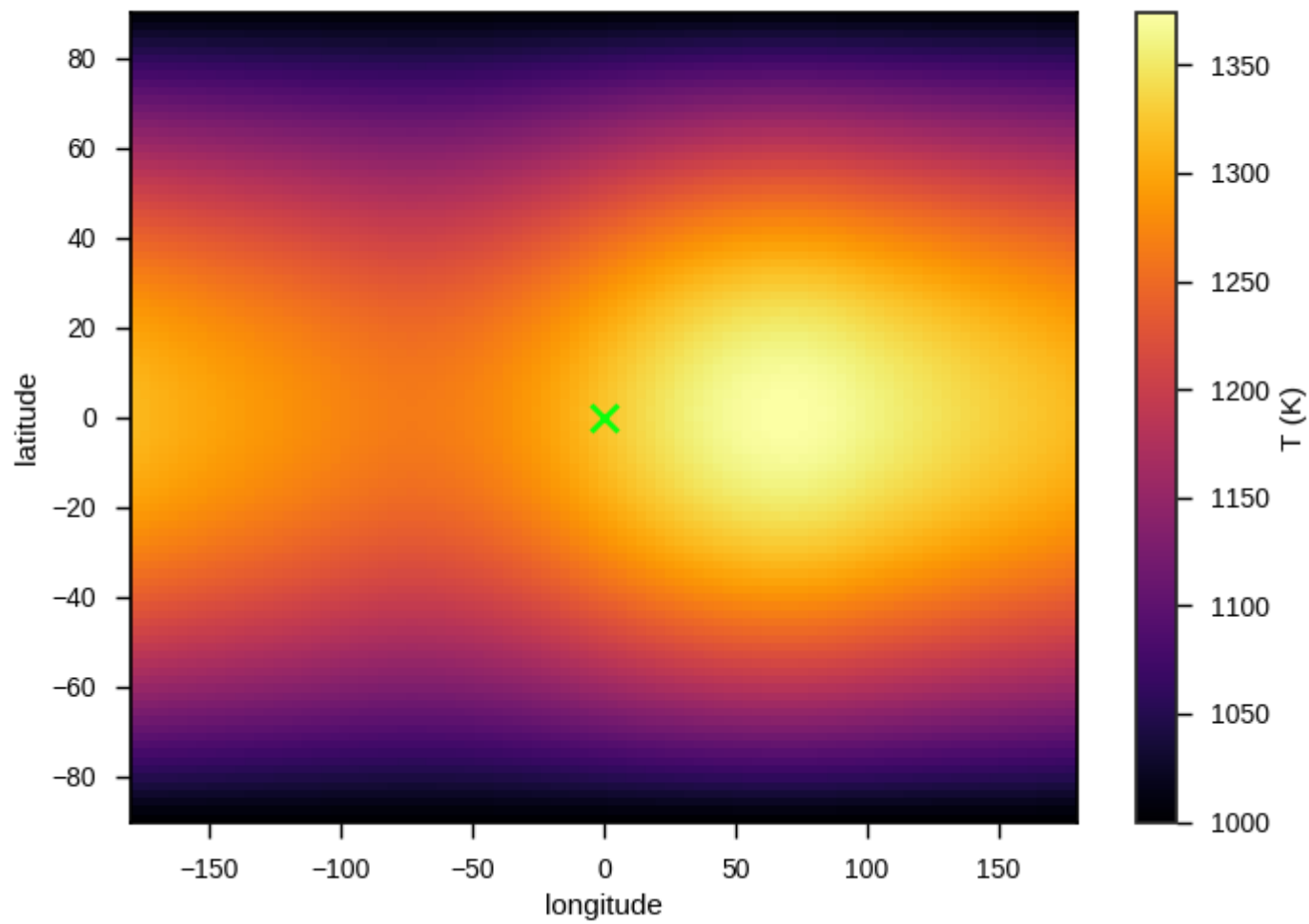
A temperature map based on the equations given in the appendix of Zhang and Showman 2017 (<http://adsabs.harvard.edu/abs/2017ApJ...836...73Z>) This semi-physical model well reproduces the main features of hot Jupiter phase-curves - offset hotspots

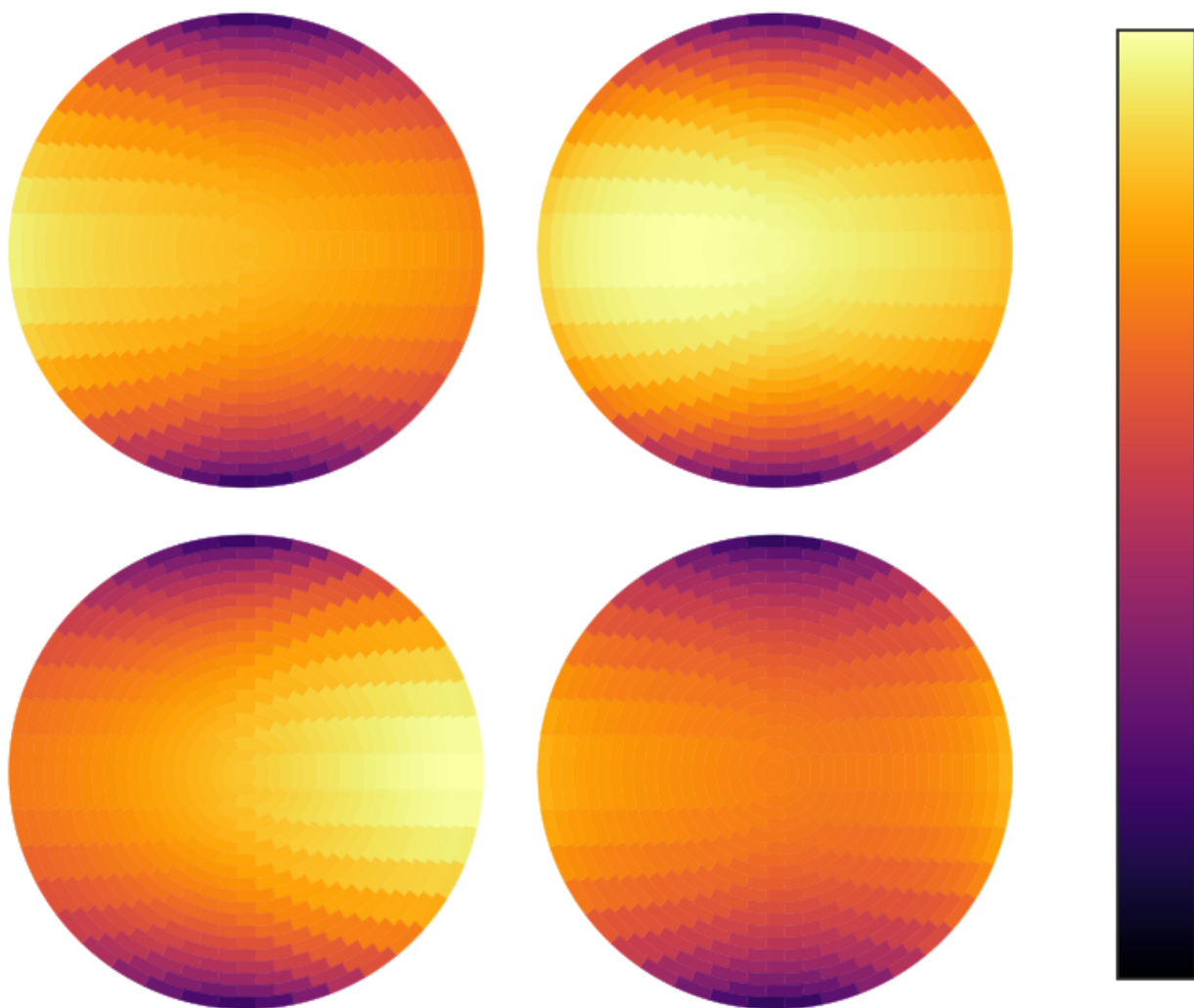
main parameters:

- xi** Ratio of radiative to advective timescale (unit: Unitless)
- T_n** Temperature of the nightside of the planet (unit: Kelvin)
- delta_T** Day-night temperature contrast (unit: Kelvin)

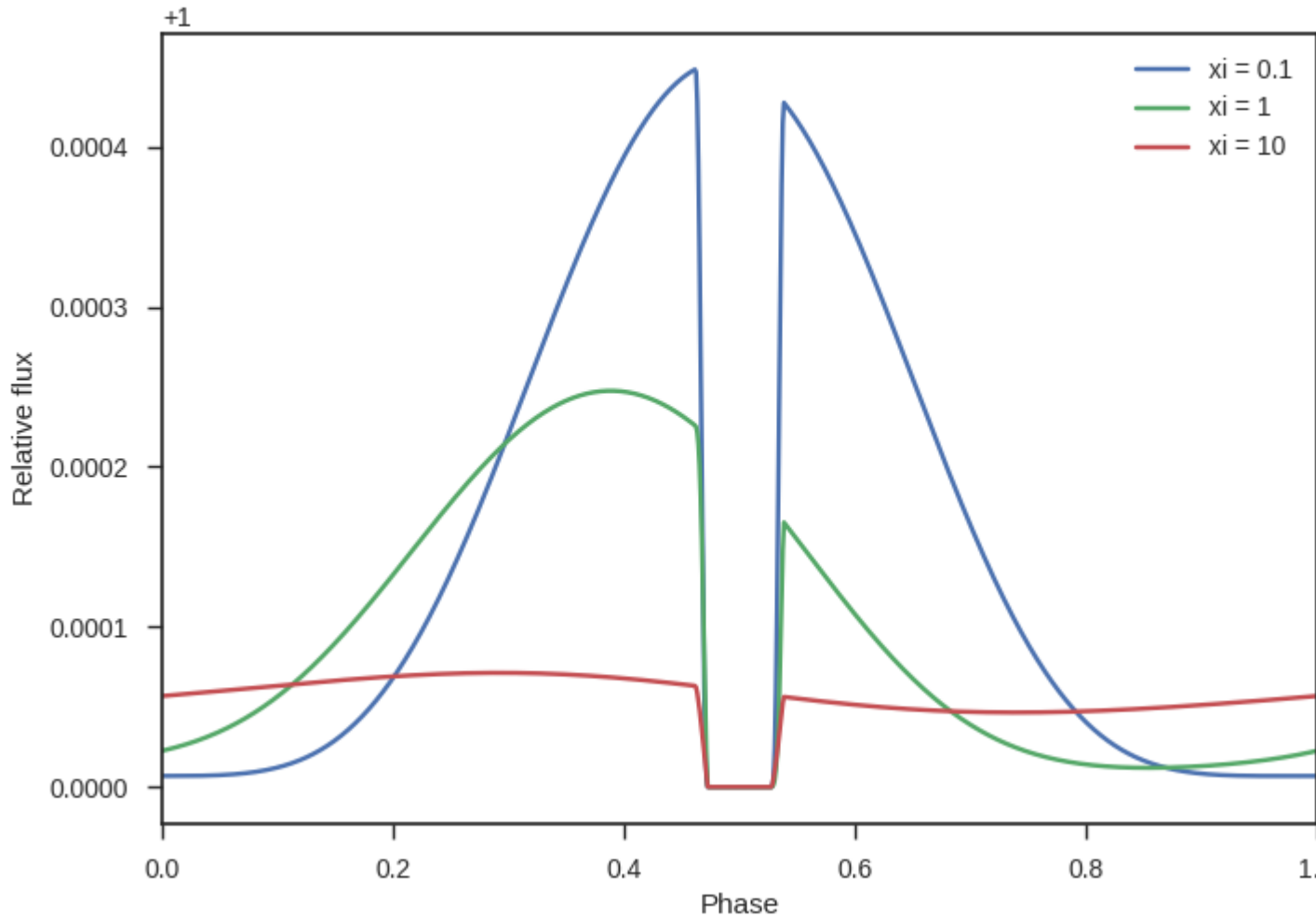
An example square plot:

An example four phase plot:





The resulting lightcurves for several parameter values



4.3 Offset hotspot

main parameters:

la0 Offset of the center of the hotspot in the latitude direction (unit: Degrees)

lo0 Offset of the center of the hotspot in the longitude direction (unit: Degrees)

size The radius of the hotspot in degrees, i.e., 90 means the hotspot covers a whole hemisphere. (unit: degrees)

The hotspot can either be specified as “hotspot_b”, to directly specify the fractional brightness, in which case these parameters are used:

spot_b The surface brightness of the hotspot as a fraction of the surface brightness of the star, typically of order $\sim 1e-4$ for hot Jupiters (unitless)

p_b The surface brightness of the planet that is not in the hotspot as a fraction of the surface brightness of the star. This value will depend strongly on the physics of heat transport in the planets atmosphere and may be several orders of magnitude fainter than the spot (unitless)

Or as “hotspot_t” to specify in terms of brightness temperature, in which case the following parameters are used instead. In this case the wavelength range to integrate over must be specified.

spot_T The surface brightness of the hotspot as a fraction of the surface brightness of the star, typically of order $\sim 1e-4$ for hot Jupiters (unitless)

p_T The brightness temperature of the planet that is not in the hotspot as a fraction of the surface brightness of the star. This value will depend strongly on the physics of heat transport in the planets atmosphere and may be several orders of magnitude fainter than the spot (unitless)

Note: Because there is a sharp contrast in flux levels between *spot* and *not spot* regions, this brightness model can have issues with quantisation, which produces unphysical “steps” in the lightcurve. This can be for the time being be solved by including a numerical integration step in regions with sharp contrasts with the optional paramter “grid_size”

cont

grid_size This model has a sharp boundary, so can have quantization issues. Regions with sharp changes in brightness are for now integrated numerically instead of analytically, this sets the number of grid points to use in the integration along each direction, to the total number of additional function calls will be this value squared. Setting this too high can significantly slow the code down, however if it is too low fits may be numerically unstable. Use caution. This is a temporary fix and is intended to be removed in a future version (default: 10)

An example square plot:

An example four phase plot:

The resulting lightcurves for several parameter values

4.4 Two sided planet

This is a simple model that only assumes that the day-side of the planet has a different flux or temperature to the night side. can be called as “two temperature” to specify with brightness temperature, or “two brightness” to secify by flux main parameters:

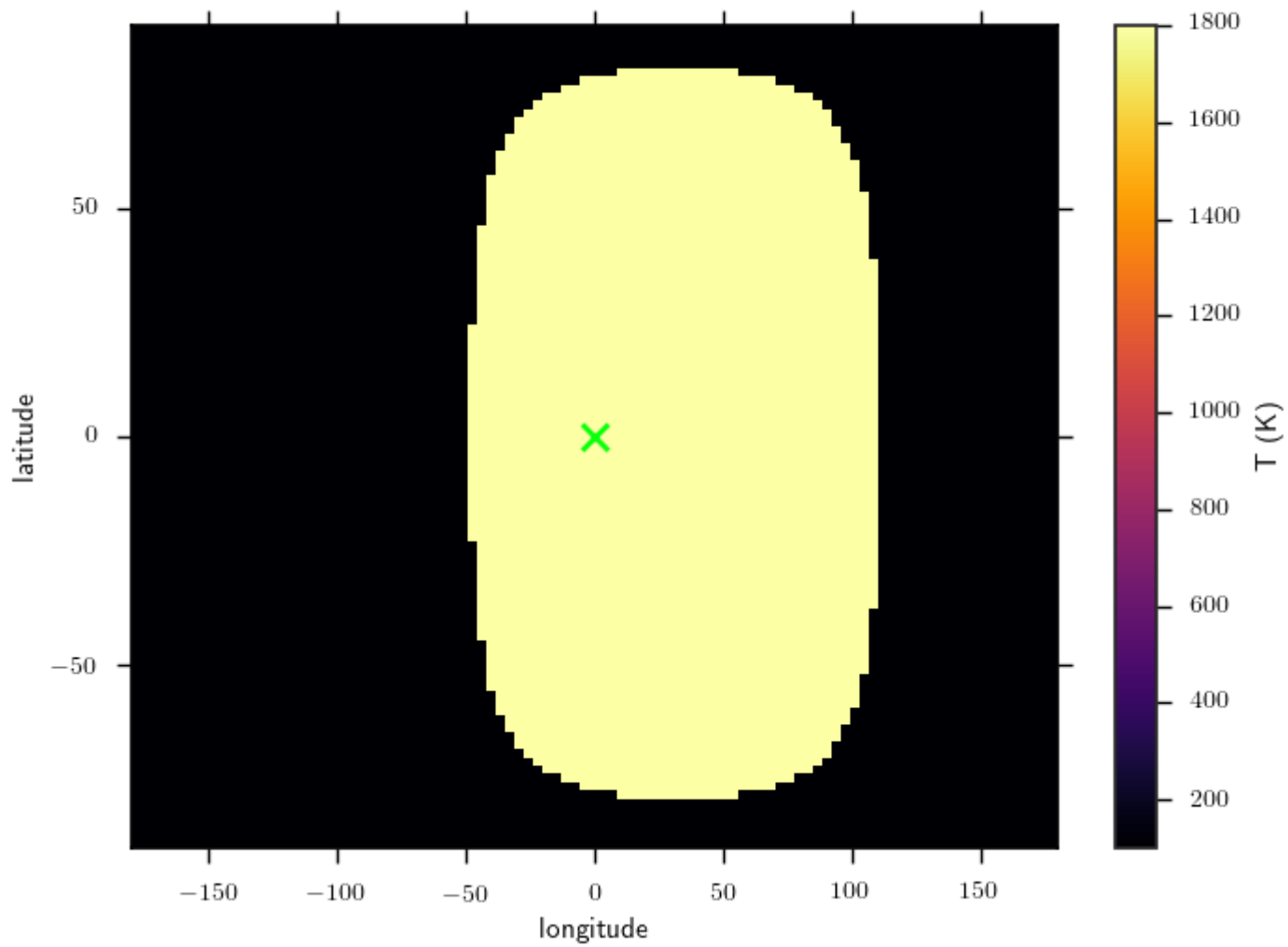
The hotspot can either be specified as “hotspot_b”, to directly specify the fractional brightness, in which case these parameters are used:

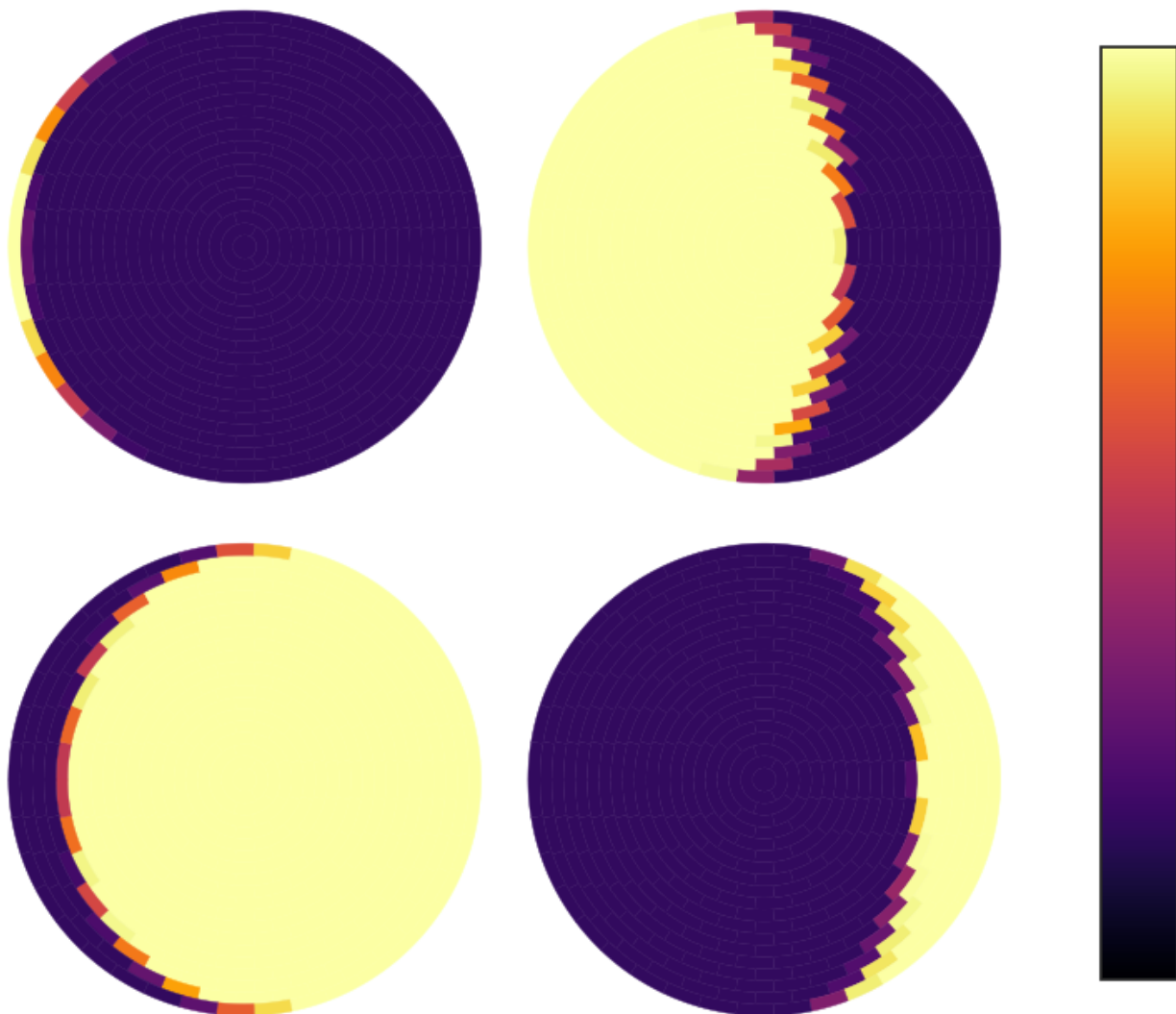
pb_d The surface brightness of the dayside as a fraction of the surface brightness of the star, typically of order $\sim 1e-4$ for hot Jupiters (unitless)

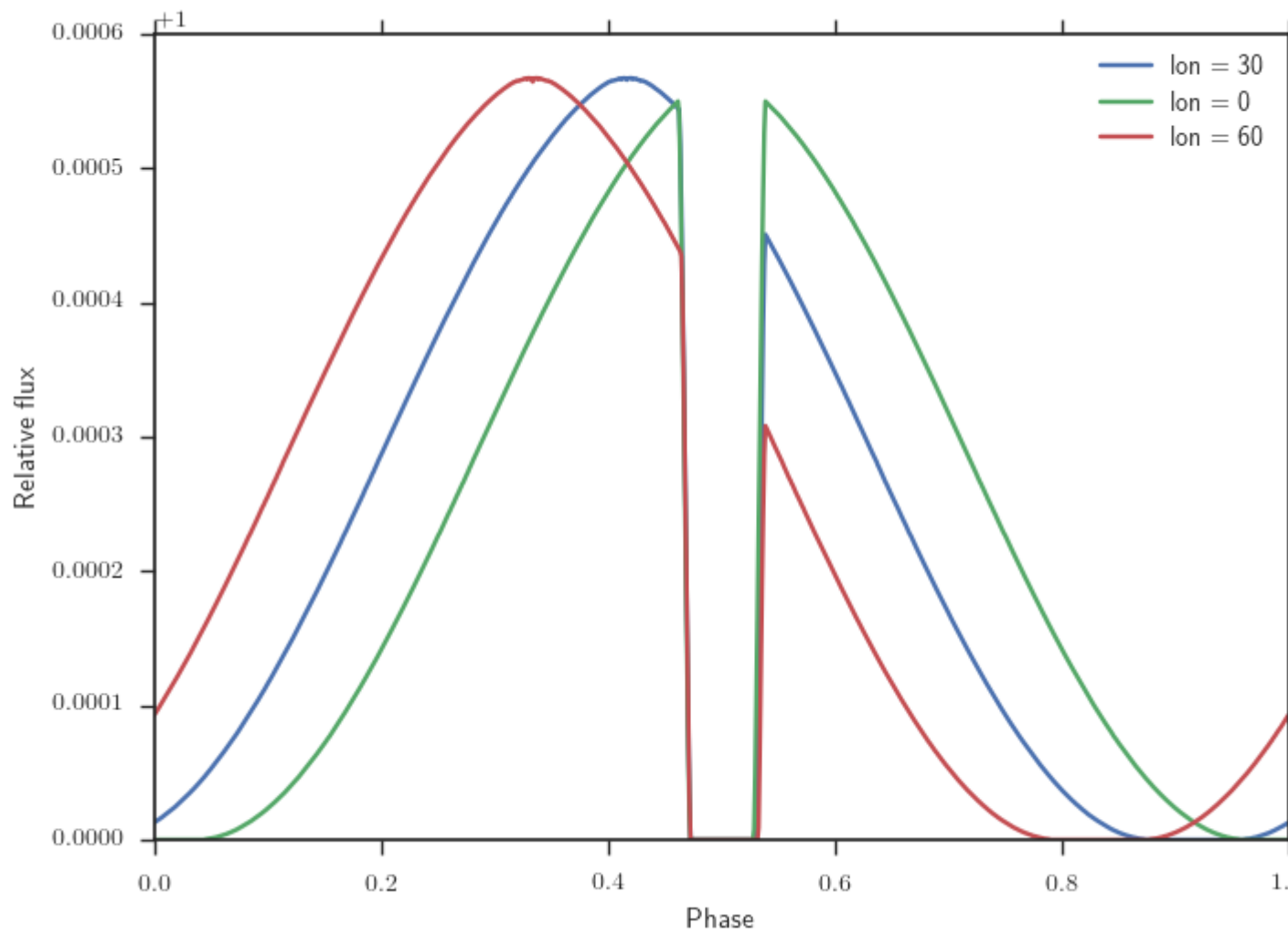
pb_n The surface brightness of the planet nightside as a fraction of the surface brightness of the star. This value will depend strongly on the physics of heat transport in the planets atmosphere and may be several orders of magnitude fainter than the spot (unitless)

Or as “hotspot_t” to specify in terms of brightness temperature, in which case the following parameters are used instead. In this case the wavelength range to integrate over must be specified.

spot_T The surface brightness of the hotspot as a fraction of the surface brightness of the star, typically of order ~ 1000 K for hot Jupiters (unit: kelvin)







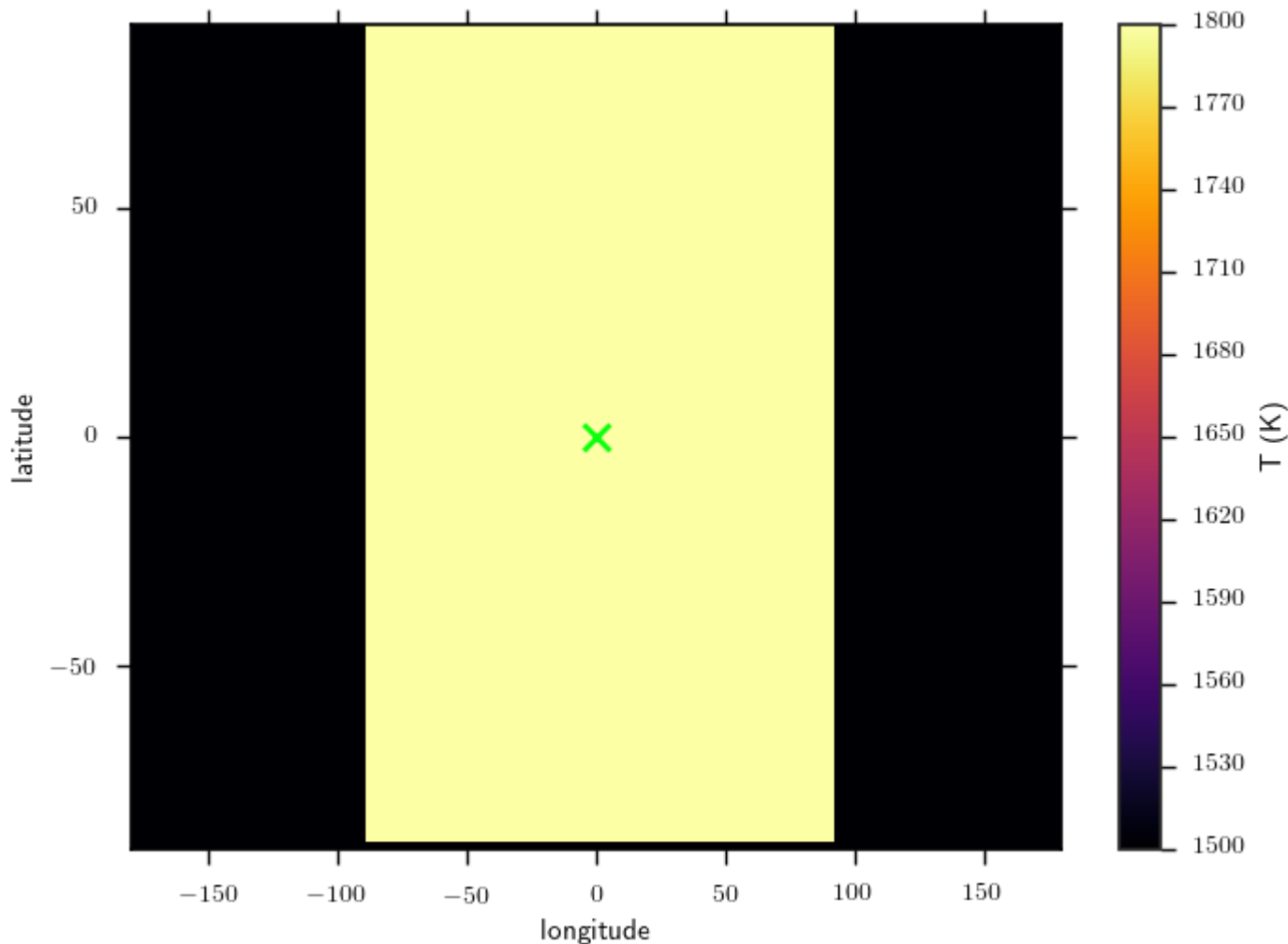
p_T The brightness temperature of the planet that is not in the hotspot. This value will depend strongly on the physics of heat transport in the planets atmosphere and may be significantly cooler than the spot (unit: degrees)

Note: Because there is a sharp contrast in flux levels between *spot* and *not spot* regions, this brightness model can have issues with quantisation, which produces unphysical “steps” in the lightcurve. This can be for the time being be solved by including a numerical integration step in regions with sharp contrasts with the optional paramter “grid_size”

cont

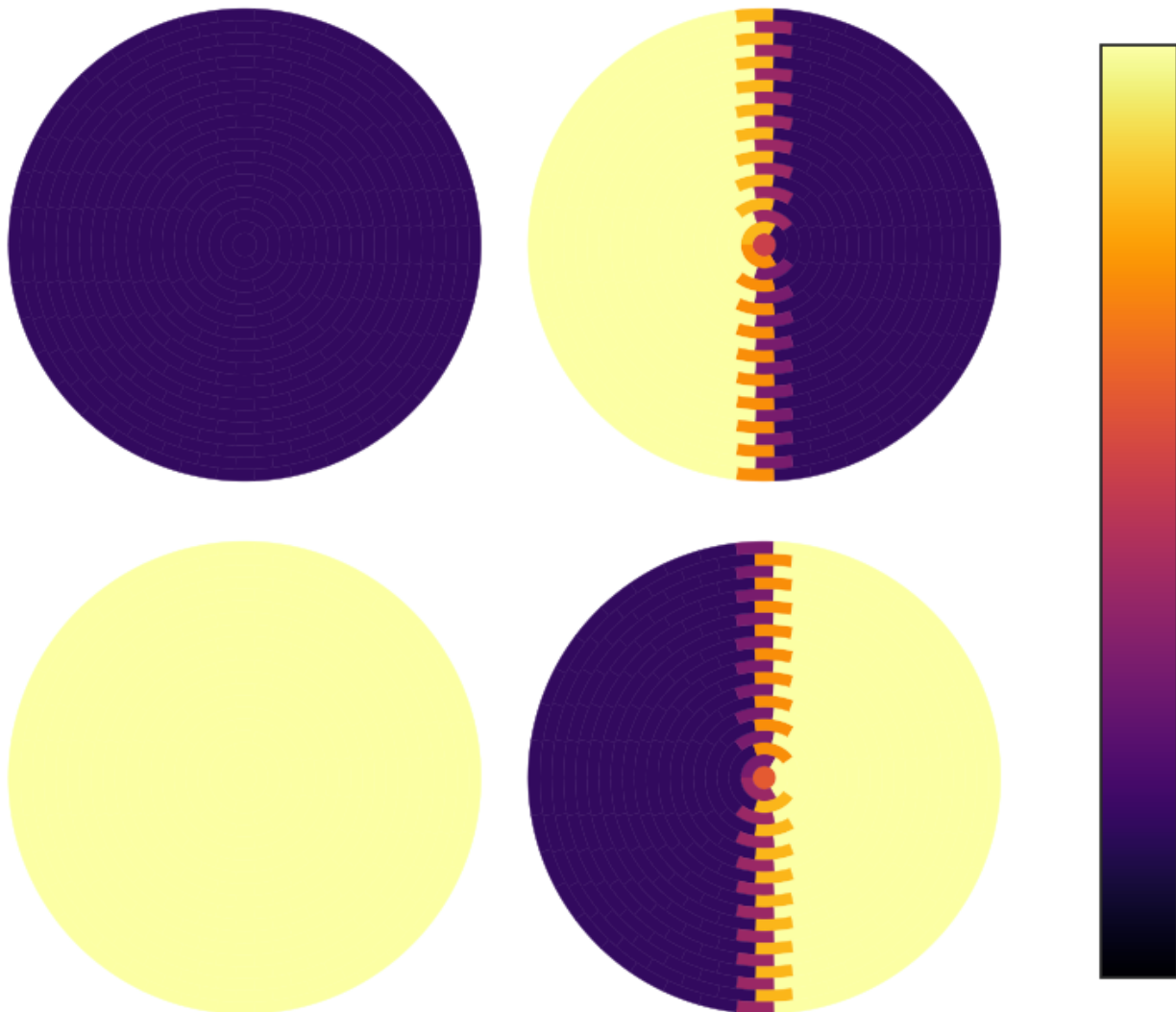
grid_size This model has a sharp boundary, so can have quantization issues. Regions with sharp changes in brightness are for now integrated numerically instead of analytically, this sets the number of grid points to use in the integration along each direction, to the total number of additional function calls will be this value squared. Setting this too high can significantly slow the code down, however if it is too low fits may be numerically unstable. Use caution. This is a temporary fix and is intended to be removed in a future version (default: 10)

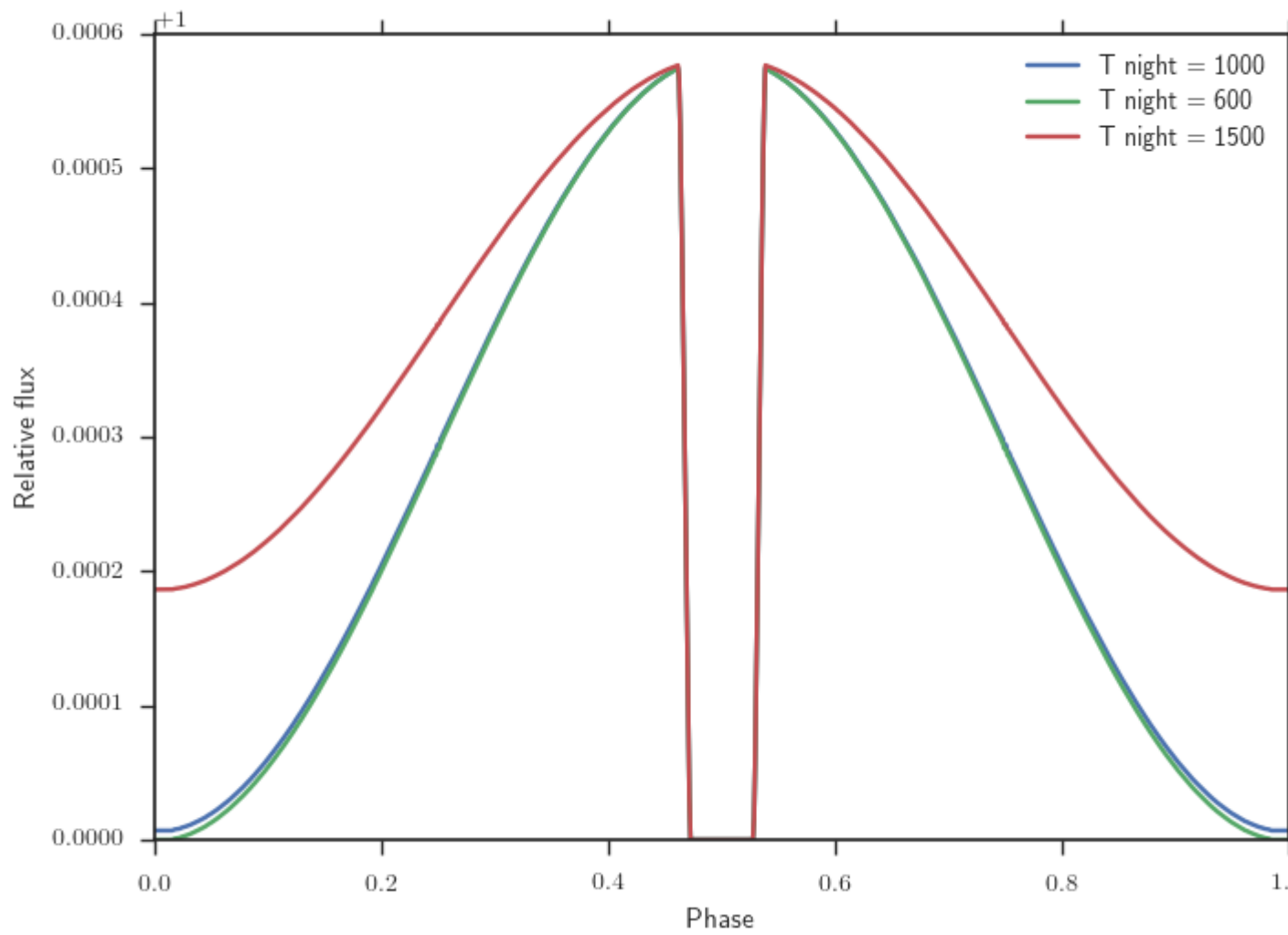
An example square plot:



An example four phase plot:

The resulting lightcurves for several parameter values





CHAPTER 5

Stellar models

In some circumstances, you might want to attempt to directly determine the absolute brightness temperature of the planet, instead of just the brightness relative to the star. Some physical models may require you to do this, for example the zhang and showman (2017) model.

To do this you need the absolute brightness of the star - don't worry, spiderman has you covered!

All models that require thermal information take the stellar effective temperature as a model parameter, and the bandpass of the observations must also be specified like so:

```
spider_params.l1 = 4520           # The stellar effective temperature in K
spider_params.l1 = 1.1e-6         # The starting wavelength in meters
spider_params.l2 = 1.7e-6         # The ending wavelength in meters
```

At the most basic level, you can assume that the spectrum of the star is simply blackbody with the effective temperature of the star. This is the default mode for spiderman. Over very wide bandpasses this will be good enough, but for narrower spectral ranges this could introduce significant errors, particularly for cooler stars with deep molecular absorption bands.

Of course, a better option would be to use realistic stellar spectra to determine the stellar flux - spiderman can do this as well, but it requires an external library of spectra, that are specified using the .spidermanrc file:

5.1 Using the spidermanrc file

a .spidermanrc file is used to give global parameters to the code. At present, the only setting needed is the location of model stellar spectra, but more customisation options may be added in future updates.

Note: Currently, the only supported model spectra are the R=10000 PHOENIX models from this page: ftp://phoenix.astro.physik.uni-goettingen.de/MedResFITS/R10000FITS/PHOENIX-ACES-AGSS-COND-2011_R10000FITS_Z-0.0.zip

Download and unpack the model spectra, then make a “.spidermanrc” file in your home directory with a line that points to this model spectra with the keyword PHOENIX_DIR, like this:

PHOENIX_DIR : /path/to/PHOENIX/

A spidermanrc file is not required - SPIDERMAN will run happily without a spidermanrc file and just calculate blackbodies for the stellar atmosphere instead.

In future releases the spidermanrc file may contain additional global parameters.

5.2 Precalculating grids of models

In order to speed up computation, spiderman will automatically generate a grid of the *summed stellar flux* in the defined bandpass as a function of stellar effective temperature, when calculating the flux of the star given its effective temperature spiderman will then use this grid to interpolate on to find the appropriate temperature, to a high level of precision.

Warning: spiderman currently only interpolates the stellar flux in 1D (Temperature) and assumes by default that the star is a dwarf with $\log g$ 4.5 - 2d interpolation with $\log g$ will be included in a future update

If you are running an MCMC, or another numerical method that requires you to call spiderman many thousands of times, especially if you are including the stellar temperature as a model parameter, you should *precalculate* this grid before beginning the model fit and pass it to spiderman to increase the efficiency. This can be done very easily with the `stellar_grid` module:

```
stellar_grid = spiderman.stellar_grid.gen_grid(l1,l2,logg=4.5)
```

Where `l1` and `l2` are the beginning and end of the spectral window in meters, and `logg` is the cgs surface gravity of the star. The `stellar_grid` object is then passed to spiderman for every light curve generation instance, e.g.

```
lc = spider_params.lightcurve(t,stellar_grid=stellar_grid)
```

If a stellar grid is not provided, spiderman will calculate it internally every time `lightcurve` is called - this will be significantly less efficient for long runs.

Instrument response

When calculating broadband phase curves, either for observations through a filter, or when summing all the data in a spectroscopic phase curve, the total instrument response may be required, as it weights how important each of the wavelengths are to the sum. Fortunately, spiderman has an easy way to account for this, simply provide a path to a “filter file”

```
spider_params.filter = 'myfilter.txt'           # The stellar effective temperature_  
↪ in K
```

This filter file must be a plain text file that consists of two columns, the wavelength in metres and the corresponding instrument response value (typically a number between 0 and 1). The code will then convolve the given filter function with the fluxes when calculating physical models with grids of blackbodies or stellar model spectra. Spiderman will linearly interpolate between the provided wavelength points.

If the filter function is too coarse, or if it contains a very sharply varying response then the results may not be accurate. In these cases it may be necessary to modify the “n_bb_seg” parameter in the lightcurve function, for which the default is 100.

CHAPTER 7

Acknowledgements

spiderman would not have happened without the following people:

- Laura Kreidberg was responsible for the early concept and much of the inspiration of this project.
- Xi Zhang for sharing his analytical solution to temperature distribution on a Hot Jupiter.
- The majority of this code was written during the [Kavli Summer Program in Astrophysics 2016](#), my thanks to the organisers and participants for a stimulating work environment.

CHAPTER 8

Release Notes

CHAPTER 9

0.3.1 (2016/08/25)

- Updated API - more object oriented.
- Made fully compatible with BATMAN.
- Added intuitive plotting scripts.
- Made example ipython notebooks.

CHAPTER 10

0.2.2 (2016/08/10)

- First stable release.