robin Documentation

Release 0.1

Laura Kreidberg, Brett Morris

Apr 16, 2018

Contents

Welcome to the documentation for `robin`, a Python package for fast calculation of exoplanet transit light curves. The package supports calculation of light curves for any radially symmetric stellar limb darkening law, using the batman integration algorithm for models that cannot be quickly calculated analytically.

In typical use, `robin` can calculate a million model light curves in well under 10 minutes for any limb darkening profile.

CHAPTER 1

Contents

1.1 Installation

1.1.1 pip

You can install `robin` with pip (recommended):

```
$ pip install robin-package
```

1.1.2 From source

You can also install directly from source. The most current stable release is up on PyPI and the development version can be found on GitHub.

Unpack the distribution with `tar -xvf` and navigate to the source root directory. To install, run the setup script:

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

Note that you'll need to `cd` out of the source directory **before** you can import `robin`.

1.1.3 On Windows

Some intrepid users have braved the path of installing batman on Windows. Their advice is available on the issue tracker.

1.1.4 Tests

To check whether the install is working, I recommend running a few basic tests with:

```
$ python -c 'import robin; robin.test()'
```

1.2 Quickstart

If you're in a hurry to model some awesome transits, you're on the right page. Here's an example of basic `robin` usage to calculate a model light curve with quadratic limb darkening. (For more detailed examples, check out the *Tutorial*.)

First, we import `robin` and a few of the usual packages:

```
import robin
import numpy as np
import matplotlib.pyplot as plt
```

Next we create a `TransitParams` object to store the physical parameters describing the transit:

```
params = robin.TransitParams()
params.t0 = 0.                          #time of inferior conjunction
params.per = 1.                         #orbital period
params.rp = 0.1                         #planet radius (in units of stellar radii)
params.a = 15.                          #semi-major axis (in units of stellar radii)
params.inc = 87.                        #orbital inclination (in degrees)
params.ecc = 0.                         #eccentricity
params.w = 90.                          #longitude of periastron (in degrees)
params.u = [0.1, 0.3]                   #limb darkening coefficients [u1, u2]
params.limb_dark = "quadratic"          #limb darkening model
```

Note that for circular orbits, `robin` uses the convention `params.w = 90`. The units for `params.t0` and `params.per` can be anything as long as they are consistent.

We also need to specify the times at which we wish to calculate the model:

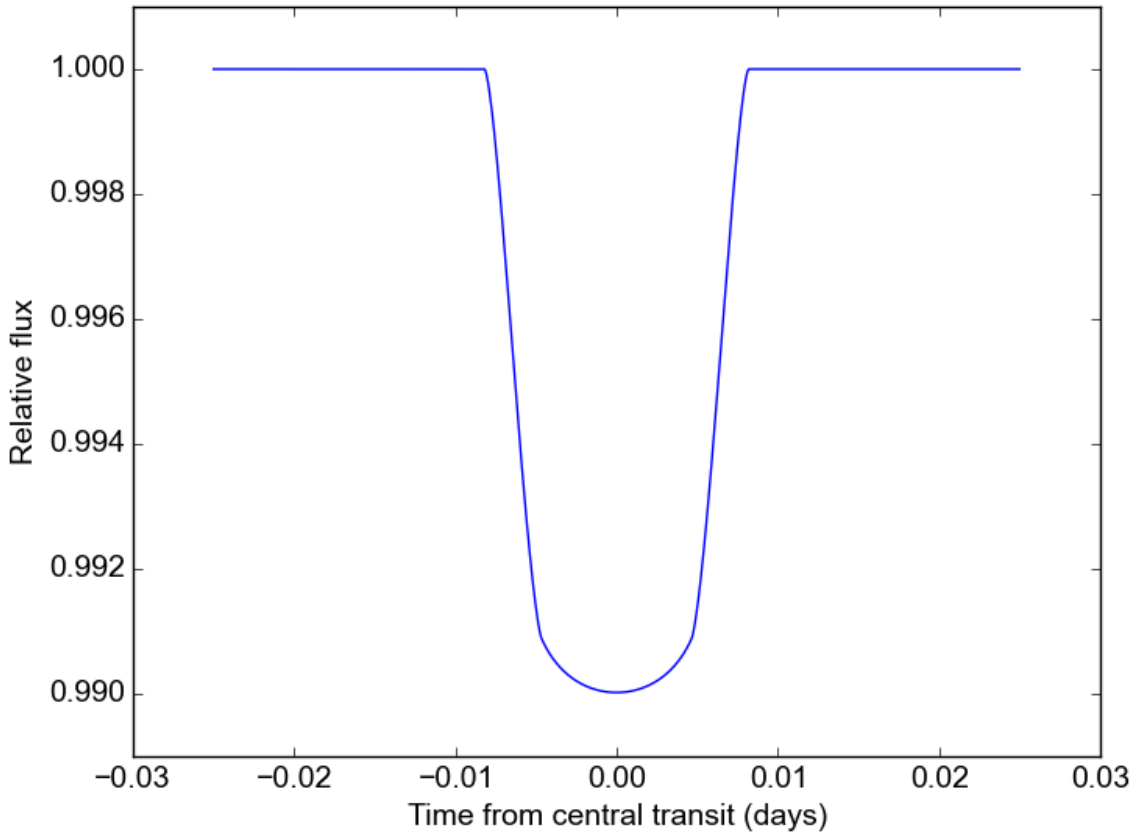
```
t = np.linspace(-0.05, 0.05, 100)
```

Using these parameters, we initialize the model and calculate a model light curve:

```
m = robin.TransitModel(params, t)    #initializes model
flux = m.light_curve(params)          #calculates light curve
```

Voilà! Here's a figure showing the light curves:

```
plt.plot(t, flux)
plt.xlabel("Time from central transit")
plt.ylabel("Relative flux")
plt.show()
```



This code is available in full at https://github.com/bmorris3/robin/tree/master/docs/quickstart.py.

1.3 Tutorial

In this tutorial, we'll go through `robin`'s functionality in more detail than in the *Quickstart*. First let's initialize a model with nonlinear limb darkening:

1.3.1 Initializing the model

```
import robin
import numpy as np
import matplotlib as plt

params = robin.TransitParams()        #object to store transit parameters
params.t0 = 0.                        #time of inferior conjunction
params.per = 1.                       #orbital period
params.rp = 0.1                       #planet radius (in units of stellar radii)
params.a = 15.                        #semi-major axis (in units of stellar radii)
params.inc = 87.                      #orbital inclination (in degrees)
params.ecc = 0.                       #eccentricity
params.w = 90.                        #longitude of periastron (in degrees)
params.limb_dark = "nonlinear"        #limb darkening model
```

```
params.u = [0.5, 0.1, 0.1, -0.1]      #limb darkening coefficients [u1, u2, u3, u4]

t = np.linspace(-0.025, 0.025, 1000)  #times at which to calculate light curve
m = robin.TransitModel(params, t)    #initializes model
```

The initialization step calculates the separation of centers between the star and the planet, as well as the integration step size (for "square-root", "logarithmic", "exponential", "nonlinear", "power2", and "custom" limb darkening).

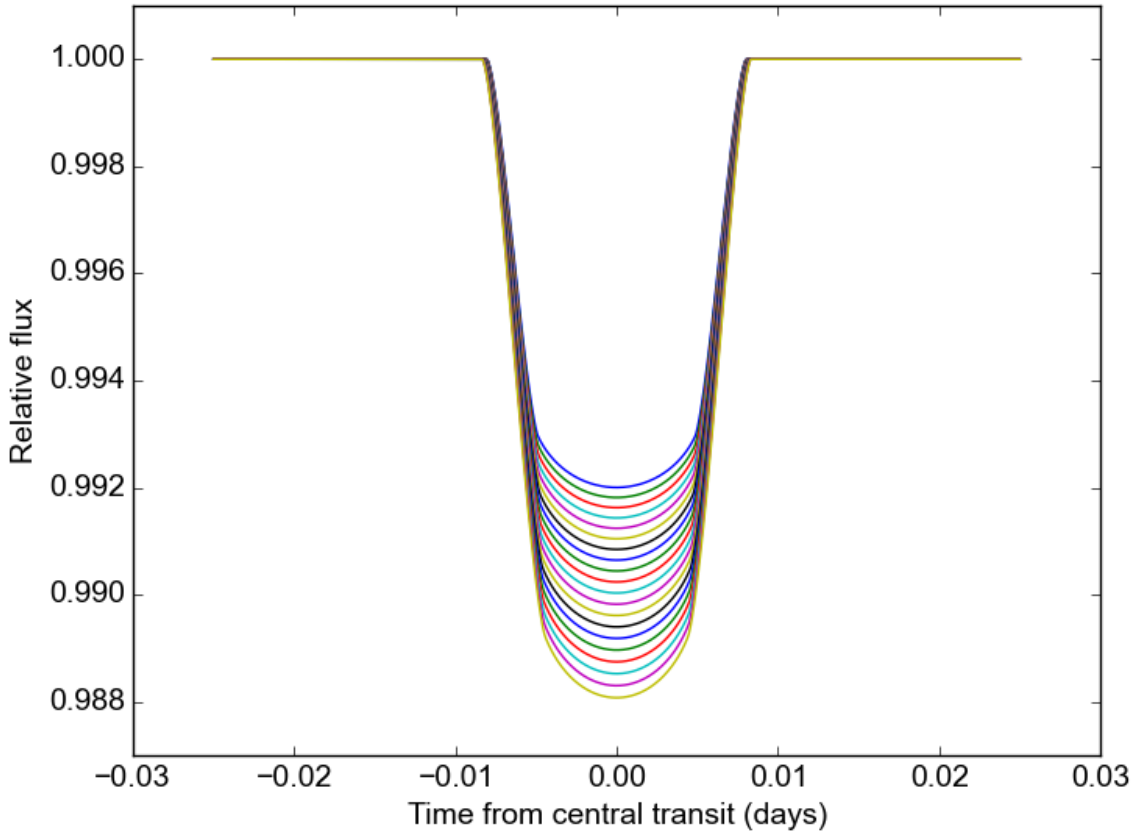
1.3.2 Calculating light curves

To make a model light curve, we use the `light_curve` method:

```
flux = m.light_curve(params)                    #calculates light curve
```

Now that the model has been set up, we can change the transit parameters and recalculate the light curve **without** reinitializing the model. For example, we can make light curves for a range of planet radii like so:

```
radii = np.linspace(0.09, 0.11, 20)
for r in radii:
        params.rp = r                           #updates planet radius
        new_flux = m.light_curve(params)        #recalculates light curve
```



1.3.3 Limb darkening options

The `robin` package currently supports the following pre-defined limb darkening options: "uniform", "linear", "quadratic", "square-root", "logarithmic", "exponential", "power2" (from Morello et al. 2017), and "nonlinear". These options assume the following form for the stellar intensity profile:

$$
\begin{aligned}
I(\mu) &= I_0 && \text{(uniform)} && (1.1)\\
I(\mu) &= I_0[1 - c_1(1-\mu)] && \text{(linear)} && (1.2)\\
I(\mu) &= I_0[1 - c_1(1-\mu) - c_2(1-\mu)^2] && \text{(quadratic)} && (1.3)\\
I(\mu) &= I_0[1 - c_1(1-\mu) - c_2(1-\sqrt{\mu})] && \text{(square-root)} && (1.4)\\
I(\mu) &= I_0[1 - c_1(1-\mu) - c_2\mu\ln\mu] && \text{(logarithmic)} && (1.5)\\
I(\mu) &= I_0\left[1 - c_1(1-\mu) - c_2/(1-\exp\mu)\right] && \text{(exponential)} && (1.6)\\
I(\mu) &= I_0\left[1 - c_1(1-\mu_2^c)\right] && \text{(power2)} && (1.7)\\
I(\mu) &= I_0[1 - c_1(1-\mu^{1/2}) - c_2(1-\mu) - c_3(1-\mu^{3/2}) - c_4(1-\mu^2)] && \text{(nonlinear)} && (1.8)
\end{aligned}
$$

where $\mu = \sqrt{1-x^2}, 0 \leq x \leq 1$ is the normalized radial coordinate and I_0 is a normalization constant such that the integrated stellar intensity is unity.

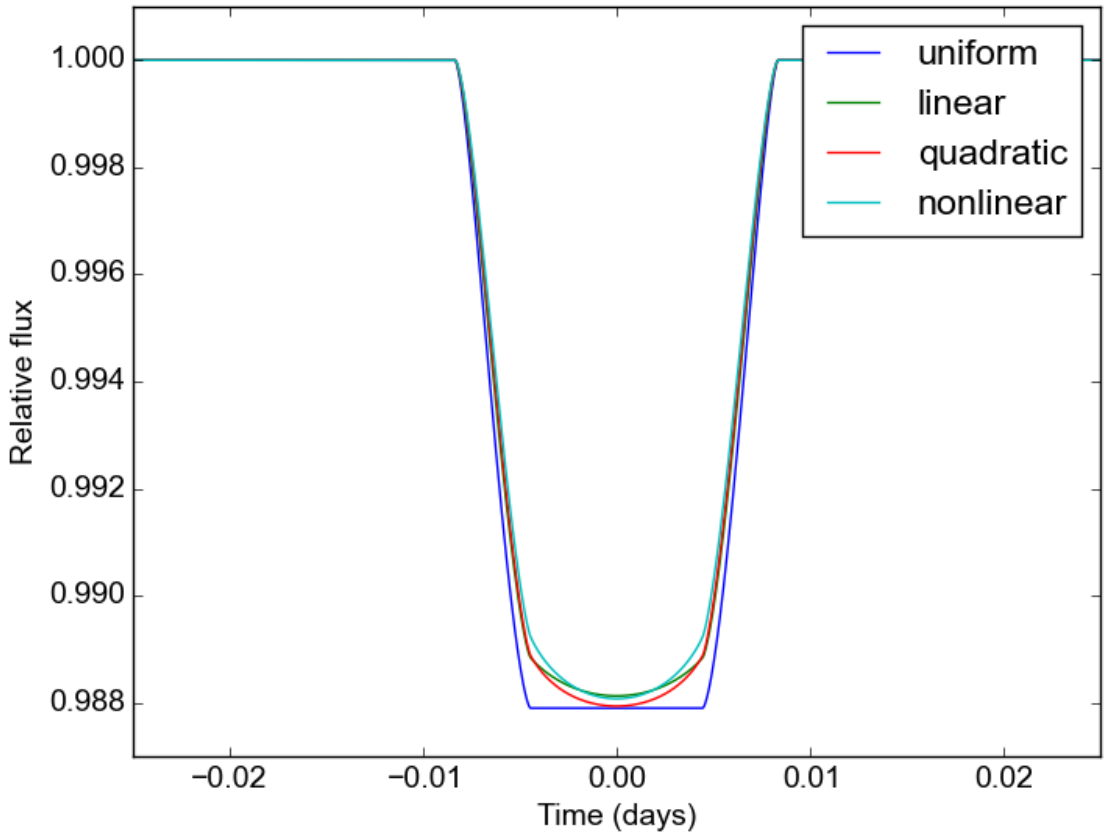
To illustrate the usage for these different options, here's a calculation of light curves for the four most common profiles:

```
ld_options = ["uniform", "linear", "quadratic", "nonlinear"]
ld_coefficients = [[], [0.3], [0.1, 0.3], [0.5, 0.1, 0.1, -0.1]]

plt.figure()

for i in range(4):
        params.limb_dark = ld_options[i]          #specifies the LD profile
        params.u = ld_coefficients[i]             #updates LD coefficients
        m = robin.TransitModel(params, t)         #initializes the model
        flux = m.light_curve(params)              #calculates light curve
        plt.plot(t, flux, label = ld_options[i])
```

The limb darkening coefficients are provided as a list of the form $[c_1, ..., c_n]$ where n depends on the limb darkening model.



1.3.4 Custom limb darkening

`robin` also supports the definition of custom limb darkening. To create a custom limb darkening law, you'll have to write a very wee bit of C code and perform a new installation of `robin`.

First, download the package from source at https://pypi.python.org/pypi/robin-package/. Unpack the files and `cd` to the root directory.

To define your stellar intensity profile, edit the `intensity` function in the file `c_src/_custom_intensity.c`. This function returns the intensity at a given radial value, $I(x)$. Its arguments are x (the normalized radial coordinate; $0 \leq x \leq 1$) and six limb darkening coefficients, $c_1, ..., c_6$.

The code provides an example intensity profile to work from:

$$I(x) = I_0 \left[1 - c_1(1 - \sqrt{1 - x^2}) - c_2 \ln\left(\frac{\sqrt{1 - x^2} + c_3}{1 + c_3}\right)\right]$$

(N.B.: This profile provides a better fit to stellar models than the quadratic law, but uses fewer coefficients than the nonlinear law. Thanks to Eric Agol for suggesting it!).

This function can be replaced with another of your choosing, so long as it satistifies the following conditions:

- The integrated intensity over the stellar disk must be unity:

$$\int_0^{2\pi} \int_0^1 I(x) x dx d\theta = 1$$

- The intensity must also be defined on the interval $0 \leq x \leq 1$. To avoid issues relating to numerical stability, I would recommend including:

```
if(x < 0.00005) x = 0.00005;
if(x > 0.99995) x = 0.99995;
```

To re-install `robin` with your custom limb darkening law, run the setup script:

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

You'll have to `cd` out of the source root directory to successfully import `robin`. Now, to calculate a model light curve with your custom limb darkening profile, use:

```
params.limb_dark = "custom"
params.u = [c1, c2, c3, c4, c5, c6]
```

with any unused limb darkening coefficients set equal to 0.

And that's it!

1.3.5 Error tolerance

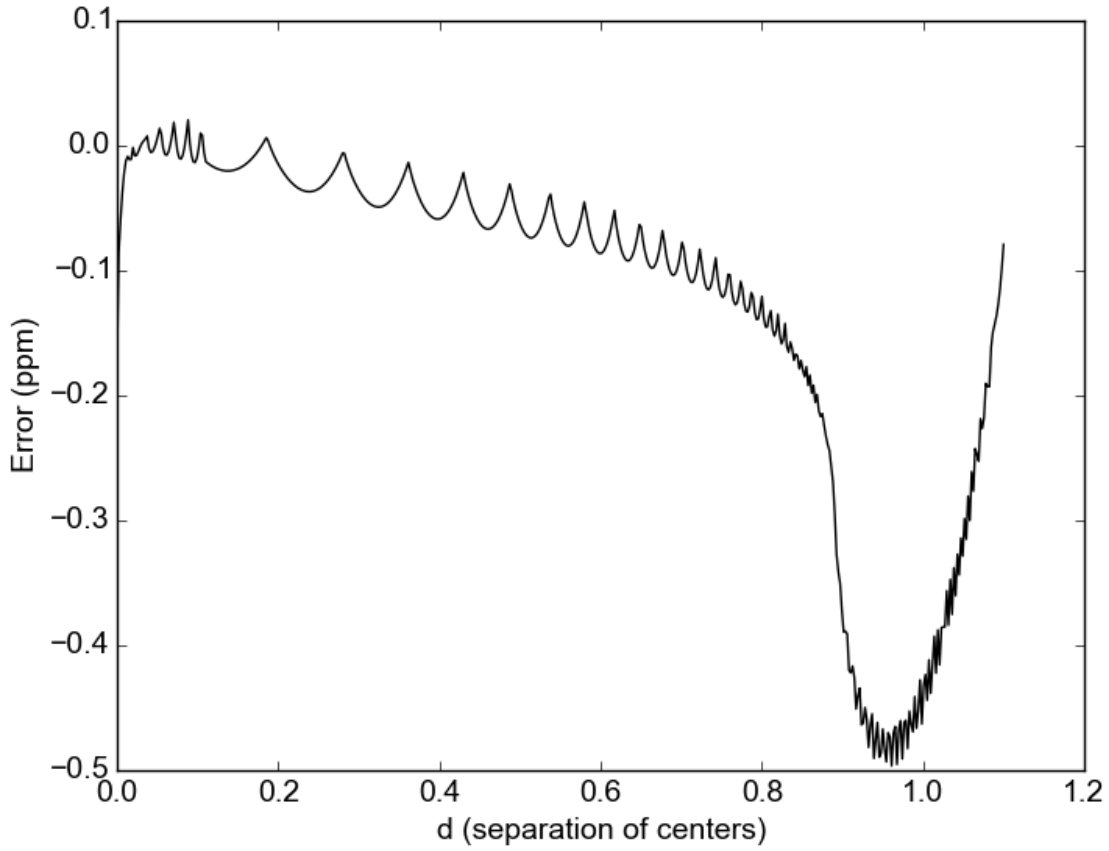
For models calculated with numeric integration ("square-root", "logarithmic", "exponential", "nonlinear" and "custom" profiles), we can specify the maximum allowed truncation error with the `max_err` parameter:

```
m = robin.TransitModel(params, t, max_err = 0.5)
```

This initializes a model with a step size tuned to yield a maximum truncation error of 0.5 ppm. The default `max_err` is 1 ppm, but you may wish to adjust it depending on the combination of speed/accuracy you require. Changing the value of `max_err` will not impact the output for the analytic models ("quadratic", "linear", and "uniform").

To validate that the errors are indeed below the `max_err` threshold, we can use `m.calc_err()`. This function returns the maximum error (in ppm) over the full range of separation of centers d ($0 \leq d \leq 1$, in units of rs). It also has the option to plot the truncation error over this range:

```
err = m.calc_err(plot = True)
```



Truncation error is larger near $d = 1$ because the stellar intensity has a larger gradient near the limb.

If you prefer not to calculate the step size automatically, it can be set explicitly with the `fac` parameter; this saves time during the model initialization step.

1.3.6 Parallelization

The default behavior for `robin` is no parallelization. If you want to speed up the calculation, you can parallelize it by setting the `nthreads` parameter. For example, to use 4 processors you would initialize a model with:

```
m = robin.TransitModel(params, t, nthreads = 4)
```

The parallelization is done at the C level with OpenMP. If your default C compiler does not support OpenMP, `robin` will raise an exception if you specify `nthreads`>1.

Note: Mac users: the C default compiler (clang) does not currently (06/2015) support OpenMP. To use a different compiler, perform a fresh install with the "CC" and "CXX" environment variables set inside "setup.py" with os.environ.

1.3.7 Modeling eclipses

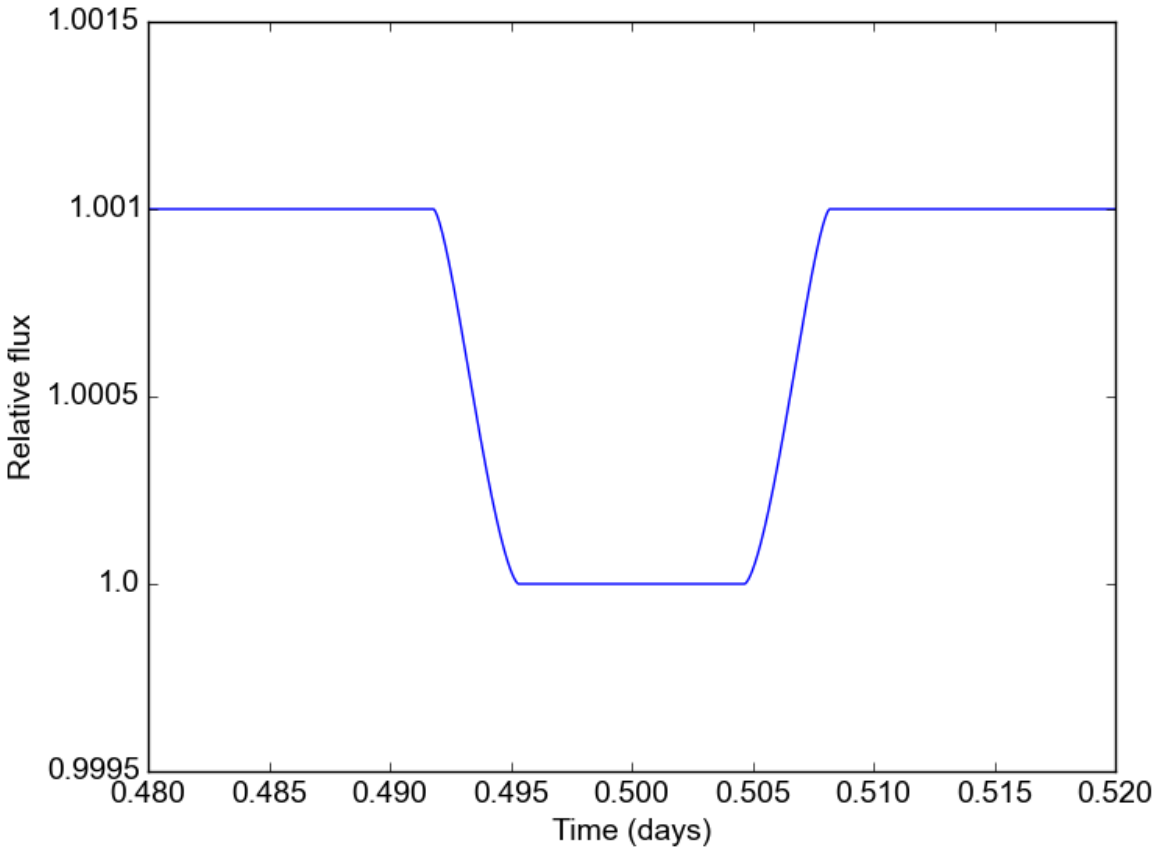
`robin` can also model eclipses (secondary transits). To do this, specify the planet-to-star flux ratio and the central

eclipse time:

```
params.fp = 0.001
params.t_secondary = 0.5
```

and initialize a model with the `transittype` parameter set to `"secondary"`:

```
t = np.linspace(0.48, 0.52, 1000)
m = robin.TransitModel(params, t, transittype="secondary")
flux = m.light_curve(params)
plt.plot(t, flux)
```



The eclipse light curve is normalized such that the flux of the star is unity. The eclipse depth is f_p. The model assumes that the eclipse center occurs when the true anomaly equals $3\pi/2 - \omega$.

For convenience, *robin* includes utilities to calculate the time of secondary eclipse from the time of inferior conjunction, and vice versa. See the `get_t_secondary` and `get_t_conjunction` methods in the API.

Warning: Note that the secondary eclipse calculation does NOT account for the light travel time in the system (which is of order minutes). Future versions of `robin` may include this feature, but for now you're on your own!

1.3.8 Supersampling

For long exposure times, you may wish to calculate the average value of the light curve over the entire exposure. To do this, initialize a model with the `supersample_factor` and `exp_time` parameters specified:

```
m = robin.TransitModel(params, t, supersample_factor = 7, exp_time = 0.001)
```

This example will return the average value of the light curve calculated from 7 evenly spaced samples over the duration of each 0.001-day exposure. The `exp_time` parameter must have the same units as the array of observation times `t`.

1.4 API

class robin.**TransitParams**

Object to store the physical parameters of the transit.

Parameters

- **t0** (*float, optional*) – Time of inferior conjunction.
- **t_secondary** (*float, optional*) – Time of secondary eclipse center.
- **per** (*float*) – Orbital period.
- **rp** (*float*) – Planet radius [in stellar radii].
- **a** (*float*) – Semi-major axis [in stellar radii].
- **inc** (*float*) – Orbital inclination [in degrees].
- **ecc** (*float*) – Orbital eccentricity.
- **w** (*float*) – Argument of periapse [in degrees]
- **u** (*array_like*) – List of limb darkening coefficients.
- **limb_dark** (*str*) – Limb darkening model (choice of "nonlinear", "quadratic", "exponential", "logarithmic", "squareroot", "linear", "uniform", "power2", or "custom").
- **fp** (*float, optional*) – Planet-to-star flux ratio (for secondary eclipse models).

Note:

- Units for the orbital period and ephemeris can be anything as long as they are consistent (e.g. both in days).
- The orbital path is calculated based on *t0* for primary transits and *t_secondary* for secondary eclipses.

Example

```
>>> import robin
>>> params = robin.TransitParams()
>>> params.t0 = 0.                                    #time of inferior conjunction
>>> params.per = 1.                                   #orbital period
>>> params.rp = 0.1                                   #planet radius (in units of␣
↪stellar radii)
>>> params.a = 15.                                    #semi-major axis (in units of␣
↪stellar radii)
>>> params.inc = 87.                                  #orbital inclination (in degrees)␣
↪
```

```
>>> params.ecc = 0.                                    #eccentricity
>>> params.w = 90.                                     #longitude of periastron (in␣
↪degrees)
>>> params.u = [0.1, 0.3]                              #limb darkening coefficients
>>> params.limb_dark = "quadratic"                     #limb darkening model
```

class robin.**TransitModel**(*params*, *t*, *max_err=1.0*, *nthreads=1*, *fac=None*, *transittype='primary'*, *supersample_factor=1*, *exp_time=0.0*)

Class for generating model transit light curves.

Parameters

- **params** (a *TransitParams* instance) – A *TransitParams* object containing the physical parameters of the transit
- **t** (*ndarray*) – Array of times at which to calculate the model.
- **max_err** (*float, optional*) – Error tolerance (in parts per million) for the model.
- **nthreads** (*int, optional*) – Number of threads to use for parallelization.
- **fac** (*float, optional*) – Scale factor for integration step size
- **transittype** (*string, optional*) – Type of transit ("primary" or "secondary")
- **supersample_factor** (*integer, optional*) – Number of points subdividing exposure
- **exp_time** (*double, optional*) – Exposure time (in same units as *t*)

Example

```
>>> m = robin.TransitModel(params, max_err = 0.5, nthreads=4)
```

calc_err(*plot=False*)

Calculate maximum error for transit light curve calculation.

Parameters **plot** (*bool*) – If True, plots the error in the light curve model as a function of separation of centers.

Returns Truncation error (parts per million)

Return type float

get_t_conjunction(*params*)

Return the time of primary transit center (calculated using *params.t_secondary*).

get_t_periastron(*params*)

Return the time of periastron passage (calculated using *params.t0*).

get_t_secondary(*params*)

Return the time of secondary eclipse center (calculated using *params.t0*).

get_true_anomaly()

Return the true anomaly at each time

light_curve(*params*)

Calculate a model light curve.

Parameters **params** (A *TransitParams* instance) – Transit parameters

Returns Relative flux

Return type ndarray

Example

```
>>> flux = m.light_curve(params)
```

1.5 Troubleshooting

1.5.1 Help! `robin` is running really slowly - why is this?

My first guess is that you're reinitializing the model many times. This is by far the slowest component of `robin`, because it calculates the optimal step size for the integration starting from a very small value. To speed this up, you can manually set the scale factor for the step size. First check what the optimal step size factor is after you initalize a model with realistic transit parameters:

:: m = robin.TransitModel(params, t)

fac = m.fac print("stepsize:", fac)

```
>>> stepsize: 0.023
```

Then, you can use set this step size as the default, like so:

:: m = robin.TransitModel(params, t, fac = fac)

This will save A LOT of time! The optimal step size will vary slightly depending on the limb darkening and the planet radius, so I would recommend choosing a conservative value for the error tolerance to account for this.

1.5.2 My light curve has nans in it!

Check and see if your transit parameters are physically realistic. In particular, confirm that the planet's orbit does not pass through the star. (You think this would never happen to you, but don't be too sure! I've been asked about it more than once.)

1.6 Acknowledgements

Many thanks to the following individuals for their support in the development of `robin`:

- Jacob Bean
- Kevin Stevenson
- Ethan Kruse
- Geert Jan Talens
- Thomas Beatty
- Brett Morris
- Eric Agol
- Karl Fogel
- Jason Eastman
- Ian Crossfield
- Mark Omohundro

- Michael Hippke
- Hannu Parviainen
- Michael Zhang
- Andrew Mayo
- Heather Cegla

Python Module Index

r

Index