PyTransit

Release 2.0

Oct 21, 2020

Contents

Welcome to PyTransit documentation! PyTransit is a package for exoplanet transit light curve modelling that offers optimised CPU and GPU implementations of exoplanet transit models with a unified interface. Transit model evaluation is trivial for simple use-cases, such as homogeneous light curves observed in a single passband, but also straightforward for more complex use-cases, such as when dealing with *heterogeneous light curves containing transits observed in different passbands and different instruments*, or with *transmission spectroscopy*.

The development of PyTransit began in 2009 to fill the need for a fast and reliable exoplanet transit modelling toolkit for Python. Since then, PyTransit has gone through several iterations, always thriving to be *the fastest and most versatile* exoplanet transit modelling tool for Python.

PyTransit v1.0 was described in Parviainen (2015), which also details the model-specific optimisations and model performance. This version relied heavily on Fortran code, which made the installation complicated in non-Linux systems. PyTransit v2 replaces all the Fortran routines with numba-accelerated python routines, and aims to implement all the major functionality also in OpenCL.

While PyTransit is aimed to work as a library offering tools for customised transit analysis codes, it can also be used directly for transit modelling and parameter estimation.

CHAPTER 1

Example

The transit model initialization is straightforward. At its simplest, the model takes an array the mid-exposure times,

```
from pytransit import QuadraticModel

tm = QuadraticModel()
tm.set_data(times)
```

after which it is ready to be evaluated

```
tm.evaluate(k=0.1, ldc=[0.2, 0.1], t0=0.0, p=1.0, a=3.0, i=0.5*pi)
```

To complicate the situation a bit, we can consider a case where we want to model several transits observed in different passbands. The stellar limb darkening varies from passband to passband, so we need to give a set of limb darkening coefficients for each passband, and we may also want to allow the radius ratio to vary from passband to passband. Now, we will only need to initialise the model with per-exposure light curve indices (*lcids*) and per-light-curve passband indices (*pbids*) (don't worry, these are simple integer arrays), after which we are ready to evaluate the model with passband-dependent radius ratio and limb darkening

```
tm.set_data(times, lcids=lcids, pbids=pbids)
tm.evaluate(k=[0.10, 0.12], ldc=[[0.2, 0.1, 0.5, 0.1]], t0=0.0, p=1.0, a=3.0, i=0.
↪5*pi)
```

We made both the radius ratio and limb darkening passband-dependent in the example above, but we could just as well evaluate the model with a single scalar radius ratio (as in the first example), in which case only the limb darkening would be passband-dependent.

We may often want to evaluate the model for a large set of parameters at the same time (such as when doing MCMC sampling with *emcee*, or using some other population-based sampling or minimization method). Give *evaluate* an array of parameters

```
tm.evaluate(k=[[0.10, 0.12], [0.11, 0.13]],
            ldc=[[0.2, 0.1, 0.5, 0.1],[0.4, 0.2, 0.75, 0.1]],
            t0=[0.0, 0.01], p=[1, 1], a=[3.0, 2.9], i=[.5*pi, .5*pi])
```

and PyTransit will calculate the models for the whole parameter set in parallel.

All the models come in CPU and GPU (OpenCL) versions. OpenCL versions can be 10-20 times faster to evaluate than the CPU versions (depending on the GPU), and switching to use the GPU is as simple as just importing an *CL* version of the model, which will work identically to the CPU version

```
from pytransit import QuadraticModelCL
```

The examples use a transit model with quadratic limb darkening by Mandel & Agol, but all the models follow the same API, so when you learn to use one, you can use all of them. Finally, these examples show just the use of the main *evaluate* method, but the models have also more optimized evaluation methods for specific use-cases, and the package comes with utilities for specialized analyses.

CHAPTER 2

Contents

2.1 Installation

Pytransit can be installed from PyPI

```
pip install pytransit
```

or by cloning the repository from GitHub and running the setup script

```
git clone https://github.com/hpparvi/PyTransit.git
cd PyTransit
python setup.py install
```

2.2 Quickstart

2.2.1 Basic transit model evaluation

PyTransit comes with a set of transit models that share a common interface (with small model-specific variations). So, while we use a Mandel & Agol quadratic limb darkening model (`pytransit.QuadraticModel`) as an example here, the evaluation works the same for all the models.

First, the transit model needs to be imported from PyTransit. After this, it can be initialised, and set up by giving it (at least) a set of mid-exposure times.

```
from pytransit import QuadraticModel

tm = QuadraticModel()
tm.set_data(time)
```

where *time* is a NumPy array (or a list or a tuple) of mid-exposure times for which the model will be evaluated.

After the initialisation and setup, the transit model can be evaluated as

```
flux = tm.evaluate(k, ldc, t0, p, a, i, e, w)
```

where *k* is the planet-star radius ratio, *t0* is the zero epoch, *p* is the orbital period, *a* is the scaled semi-major axis, *i* is the inclination, *e* is the eccentricity, *w* is the argument of periastron, and *ldc* is an *ndarray* containing the model-specific limb darkening coefficients.

The calling simplifies further if we assume a circular orbit, when we can leave *e* and *w* out

```
flux = tm.evaluate(k, ldc, t0, p, a, i)
```

The radius ratio can either be a scalar, a 1D vector, or a 2D array, the limb darkening coefficients are given as a 1D vector or a 2D array, and the orbital parameters (*t0*, *p*, *a*, *i*, *e*, and *w*) can be either scalars or vectors.

In the most simple case the limb darkening coefficients are given as a single vector and the rest of the parameters are scalars, in which case the *flux* array will also be one dimensional. However, if we want to evaluate the model for multiple parameter values (such as when using *emcee* for MCMC sampling), giving a 2D array of limb darkening coefficients and the rest of the parameters as vectors allows PyTransit to evaluate the models in parallel, which can lead to significant performance improvements (especially with the OpenCL versions of the transit models). Evaluating the model for *n* sets of parameters will result in a *flux* array with a shape *(n, time.size)*.

A third case, giving a 2D array of radius ratios (or a 1D vector of radius ratios when the orbital parameters are scalars), is slightly more advanced, and is used when modelling multicolor photometry (or transmission spectroscopy). In this case the model assumes the radius ratio varies from passband to passband, and the setup requires also passband indices (see later).

2.3 Transit Models

PyTransit implements five of the most important exoplanet transit light curve models, each with model-specific optimisations to make their evaluation efficient. The models come in two flavours

- Numba-accelerated implementations for CPU computing. These implementations are multi-threaded, and can be the best choice when modelling large amounts of short-cadence observations where the data transfer between the GPU and main memory would create a bottleneck.
- OpenCL implementations for GPU computing. These can be orders of magnitude faster than the CPU implementations if ran in a powerful GPU, especially when modelling long cadence data where the amount of computation per observation dominates over the time for data transfer.

The CPU and GPU implementations aim to offer equal functionality, but, at the moment of writing, they have some variation in the available features.

2.3.1 Transit model interface

The transit models share a unified interface with small variations to account for model-specific parameters and settings. Some of the models have also special evaluation methods aimed for specific science cases, such as transmission spectroscopy where the light curves have been created from a spectroscopic time series.

The models are made to work with heterogeneous photometric time series. That is, a single model evaluation can model observations in different passbands (with different limb darkening), different exposure times, and different supersampling rates.

2.3.2 Model initialisation

Model initialisation is straightforward. The Mandel-Agol model with quadratic limb darkening can be imported and initialised by

```
from pytransit import QuadraticModel

tm = QuadraticModel()
```

After the initialisation, the model still needs to be set up by giving it the observation centre-times, and optionally other information, such as passbands, exposure times, supersampling rates, etc.

2.3.3 Data setup

Basics

At its simplest, the data setup requires the mid-observation times. If no other other information is given, the model assumes that all the data have been observed in a single passband and that the exposure time is short enough so that supersampling is not needed.

```
tm.set_data(time)
```

Heterogeneous light curves

PyTransit can be used to model heterogeneous time series. That is, the time array can consist of many transit light curves observed in different passbands and with different exposure times (requiring different supersampling rates). For this to work, the model first needs to assign each individual exposure to a single *light curve*. This is done by passing the model an integer array of light curve indices (*lcids*), where each element maps an exposure to a light curve.

```
tm.set_data(time=[0,1,2,3,4], lcids=[0,0,0,1,1])
```

Just setting the light curve indices doesn't do anything by itself, but it is necessary to use the more advanced features described below.

The model doesn't need to be told explicitly how many light curves the dataset contains, since the number of light curves is obtained from the unique *lcids* elements.

Multiple passbands

PyTransit can model transits observed in multiple passbands, where each passband has a different stellar limb darkening profile. For this, the model needs to be given an integer array of passband indices (*pbids*), where *each element maps a light curve to a single passband*. Expanding the previous example, we can tell the model that the two light curves belong to different passbands as

```
tm.set_data(time=[0,1,2,3,4], lcids=[0,0,0,1,1], pbids=[0,1])
```

After this, the model expects to get a two-dimensional array of limb darkening coefficients when evaluated, as explained later in more detail.

Supersampling

If the exposure time is long (Kepler and TESS long cadence mode, for example), supersampling can be set up by giving the exposure time (*exptime*) and supersampling rate (*nsamples*), where *exptime* and *nsamples* are either floats or arrays.

A single float can be given when modelling a homogeneous time series

```
tm.set_data(time, exptime=0.02, nsamples=10)
```

in which case the whole time series will have a constant supersampling rate. An array of per-light-curve values can be given when modelling heterogeneous time series

```
tm.set_data(time=[0,1,2,3,4], lcids=[0,0,0,1,1], exptime=[0.0007, 0.02], nsamples=[1,␣
↪10])
```

in which case each light curve will have a separate supersampling rate.

Advanced example

For a slightly more advanced example, a set of three light curves, two observed in one passband and the third in another passband, with times

```
times_1 (lc = 0, pb = 0, sc) = [1, 2, 3, 4]
times_2 (lc = 1, pb = 0, lc) = [3, 4]
times_3 (lc = 2, pb = 1, sc) = [1, 5, 6]
```

would be set up as

```
tm.set_data(time  = [1, 2, 3, 4, 3, 4, 1, 5, 6],
            lcids = [0, 0, 0, 0, 1, 1, 2, 2, 2],
            pbids = [0, 0, 1],
            nsamples = [  1,  10,   1],
            exptimes = [0.1, 1.0, 0.1])
```

2.3.4 Model evaluation

```
tm.evaluate_ps()
```

```
tm.evaluate_pv()
```

2.3.5 OpenCL

The OpenCL versions of the models work identically to the Python version, except that the OpenCL context and queue can be given as arguments in the initialiser, and the model evaluation method can be told to not to copy the model from the GPU memory. If the context and queue are not given, the model creates a default context using *cl.create_some_context()*.

```
import pyopencl as cl
from src import QuadraticModelCL

ctx = cl.create_some_context()
```

(continues on next page)

(continued from previous page)

```
queue = cl.CommandQueue(ctx)

tm = QuadraticModelCL(cl_ctx=ctx, cl_queue=queue)
```

2.4 Implemented transit models

PyTransit implements a set of transit models that all share a common interface that is described in more detail in *Transit Models*.

2.4.1 Road Runner model

RoadRunner (*pytransit.RoadRunnerModel*) is a fast and flexible transit model presented in Parviainen (accepted to MNRAS 2020). I'll write a proper documentation soon, but these example notebooks should help you up to speed until then

- Example 1: basics
- Example 2: custom limb darkening
- Example 3: LDTk limb darkening model

2.4.2 Uniform model

The uniform model (*pytransit.UniformModel* and *pytransit.UniformModelCL*) reproduces an exoplanet transit over a uniform disc. This model is useful when modelling secondary eclipses, or when the effects from the stellar limb darkening can be ignored.

- Uniform model example

2.4.3 Quadratic model

The quadratic transit model (*pytransit.QuadraticModel* and *pytransit.QuadraticModelCL*) reproduces an exoplanet transit over a stellar disk with the limb darkening modelled by a quadratic limb darkening model, as presented in Mandel & Agol (ApJ 580, 2001).

- Quadratic model example

2.4.4 Oblate star model

TBD

2.4.5 Power-2 model

Power-2 model (*pytransit.QPower2Model* and `pytransit.QPower2ModelCL`) implements the transit model with a power-2 law limb darkening profile presented by Maxted & Gill (A&A 622, A33 2019). The model is fast to evaluate and aims to model the limb darkening accurately for *cool stars*.

- Power-2 model example

Notes:

- Accurate limb darkening model for cool stars.
- Fast to evaluate.

2.4.6 General model

The general model (`pytransit.GeneralModel`) implements the flexible transit model presented by Giménez (A&A 450, 2006). The stellar limb darkening follows a "general" limb darkening model, and the accuracy of limb darkening can be increased as needed.

The model is calculated using a polynomial series and both the number of polynomials *npoly* and the number of limb darkening coefficients *nldc* can be set in the initialisation. Higher *npoly* leads to a more accurate transit model, but also increases computation time. Increasing the number of limb darkening coefficients doesn't significantly increase computation time, but

Notes:

- A flexible model that can model limb darkening accurately.
- Somewhat slower to evaluate than the specialized models.
- PyTransit implements a special "transmission spectroscopy mode" for the general model that accelerates the transit model evaluation significantly for transmission spectroscopy where the light curves are computed from a spectroscopic time series.
- The four-coefficient model presented in Mandel & Agol (ApJ 580, 2001) is not implemented in PyTransit since the Giménez model offers the same functionality with higher flexibility.

2.4.7 Chromosphere model

Optically thin shell model (`pytransit.ChromosphereModel` and `pytransit.ChromosphereModelCL`) by Schlawin et al. (ApJL 722, 2010) to model a transit over a chromosphere.

- Chromosphere model example

2.5 Log posterior functions

2.5.1 Main LPFs

- BaseLPF

2.5.2 Mixin classes

- Baselines

2.6 Advanced topics

2.6.1 Supersampling

The transit models offer built-in *supersampling* for accurate modelling of long-cadence observations. The number of samples and the exposure time can be given when setting up the model

tm.set_data(times, nsamples=10, exptimes=0.02)

2.6.2 Heterogeneous time series

PyTransit allows for heterogeneous time series, that is, a single time series can contain several individual light curves (with, e.g., different time cadences and required supersampling rates) observed (possibly) in different passbands.

If a time series contains several light curves, it also needs the light curve indices for each exposure. These are given through *lcids* argument, which should be an array of integers. If the time series contains light curves observed in different passbands, the passband indices need to be given through *pbids* argument as an integer array, one per light curve. Supersampling can also be defined on per-light curve basis by giving the *nsamples'and 'exptimes* as arrays with one value per light curve.

For example, a set of three light curves, two observed in one passband and the third in another passband

```
times_1 (lc = 0, pb = 0, sc) = [1, 2, 3, 4]
times_2 (lc = 1, pb = 0, lc) = [3, 4]
times_3 (lc = 2, pb = 1, sc) = [1, 5, 6]
```

Would be set up as

```
tm.set_data(time  = [1, 2, 3, 4, 3, 4, 1, 5, 6],
            lcids = [0, 0, 0, 0, 1, 1, 2, 2, 2],
            pbids = [0, 0, 1],
            nsamples = [  1,  10,   1],
            exptimes = [0.1, 1.0, 0.1])
```

Further, each passband requires two limb darkening coefficients, so the limb darkening coefficient array for a single parameter set should now be

ldc = [u1, v1, u2, v2]

where u and v are the passband-specific quadratic limb darkening model coefficients.

2.7 API

2.7.1 Transit models

Road Runner model

class pytransit.**RoadRunnerModel**(*ldmodel: Union[str, Callable, Tuple[Callable, Callable]] = 'quadratic', interpolate: bool = False, klims: tuple = (0.005, 0.5), nk: int = 256, nzin: int = 20, nzlimb: int = 20, zcut=0.7, ng: int = 50, parallel: bool = False, small_planet_limit: float = 0.05*)

__init__(*ldmodel: Union[str, Callable, Tuple[Callable, Callable]] = 'quadratic', interpolate: bool = False, klims: tuple = (0.005, 0.5), nk: int = 256, nzin: int = 20, nzlimb: int = 20, zcut=0.7, ng: int = 50, parallel: bool = False, small_planet_limit: float = 0.05*)

The RoadRunner transit model by Parviainen (2020).

Parameters

- **interpolate** (*bool, optional*) – Use the interpolation method presented in Parviainen (2015) if true.

- **klims** (*tuple, optional*) – Radius ratio limits (kmin, kmax) for the interpolated model.
- **nk** (*int, optional*) – Radius ratio grid size for the interpolated model.
- **nz** (*int, optional*) – Normalized distance grid size for the interpolated model.

set_data(*time: Union[numpy.ndarray, List[T]], lcids: Union[numpy.ndarray, List[T], None] = None, pbids: Union[numpy.ndarray, List[T], None] = None, nsamples: Union[numpy.ndarray, List[T], None] = None, exptimes: Union[numpy.ndarray, List[T], None] = None, epids: Union[numpy.ndarray, List[T], None] = None*) → None

Set the data for the transit model.

Parameters

- **time** (*array-like*) – Array of mid-exposure times for which the model will be evaluated.
- **lcids** (*array-like, optional*) – Array of integer light curve indices. Must have the same size as the time array.
- **pbids** (*array-like, optional*) – Array of passband indices, one per light curve. Must satisfy *pbids.size == unique(lcids).size*.
- **nsamples** (*int or array-like, optional*) – Number of samples per exposure. Can either be an integer, in which case all the light curves will have the same supersampling rate, or an array of integers, in which case each light curve can have a different rate.
- **exptimes** (*float or array-like, optional*) – Exposure times, again either for all the modelled data, or one value per light curve.
- **epids** (*array-like, optional*) – Epoch indices that can be used to link a light curve to a specific zero epoch and period (for TTV calculations).

evaluate(*k: Union[float, numpy.ndarray], ldc: Union[numpy.ndarray, List[T]], t0: Union[float, numpy.ndarray], p: Union[float, numpy.ndarray], a: Union[float, numpy.ndarray], i: Union[float, numpy.ndarray], e: Union[float, numpy.ndarray, None] = None, w: Union[float, numpy.ndarray, None] = None, copy: bool = True*) → numpy.ndarray

Evaluate the transit model for a set of scalar or vector parameters.

Parameters

- **k** – Radius ratio(s) either as a single float, 1D vector, or 2D array.
- **ldc** – Limb darkening coefficients as a 1D or 2D array.
- **t0** – Transit center(s) as a float or a 1D vector.
- **p** – Orbital period(s) as a float or a 1D vector.
- **a** – Orbital semi-major axis (axes) divided by the stellar radius as a float or a 1D vector.
- **i** – Orbital inclination(s) as a float or a 1D vector.
- **e** (*optional*) – Orbital eccentricity as a float or a 1D vector.
- **w** (*optional*) – Argument of periastron as a float or a 1D vector.

Notes

The model can be evaluated either for one set of parameters or for many sets of parameters simultaneously. In the first case, the orbital parameters should all be given as floats. In the second case, the orbital parameters should be given as a 1D array-like.

Returns Modelled flux either as a 1D or 2D ndarray.

Return type ndarray

evaluate_ps(*k: Union[float, numpy.ndarray], ldc: numpy.ndarray, t0: Union[float, numpy.ndarray], p: float, a: float, i: float, e: float = 0.0, w: float = 0.0, copy: bool = True*) → numpy.ndarray

Evaluate the transit model for a set of scalar parameters.

Parameters

- **k** (*array-like*) – Radius ratio(s) either as a single float or an 1D array.
- **ldc** (*array-like*) – Limb darkening coefficients as a 1D array.
- **t0** (*float*) – Transit center as a float.
- **p** (*float*) – Orbital period as a float.
- **a** (*float*) – Orbital semi-major axis divided by the stellar radius as a float.
- **i** (*float*) – Orbital inclination(s) as a float.
- **e** (*float, optional*) – Orbital eccentricity as a float.
- **w** (*float, optional*) – Argument of periastron as a float.

Notes

This version of the *evaluate* method is optimized for calculating a single transit model (such as when using a local optimizer). If you want to evaluate the model for a large number of parameters simultaneously, use either *evaluate* or *evaluate_pv*.

Returns Modelled flux as a 1D ndarray.

Return type ndarray

evaluate_pv(*pvp: numpy.ndarray, ldc: numpy.ndarray, copy: bool = True*) → numpy.ndarray

Uniform model

class pytransit.**UniformModel**(*eclipse: bool = False*)

set_data(*time: Union[numpy.ndarray, List[T]], lcids: Union[numpy.ndarray, List[T], None] = None, pbids: Union[numpy.ndarray, List[T], None] = None, nsamples: Union[numpy.ndarray, List[T], None] = None, exptimes: Union[numpy.ndarray, List[T], None] = None, epids: Union[numpy.ndarray, List[T], None] = None*) → None

Set the data for the transit model.

Parameters

- **time** (*array-like*) – Array of mid-exposure times for which the model will be evaluated.
- **lcids** (*array-like, optional*) – Array of integer light curve indices. Must have the same size as the time array.
- **pbids** (*array-like, optional*) – Array of passband indices, one per light curve. Must satisfy *pbids.size == unique(lcids).size*.

- **nsamples** (*int or array-like, optional*) – Number of samples per exposure. Can either be an integer, in which case all the light curves will have the same supersampling rate, or an array of integers, in which case each light curve can have a different rate.
- **exptimes** (*float or array-like, optional*) – Exposure times, again either for all the modelled data, or one value per light curve.
- **epids** (*array-like, optional*) – Epoch indices that can be used to link a light curve to a specific zero epoch and period (for TTV calculations).

evaluate(*k: Union[float, numpy.ndarray], t0: Union[float, numpy.ndarray], p: Union[float, numpy.ndarray], a: Union[float, numpy.ndarray], i: Union[float, numpy.ndarray], e: Union[float, numpy.ndarray] = None, w: Union[float, numpy.ndarray] = None, copy: bool = True*) → numpy.ndarray

Evaluates the uniform transit model for a set of scalar or vector parameters.

Parameters

- **k** – Radius ratio(s) either as a single float, 1D vector, or 2D array.
- **t0** – Transit center(s) as a float or a 1D vector.
- **p** – Orbital period(s) as a float or a 1D vector.
- **a** – Orbital semi-major axis (axes) divided by the stellar radius as a float or a 1D vector.
- **i** – Orbital inclination(s) as a float or a 1D vector.
- **e** – Orbital eccentricity as a float or a 1D vector.
- **w** – Argument of periastron as a float or a 1D vector.
- **copy** –

Notes

The model can be evaluated either for one set of parameters or for many sets of parameters simultaneously. The orbital parameters can be given either as a float or a 1D array-like (preferably ndarray for optimal speed.)

Returns

Return type Transit model

evaluate_ps(*k: float, t0: float, p: float, a: float, i: float, e: float = 0.0, w: float = 0.0*) → numpy.ndarray

Evaluate the transit model for a set of scalar parameters.

Parameters

- **k** (*array-like*) – Radius ratio(s) either as a single float or an 1D array.
- **t0** (*float*) – Transit center as a float.
- **p** (*float*) – Orbital period as a float.
- **a** (*float*) – Orbital semi-major axis divided by the stellar radius as a float.
- **i** (*float*) – Orbital inclination(s) as a float.
- **e** (*float, optional*) – Orbital eccentricity as a float.
- **w** (*float, optional*) – Argument of periastron as a float.

Notes

This version of the *evaluate* method is optimized for calculating a single transit model (such as when using a local optimizer). If you want to evaluate the model for a large number of parameters simultaneously, use either *evaluate* or *evaluate_pv*.

Returns Modelled flux as a 1D ndarray.

Return type ndarray

evaluate_pv(*pvp: numpy.ndarray*) → numpy.ndarray

Evaluate the transit model for a 2D parameter array.

Parameters **pvp** – Parameter array with a shape *(npv, npar)* where *npv* is the number of parameter vectors, and each row contains a set of parameters *[k, t0, p, a, i, e, w]*. The radius ratios can also be given per passband, in which case the row should be structured as *[k_0, k_1, k_2, . . . , k_npb, t0, p, a, i, e, w]*.

Notes

This version of the *evaluate* method is optimized for calculating several models in parallel, such as when using *emcee* for MCMC sampling.

Returns Modelled flux either as a 1D or 2D ndarray.

Return type ndarray

OpenCL Uniform model

pytransit.**UniformModelCL**

alias of pytransit.DummyModelCL

Quadratic model

class pytransit.**QuadraticModel**(*interpolate: bool = True, klims: tuple = (0.01, 0.5), nk: int = 256, nz: int = 256*)

Transit model with quadratic limb darkening (Mandel & Agol, ApJ 580, L171-L175 2002).

__init__(*interpolate: bool = True, klims: tuple = (0.01, 0.5), nk: int = 256, nz: int = 256*)

Transit model with quadratic limb darkening (Mandel & Agol, ApJ 580, L171-L175 2002).

Parameters

- **interpolate** (*bool, optional*) – Use the interpolation method presented in Parviainen (2015) if true.
- **klims** (*tuple, optional*) – Radius ratio limits (kmin, kmax) for the interpolated model.
- **nk** (*int, optional*) – Radius ratio grid size for the interpolated model.
- **nz** (*int, optional*) – Normalized distance grid size for the interpolated model.

set_data(*time: Union[numpy.ndarray, List[T]], lcids: Union[numpy.ndarray, List[T], None] = None, pbids: Union[numpy.ndarray, List[T], None] = None, nsamples: Union[numpy.ndarray, List[T], None] = None, exptimes: Union[numpy.ndarray, List[T], None] = None, epids: Union[numpy.ndarray, List[T], None] = None*) → None

Set the data for the transit model.

Parameters

- **time** (*array-like*) – Array of mid-exposure times for which the model will be evaluated.
- **lcids** (*array-like, optional*) – Array of integer light curve indices. Must have the same size as the time array.
- **pbids** (*array-like, optional*) – Array of passband indices, one per light curve. Must satisfy *pbids.size == unique(lcids).size*.
- **nsamples** (*int or array-like, optional*) – Number of samples per exposure. Can either be an integer, in which case all the light curves will have the same supersampling rate, or an array of integers, in which case each light curve can have a different rate.
- **exptimes** (*float or array-like, optional*) – Exposure times, again either for all the modelled data, or one value per light curve.
- **epids** (*array-like, optional*) – Epoch indices that can be used to link a light curve to a specific zero epoch and period (for TTV calculations).

evaluate(*k: Union[float, numpy.ndarray], ldc: Union[numpy.ndarray, List[T]], t0: Union[float, numpy.ndarray], p: Union[float, numpy.ndarray], a: Union[float, numpy.ndarray], i: Union[float, numpy.ndarray], e: Union[float, numpy.ndarray, None] = None, w: Union[float, numpy.ndarray, None] = None, copy: bool = True*) → numpy.ndarray

Evaluate the transit model for a set of scalar or vector parameters.

Parameters

- **k** – Radius ratio(s) either as a single float, 1D vector, or 2D array.
- **ldc** – Limb darkening coefficients as a 1D or 2D array.
- **t0** – Transit center(s) as a float or a 1D vector.
- **p** – Orbital period(s) as a float or a 1D vector.
- **a** – Orbital semi-major axis (axes) divided by the stellar radius as a float or a 1D vector.
- **i** – Orbital inclination(s) as a float or a 1D vector.
- **e** (*optional*) – Orbital eccentricity as a float or a 1D vector.
- **w** (*optional*) – Argument of periastron as a float or a 1D vector.

Notes

The model can be evaluated either for one set of parameters or for many sets of parameters simultaneously. In the first case, the orbital parameters should all be given as floats. In the second case, the orbital parameters should be given as a 1D array-like.

Returns Modelled flux either as a 1D or 2D ndarray.

Return type ndarray

evaluate_ps(*k: Union[float, numpy.ndarray], ldc: numpy.ndarray, t0: Union[float, numpy.ndarray], p: float, a: float, i: float, e: float = 0.0, w: float = 0.0, copy: bool = True*) → numpy.ndarray

Evaluate the transit model for a set of scalar parameters.

Parameters

- **k** (*array-like*) – Radius ratio(s) either as a single float or an 1D array.

- **ldc** (*array-like*) – Limb darkening coefficients as a 1D array.
- **t0** (*float*) – Transit center as a float.
- **p** (*float*) – Orbital period as a float.
- **a** (*float*) – Orbital semi-major axis divided by the stellar radius as a float.
- **i** (*float*) – Orbital inclination as a float.
- **e** (*float, optional*) – Orbital eccentricity as a float.
- **w** (*float, optional*) – Argument of periastron as a float.

Notes

This version of the *evaluate* method is optimized for calculating a single transit model (such as when using a local optimizer). If you want to evaluate the model for a large number of parameters simultaneously, use either *evaluate* or *evaluate_pv*.

Returns Modelled flux as a 1D ndarray.

Return type ndarray

evaluate_pv(*pvp: numpy.ndarray*, *ldc: numpy.ndarray*, *copy: bool = True*) → numpy.ndarray

Evaluate the transit model for a 2D parameter array.

Parameters

- **pvp** (*ndarray*) – Parameter array with a shape *(npv, npar)* where *npv* is the number of parameter vectors, and each row contains a set of parameters *[k, t0, p, a, i, e, w]*. The radius ratios can also be given per passband, in which case the row should be structured as *[k_0, k_1, k_2, . . . , k_npb, t0, p, a, b, e, w]*.
- **ldc** (*ndarray*) – Limb darkening coefficient array with shape *(npv, 2*npb)*, where *npv* is the number of parameter vectors and *npb* is the number of passbands.

Notes

This version of the *evaluate* method is optimized for calculating several models in parallel, such as when using *emcee* for MCMC sampling.

Returns Modelled flux either as a 1D or 2D ndarray.

Return type ndarray

OpenCL Quadratic model

pytransit.**QuadraticModelCL**

alias of pytransit.DummyModelCL

Oblate star model

class pytransit.**OblateStarModel**(*rstar: float = 1.0*, *wavelength: float = 510*, *sres: int = 80*, *pres: int = 6*)

Transit model for a gravity-darkened fast-rotating oblate star.

Transit model for a gravity-darkened fast-rotating oblate star following Barnes (ApJ, 2009, 705).

__init__(*rstar: float = 1.0, wavelength: float = 510, sres: int = 80, pres: int = 6*)

Parameters

- **rstar** – Stellar radius [R_Sun]
- **wavelength** – Effective wavelength [nm]
- **sres** – Stellar discretization resolution
- **pres** – Planet discretization resolution

set_data(*time: Union[numpy.ndarray, List[T]], lcids: Union[numpy.ndarray, List[T], None] = None, pbids: Union[numpy.ndarray, List[T], None] = None, nsamples: Union[numpy.ndarray, List[T], None] = None, exptimes: Union[numpy.ndarray, List[T], None] = None, epids: Union[numpy.ndarray, List[T], None] = None*) → None

Set the data for the transit model.

Parameters

- **time** (*array-like*) – Array of mid-exposure times for which the model will be evaluated.
- **lcids** (*array-like, optional*) – Array of integer light curve indices. Must have the same size as the time array.
- **pbids** (*array-like, optional*) – Array of passband indices, one per light curve. Must satisfy *pbids.size == unique(lcids).size*.
- **nsamples** (*int or array-like, optional*) – Number of samples per exposure. Can either be an integer, in which case all the light curves will have the same supersampling rate, or an array of integers, in which case each light curve can have a different rate.
- **exptimes** (*float or array-like, optional*) – Exposure times, again either for all the modelled data, or one value per light curve.
- **epids** (*array-like, optional*) – Epoch indices that can be used to link a light curve to a specific zero epoch and period (for TTV calculations).

evaluate(*k: Union[float, numpy.ndarray], ldc: numpy.ndarray, t0: Union[float, numpy.ndarray], p: Union[float, numpy.ndarray], a: Union[float, numpy.ndarray], i: Union[float, numpy.ndarray], e: Union[float, numpy.ndarray] = None, w: Union[float, numpy.ndarray] = None, copy: bool = True*) → numpy.ndarray

evaluate_ps(*k: Union[float, numpy.ndarray], rho: float, rperiod: float, tpole: float, phi: float, beta: float, ldc: numpy.ndarray, t0: float, p: float, a: float, i: float, l: float = 0.0, e: float = 0.0, w: float = 0.0, copy: bool = True*) → numpy.ndarray

Evaluate the transit model for a set of scalar parameters.

Parameters

- **k** (*array-like*) – Radius ratio(s) either as a single float or an 1D array
- **rho** (*float*) – Stellar density [g/cm^3]
- **rperiod** (*float*) – Stellar rotation period [d]
- **tpole** (*float*) – Temperature at the pole [K]
- **phi** (*float*) – Star's obliquity to the plane of the sky [rad]
- **beta** (*float*) – Gravity darkening parameter
- **ldc** (*array-like*) – Limb darkening coefficients as a 1D array

- **t0** (*float*) – Zero epoch
- **p** (*float*) – Orbital period [d]
- **a** (*float*) – Scaled orbital semi-major axis [R_star]
- **i** (*float*) – Orbital inclination [rad]
- **l** (*float*) – Orbital azimuth angle [rad]
- **e** (*float, optional*) – Orbital eccentricity
- **w** (*float, optional*) – Argument of periastron

Notes

This version of the *evaluate* method is optimized for calculating a single transit model (such as when using a local optimizer). If you want to evaluate the model for a large number of parameters simultaneously, use either *evaluate* or *evaluate_pv*.

Returns Modelled flux as a 1D ndarray.

Return type ndarray

evaluate_pv(*pvp: numpy.ndarray, ldc: numpy.ndarray, copy: bool = True*) → numpy.ndarray

General model

class pytransit.**GeneralModel**(*npol: int = 50, nldc: int = 2, mode: int = 0*)

Transit model with general limb darkening (Giménez, A&A 450, 1231–1237, 2006).

Transit model with general limb darkening (Giménez, A&A 450, 1231–1237, 2006) with optimizations described in Parviainen (MNRAS 450, 3233–3238, 2015).

The general limb darkening law is

$$I(\mu) = I(1)(1 - \sum_{n=1}^{N} u_n(1 - \mu^n))$$

__init__(*npol: int = 50, nldc: int = 2, mode: int = 0*)

Initialize self. See help(type(self)) for accurate signature.

set_data(*time: Union[numpy.ndarray, List[T]], lcids: Union[numpy.ndarray, List[T], None] = None, pbids: Union[numpy.ndarray, List[T], None] = None, nsamples: Union[numpy.ndarray, List[T], None] = None, exptimes: Union[numpy.ndarray, List[T], None] = None, epids: Union[numpy.ndarray, List[T], None] = None*) → None

Set the data for the transit model.

Parameters

- **time** (*array-like*) – Array of mid-exposure times for which the model will be evaluated.
- **lcids** (*array-like, optional*) – Array of integer light curve indices. Must have the same size as the time array.
- **pbids** (*array-like, optional*) – Array of passband indices, one per light curve. Must satisfy *pbids.size == unique(lcids).size*.

- **nsamples** (*int or array-like, optional*) – Number of samples per exposure. Can either be an integer, in which case all the light curves will have the same supersampling rate, or an array of integers, in which case each light curve can have a different rate.
- **exptimes** (*float or array-like, optional*) – Exposure times, again either for all the modelled data, or one value per light curve.
- **epids** (*array-like, optional*) – Epoch indices that can be used to link a light curve to a specific zero epoch and period (for TTV calculations).

evaluate(*k: Union[float, numpy.ndarray], ldc: Union[numpy.ndarray, List[T]], t0: Union[float, numpy.ndarray], p: Union[float, numpy.ndarray], a: Union[float, numpy.ndarray], i: Union[float, numpy.ndarray], e: Union[float, numpy.ndarray, None] = None, w: Union[float, numpy.ndarray, None] = None, copy: bool = True*) → numpy.ndarray

Evaluate the transit model for a set of scalar or vector parameters.

Parameters

- **k** – Radius ratio(s) either as a single float, 1D vector, or 2D array.
- **ldc** – Limb darkening coefficients as a 1D or 2D array.
- **t0** – Transit center(s) as a float or a 1D vector.
- **p** – Orbital period(s) as a float or a 1D vector.
- **a** – Orbital semi-major axis (axes) divided by the stellar radius as a float or a 1D vector.
- **i** – Orbital inclination(s) as a float or a 1D vector.
- **e** (*optional*) – Orbital eccentricity as a float or a 1D vector.
- **w** (*optional*) – Argument of periastron as a float or a 1D vector.

Notes

The model can be evaluated either for one set of parameters or for many sets of parameters simultaneously. In the first case, the orbital parameters should all be given as floats. In the second case, the orbital parameters should be given as a 1D array-like.

Returns Modelled flux either as a 1D or 2D ndarray.

Return type ndarray

evaluate_ps(*k: Union[float, numpy.ndarray], ldc: numpy.ndarray, t0: float, p: float, a: float, i: float, e: float = 0.0, w: float = 0.0, copy: bool = True*) → numpy.ndarray

Evaluate the transit model for a set of scalar parameters.

Parameters

- **k** (*array-like*) – Radius ratio(s) either as a single float or an 1D array.
- **ldc** (*array-like*) – Limb darkening coefficients as a 1D array.
- **t0** (*float*) – Transit center as a float.
- **p** (*float*) – Orbital period as a float.
- **a** (*float*) – Orbital semi-major axis divided by the stellar radius as a float.
- **i** (*float*) – Orbital inclination(s) as a float.
- **e** (*float, optional*) – Orbital eccentricity as a float.
- **w** (*float, optional*) – Argument of periastron as a float.

Notes

This version of the *evaluate* method is optimized for calculating a single transit model (such as when using a local optimizer). If you want to evaluate the model for a large number of parameters simultaneously, use either *evaluate* or *evaluate_pv*.

Returns Modelled flux as a 1D ndarray.

Return type ndarray

evaluate_pv(*pvp: numpy.ndarray*, *ldc: numpy.ndarray*, *copy: bool = True*) → numpy.ndarray

Evaluate the transit model for a 2D parameter array.

Parameters

- **pvp** (*ndarray*) – Parameter array with a shape *(npv, npar)* where *npv* is the number of parameter vectors, and each row contains a set of parameters *[k, t0, p, a, i, e, w]*. The radius ratios can also be given per passband, in which case the row should be structured as *[k_0, k_1, k_2, . . . , k_npb, t0, p, a, i, e, w]*.
- **ldc** (*ndarray*) – Limb darkening coefficient array with shape *(npv, 2*npb)*, where *npv* is the number of parameter vectors and *npb* is the number of passbands.

Notes

This version of the *evaluate* method is optimized for calculating several models in parallel, such as when using *emcee* for MCMC sampling.

Returns Modelled flux either as a 1D or 2D ndarray.

Return type ndarray

QPower2 model

class pytransit.**QPower2Model**

QPower2 transit model by Maxted & Gill (A&A, 622, A33, 2019).

set_data(*time: Union[numpy.ndarray, List[T]], lcids: Union[numpy.ndarray, List[T], None] = None, pbids: Union[numpy.ndarray, List[T], None] = None, nsamples: Union[numpy.ndarray, List[T], None] = None, exptimes: Union[numpy.ndarray, List[T], None] = None, epids: Union[numpy.ndarray, List[T], None] = None*) → None

Set the data for the transit model.

Parameters

- **time** (*array-like*) – Array of mid-exposure times for which the model will be evaluated.
- **lcids** (*array-like, optional*) – Array of integer light curve indices. Must have the same size as the time array.
- **pbids** (*array-like, optional*) – Array of passband indices, one per light curve. Must satisfy *pbids.size == unique(lcids).size*.
- **nsamples** (*int or array-like, optional*) – Number of samples per exposure. Can either be an integer, in which case all the light curves will have the same supersampling rate, or an array of integers, in which case each light curve can have a different rate.

- **exptimes** (*float or array-like, optional*) – Exposure times, again either for all the modelled data, or one value per light curve.
- **epids** (*array-like, optional*) – Epoch indices that can be used to link a light curve to a specific zero epoch and period (for TTV calculations).

evaluate(*k: Union[float, numpy.ndarray], ldc: numpy.ndarray, t0: Union[float, numpy.ndarray], p: Union[float, numpy.ndarray], a: Union[float, numpy.ndarray], i: Union[float, numpy.ndarray], e: Union[float, numpy.ndarray, None] = None, w: Union[float, numpy.ndarray, None] = None, copy: bool = True*) → numpy.ndarray

Evaluate the transit model for a set of scalar or vector parameters.

Parameters

- **k** – Radius ratio(s) either as a single float, 1D vector, or 2D array.
- **ldc** – Limb darkening coefficients as a 1D or 2D array.
- **t0** – Transit center(s) as a float or a 1D vector.
- **p** – Orbital period(s) as a float or a 1D vector.
- **a** – Orbital semi-major axis (axes) divided by the stellar radius as a float or a 1D vector.
- **i** – Orbital inclination(s) as a float or a 1D vector.
- **e** (*optional*) – Orbital eccentricity as a float or a 1D vector.
- **w** (*optional*) – Argument of periastron as a float or a 1D vector.

Notes

The model can be evaluated either for one set of parameters or for many sets of parameters simultaneously. In the first case, the orbital parameters should all be given as floats. In the second case, the orbital parameters should be given as a 1D array-like.

Returns Modelled flux either as a 1D or 2D ndarray.

Return type ndarray

evaluate_ps(*k: float, ldc: numpy.ndarray, t0: float, p: float, a: float, i: float, e: float = 0.0, w: float = 0.0*) → numpy.ndarray

Evaluate the transit model for a set of scalar parameters.

Parameters

- **k** (*array-like*) – Radius ratio(s) either as a single float or an 1D array.
- **ldc** – Limb darkening coefficients as a 1D or 2D array.
- **t0** (*float*) – Transit center as a float.
- **p** (*float*) – Orbital period as a float.
- **a** (*float*) – Orbital semi-major axis divided by the stellar radius as a float.
- **i** (*float*) – Orbital inclination(s) as a float.
- **e** (*float, optional*) – Orbital eccentricity as a float.
- **w** (*float, optional*) – Argument of periastron as a float.

Notes

This version of the *evaluate* method is optimized for calculating a single transit model (such as when using a local optimizer). If you want to evaluate the model for a large number of parameters simultaneously, use either *evaluate* or *evaluate_pv*.

Returns Modelled flux as a 1D ndarray.

Return type ndarray

evaluate_pv(*pvp: numpy.ndarray*, *ldc: numpy.ndarray*) → numpy.ndarray

Evaluate the transit model for a 2D parameter array.

Parameters **pvp** – Parameter array with a shape *(npv, npar)* where *npv* is the number of parameter vectors, and each row contains a set of parameters *[k, t0, p, a, i, e, w]*. The radius ratios can also be given per passband, in which case the row should be structured as *[k_0, k_1, k_2, . . . , k_npb, t0, p, a, i, e, w]*.

Notes

This version of the *evaluate* method is optimized for calculating several models in parallel, such as when using *emcee* for MCMC sampling.

Returns Modelled flux either as a 1D or 2D ndarray.

Return type ndarray

Chromosphere model

class pytransit.**ChromosphereModel**

set_data(*time: Union[numpy.ndarray, List[T]], lcids: Union[numpy.ndarray, List[T], None] = None, pbids: Union[numpy.ndarray, List[T], None] = None, nsamples: Union[numpy.ndarray, List[T], None] = None, exptimes: Union[numpy.ndarray, List[T], None] = None, epids: Union[numpy.ndarray, List[T], None] = None*) → None

Set the data for the transit model.

Parameters

- **time** (*array-like*) – Array of mid-exposure times for which the model will be evaluated.
- **lcids** (*array-like, optional*) – Array of integer light curve indices. Must have the same size as the time array.
- **pbids** (*array-like, optional*) – Array of passband indices, one per light curve. Must satisfy *pbids.size == unique(lcids).size*.
- **nsamples** (*int or array-like, optional*) – Number of samples per exposure. Can either be an integer, in which case all the light curves will have the same supersampling rate, or an array of integers, in which case each light curve can have a different rate.
- **exptimes** (*float or array-like, optional*) – Exposure times, again either for all the modelled data, or one value per light curve.
- **epids** (*array-like, optional*) – Epoch indices that can be used to link a light curve to a specific zero epoch and period (for TTV calculations).

evaluate_ps(*k: float, t0: float, p: float, a: float, i: float, e: float = 0.0, w: float = 0.0*) → numpy.ndarray

Evaluate the transit model for a set of scalar parameters.

Parameters

- **k** (*array-like*) – Radius ratio(s) either as a single float or an 1D array.
- **t0** (*float*) – Transit center as a float.
- **p** (*float*) – Orbital period as a float.
- **a** (*float*) – Orbital semi-major axis divided by the stellar radius as a float.
- **i** (*float*) – Orbital inclination(s) as a float.
- **e** (*float, optional*) – Orbital eccentricity as a float.
- **w** (*float, optional*) – Argument of periastron as a float.

Notes

This version of the *evaluate* method is optimized for calculating a single transit model (such as when using a local optimizer). If you want to evaluate the model for a large number of parameters simultaneously, use either *evaluate* or *evaluate_pv*.

Returns Modelled flux as a 1D ndarray.

Return type ndarray

evaluate_pv(*pvp: numpy.ndarray*) → numpy.ndarray

Evaluate the transit model for a 2D parameter array.

Parameters **pvp** – Parameter array with a shape *(npv, npar)* where *npv* is the number of parameter vectors, and each row contains a set of parameters *[k, t0, p, a, i, e, w]*. The radius ratios can also be given per passband, in which case the row should be structured as *[k_0, k_1, k_2, . . . , k_npb, t0, p, a, i, e, w]*.

Notes

This version of the *evaluate* method is optimized for calculating several models in parallel, such as when using *emcee* for MCMC sampling.

Returns Modelled flux either as a 1D or 2D ndarray.

Return type ndarray

2.7.2 Log posterior functions

Log posterior functions for transit modelling and parameter estimation.

A log posterior function (LPF) class creates a basis for Bayesian parameter estimation from transit light curves. In PyTransit, LPFs are a bit more than what the name implies. An LPF stores the observations, model priors, etc. It also contains methods for posterior optimisation and MCMC sampling.

class pytransit.lpf.**BaseLPF**(*name: str, passbands: list, times: list = None, fluxes: Iterable[T_co] = None, errors: list = None, pbids: list = None, covariates: list = None, wnids: list = None, tm: pytransit.models.transitmodel.TransitModel = None, nsamples: tuple = 1, exptimes: tuple = 0.0, init_data=True, result_dir: pathlib.Path = None, tref: float = 0.0, lnlikelihood: str = 'wn'*)

add_as_prior(*mean: float, std: float*) → None

Add a normal prior on the scaled semi-major axis $(a/R_\star)$.

Parameters

- **mean** (*float*) – Mean of the normal distribution.
- **std** (*float*) – Standard deviation of the normal distribution

add_ldtk_prior(*teff: tuple, logg: tuple, z: tuple, passbands: tuple, uncertainty_multiplier: float = 3, **kwargs*) → None

Add a LDTk-based prior on the limb darkening.

Parameters

- **teff** –
- **logg** –
- **z** –
- **passbands** –
- **uncertainty_multiplier** –

add_prior(*prior*)

add_t14_prior(*mean: float, std: float*) → None

Add a normal prior on the transit duration.

Parameters

- **mean** (*float*) – Mean of the normal distribution
- **std** (*float*) – Standard deviation of the normal distribution.

baseline(*pv*)

create_pv_population(*npop=50*)

flux_model(*pv*)

lnlikelihood(*pvp*)

Log likelihood for a 1D or 2D array of model parameters.

Parameters **pvp** (*ndarray*) – Either a 1D parameter vector or a 2D parameter array.

Returns

Return type Log likelihood for the given parameter vector(s)

plot_light_curves(*method='de', ncol: int = 3, width: float = 2.0, max_samples: int = 1000, figsize=None, data_alpha=0.5, ylim=None*)

posterior_samples(*burn: int = 0, thin: int = 1, derived_parameters: bool = True*)

remove_outliers(*sigma=5*)

remove_transits(*tids*)

residuals(*pv*)

set_radius_ratio_prior(*kmin*, *kmax*)
Set a uniform prior on all radius ratios.

transit_model(*pv*, *copy=True*)

trends(*pv*)
Additive trends

2.7.3 pytransit.contamination

Module to model flux contamination in transit light curves.

class pytransit.contamination.**SMContamination**(*instrument: pytransit.contamination.instrument.Instrument*, *ref_pb: str = None*)

Bases: pytransit.contamination.contamination._BaseContamination

A class that models flux contamination based on stellar spectrum models.

absolute_flux(*teff: float, wl: Union[float, Iterable[T_co]]*) → numpy.ndarray
The absolute flux given an effective temperature and a set of wavelength.

Parameters

- **teff** (*float*) – The effective temperature [K]
- **wl** (*array-like*) – The wavelengths to calculate the flux in [nm]

Returns

Return type Spectral radiance.

apply_extinction

contamination(*cref: Union[float, numpy.ndarray], teff1: Union[float, numpy.ndarray], teff2: Union[float, numpy.ndarray]*)
Contamination given reference contamination, host TEff, and contaminant TEff(s)

Per-passband contamination given the contamination in the reference passband and TEffs of the two stars.

Parameters

- **cref** (*float*) – contamination in the reference passband
- **teff1** – Effective stellar temperature [K]
- **teff2** – Effective stellar temperature [K]

Returns

Return type Per-passband contamination

reddening(*a*)

relative_flux(*teff: float, wl: Union[float, numpy.ndarray], wlref: float*) → numpy.ndarray
The stellar flux normalized to a given reference wavelength.

Parameters

- **teff** (*float*) – The effective temperature of the radiating body [K]
- **wl** (*array-like*) – The wavelengths to calculate the flux in [nm]
- **wlref** (*float*) – The reference wavelength [nm]

Returns

Return type The flux normalized to a given reference wavelength

relative_flux_mixture(*teffs*, *fractions*, *rdc=None*)

relative_fluxes(*teff: Union[float, numpy.ndarray], rdc=None, rpb=None*)

class pytransit.contamination.**BBContamination**(*instrument: pytransit.contamination.instrument.Instrument*, *ref_pb: str*, *delta_l: float = 10*)

Bases: pytransit.contamination.contamination._BaseContamination

Third light contamination based on black body approximation.

This class offers a simple black-body model for flux contamination in which the target star and the contaminant(s) are approximated as black bodies with effective temperatures Tt, Tc1, Tc2, . . . , Tcn.

static absolute_flux(*teff: float, wl: Union[float, Iterable[T_co]]*) → numpy.ndarray

The absolute flux given an effective temperature and wavelength.

Parameters

- **teff** – The effective temperature in K
- **wl** – The wavelength (or an array of) in nm

Returns

Return type Black body spectral radiance.

absolute_fluxes(*teff: float*) → numpy.ndarray

Calculates the integrated absolute fluxes for all filters for a star with the given effective temperature

Parameters **teff** – The effective temperature of the radiating body [K]

Returns

Return type The integrated absolute fluxes for the filters in the instrument.

contamination(*cref: float*, *teff1: float*, *teff2: float*) → numpy.ndarray

Calculates the contamination factors for all the filters given the contamination in the reference passband.

Parameters

- **cref** – Reference passband contamination
- **teff1** – Host star effective temperature
- **teff2** – Contaminant effective temperature

Returns

Return type Contamination factors for all the filters.

static relative_flux(*teff: float, wl: Union[float, numpy.ndarray], wlref: float*) → numpy.ndarray

The black body flux normalized to a given reference wavelength.

Parameters

- **teff** – The effective temperature of the radiating body [K]
- **wl** – The wavelength [nm]
- **wlref** – The reference wavelength [nm]

Returns

Return type The black body flux normalized to a given reference wavelength

relative_fluxes(*teff: Union[float, numpy.ndarray]*) → numpy.ndarray
Calculates the integrated fluxes for all filters normalized to the reference passband.

Parameters **teff** – The effective temperature of the radiating body [K]

Returns

Return type The integrated fluxes for all filters normalized to the reference passband.

class pytransit.contamination.**Instrument**(*name*, *filters*, *qes=None*)
Bases: object

class pytransit.contamination.**ClearFilter**(*name*)
Bases: pytransit.contamination.filter.Filter

Constant unity transmission.

class pytransit.contamination.**BoxcarFilter**(*name*, *wl_min*, *wl_max*)
Bases: pytransit.contamination.filter.Filter

Filter with a transmission of 1 inside the minimum and maximum wavelengths and 0 outside.

class pytransit.contamination.**TabulatedFilter**(*name*, *wl*, *tm*)
Bases: pytransit.contamination.filter.Filter

Interpolated tabulated filter.

pytransit.contamination.**true_radius_ratio**(*apparent_k: float*, *contamination: float*) → float

pytransit.contamination.**apparent_radius_ratio**(*true_k: float*, *contamination: float*) → float

pytransit.contamination.**contaminate_light_curve**
Contaminates a transit light curve.

Contaminates a transit light curve with npb passbands.

Parameters

- **flux** (*1d array-like*) – Transit light curve with npb passbands.
- **contamination** (*1d array-like*) – Array of per-passband contamination values.
- **pbids** (*1d array-like*) – Passband indices that map each light curve element to a single passband.

Returns

Return type Contaminated transit light curve

2.7.4 Phase curves

pytransit.utils.phasecurves.**doppler_boosting_alpha**(*teff: float*, *flt*)
The photon weighted bandpass-integrated boosting factor.

Parameters

- **teff** – Effective temperature of the star [K]
- **flt** – Passband transmission

Returns The photon weighted bandpass-integrated boosting factor.

Return type float

`pytransit.utils.phasecurves.`**`doppler_boosting_amplitude`**(*mp: Union[float, numpy.ndarray], ms: Union[float, numpy.ndarray], period: Union[float, numpy.ndarray], alpha: Union[float, numpy.ndarray]*) → Union[float, numpy.ndarray]

The amplitude of the doppler boosting signal.

Calculates the amplitude of the doppler boosting (beaming, reflex doppler effect) signal following the approach described by Loeb & Gaudi in [Loeb2003] . Note that you need to pre-calculate the photon-weighted bandpass-integrated boosting factor (alpha) [Bloemen2010] [Barclay2012] for the star and the instrument using `doppler_boosting_alpha`.

Parameters

- **mp** (*float or ndarray*) – Planetary mass [MJup]
- **ms** (*float or ndarray*) – Stellar mass [MSun]
- **period** (*float or ndarray*) – Orbital period [d]
- **alpha** (*float or ndarray*) – Photon-weighted bandpass-integrated boosting factor

Returns Doppler boosting signal amplitude

Return type float or ndarray

References

`pytransit.utils.phasecurves.`**`ellipsoidal_variation_amplitude`**(*mp: Union[float, numpy.ndarray], ms: Union[float, numpy.ndarray], a: Union[float, numpy.ndarray], i: Union[float, numpy.ndarray], u: Union[float, numpy.ndarray], g: Union[float, numpy.ndarray]*) → Union[float, numpy.ndarray]

The amplitude of the ellipsoidal variation signal.

Calculates the amplitude of the ellipsoidal variation signal following the approach described by Lillo-Box et al. in [Lillo-Box2014], page 11.

Parameters

- **mp** (*float or ndarray*) – Planetary mass [MJup]
- **ms** (*float or ndarray*) – Stellar mass [MSun]
- **a** (*float or ndarray*) – Semi-major axis of the orbit divided by the stellar radius
- **i** (*float or ndarray*) – Orbital inclination [rad]

- **u** (*float or ndarray*) – Linear limb darkening coefficient
- **g** (*float or ndarray*) – Gravity darkening coefficient

Returns **ev_amplitude** – The amplitude of the ellipsoidal variation signal

Return type float or ndarray

References

pytransit.utils.phasecurves.**ellipsoidal_variation_signal**(*f: Union[float, numpy.ndarray], theta: Union[float, numpy.ndarray], e: float*) → Union[float, numpy.ndarray]

Parameters

- **f** – True anomaly [rad]
- **theta** – Angle between the line-of-sight and the star-planet direction
- **e** – Eccentricity

pytransit.utils.phasecurves.**emission**(*tp: Union[float, numpy.ndarray], tstar: Union[float, numpy.ndarray], k: Union[float, numpy.ndarray], flt*) → Union[float, numpy.ndarray]

Thermal emission from the planet.

Parameters

- **tp** (*float or ndarray*) – Equilibrium temperature of the planet [K]
- **tstar** (*float or ndarray*) – Effective temperature of the star [K]
- **k** (*float or ndarray*) – Planet-star radius ratio
- **flt** (*Filter*) – Passband transmission

Returns

- *float or ndarray*
- *References*

pytransit.utils.phasecurves.**equilibrium_temperature**(*tstar: Union[float, numpy.ndarray], a: Union[float, numpy.ndarray], f: Union[float, numpy.ndarray], ab: Union[float, numpy.ndarray]*) → Union[float, numpy.ndarray]

Planetary equilibrium temperature [K].

Parameters

- **tstar** – Effective stellar temperature [K]
- **a** – Scaled semi-major axis [Rsun]
- **f** – Redistribution factor
- **ab** – Bond albedo

Returns **Teq** – Equilibrium temperature [K]

Return type float or ndarray

pytransit.utils.phasecurves.**flux_ratio**(*tstar: Union[float, numpy.ndarray], a: Union[float, numpy.ndarray], f: Union[float, numpy.ndarray], ab: Union[float, numpy.ndarray], l: Union[float, numpy.ndarray], r: Union[float, numpy.ndarray] = 1.5, ti: Union[float, numpy.ndarray] = 0*) → Union[float, numpy.ndarray]

Total flux ratio per projected area element.

Parameters

- **tstar** – Effective stellar temperature [K]
- **a** – Scaled semi-major axis [Rs]
- **f** – Redistribution factor
- **ab** – Bond albedo
- **l** – Wavelength [m]
- **r** – Inverse of the phase integral
- **ti** – Temperature [K]

Returns **fr** – Total flux ratio

Return type float

pytransit.utils.phasecurves.**planck**(*t: Union[float, numpy.ndarray], l: Union[float, numpy.ndarray]*) → Union[float, numpy.ndarray]

Radiance of a black body as a function of wavelength.

Parameters

- **t** – Black body temperature [K]
- **l** – Wavelength [m]

Returns **L** – Back body radiance [W m^-2 sr^-1]

Return type float or ndarray

pytransit.utils.phasecurves.**reflected_fr**(*a: Union[float, numpy.ndarray], ab: Union[float, numpy.ndarray], r: Union[float, numpy.ndarray] = 1.5*) → Union[float, numpy.ndarray]

Reflected flux ratio per projected area element.

Parameters

- **a** – Scaled semi-major axis [Rsun]
- **ab** – Bond albedo
- **r** – Inverse of the phase integral

Returns **fr** – Reflected flux ratio

Return type float

pytransit.utils.phasecurves.**solve_ab**(*fr, tstar, a, f, l, r=1.5, ti=0*)

Solve the Bond albedo.

Parameters

- **fr** – Flux ratio [-]

- **tstar** – Effective stellar temperature [K]
- **a** – Scaled semi-major axis [Rs]
- **A** – Bond albedo [-]
- **l** – Wavelength [m]
- **r** – Inverse of the phase integral [-]
- **ti** – Temperature [K]

Returns **A**

Return type Bond albedo

`pytransit.utils.phasecurves.`**`solve_redistribution`**(*fr*, *tstar*, *a*, *ab*, *l*)

Solve the redistribution factor.

Parameters

- **fr** – Flux ratio [-]
- **tstar** – Effective stellar temperature [K]
- **a** – Scaled semi-major axis [Rs]
- **ab** – Bond albedo [-]
- **l** – Wavelength [m]
- **r** – Inverse of the phase integral [-]
- **Ti** (*t*) – Temperature [K]

Returns **f**

Return type Redistribution factor

`pytransit.utils.phasecurves.`**`solve_teq`**(*fr*, *tstar*, *a*, *ab*, *l*, *r=1.5*, *ti=0*)

Solve the equilibrium temperature.

Parameters

- **fr** – Flux ratio
- **tstar** – Effective stellar temperature [K]
- **a** – Scaled semi-major axis [Rs]
- **ab** – Bond albedo
- **l** – Wavelength [m]
- **r** – Inverse of the phase integral
- **ti** – Temperature [K]

Returns **Teq** – Equilibrium temperature

Return type float or ndarray

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Bibliography

[Loeb2003] Loeb, A. & Gaudi, B. S. Periodic Flux Variability of Stars due to the Reflex Doppler Effect Induced by Planetary Companions. Astrophys. J. 588, L117–L120 (2003).

[Bloemen2010] Bloemen, S. et al. Kepler observations of the beaming binary KPD 1946+4340. MNRAS 410, (2010).

[Barclay2012] Barclay, T. et al. PHOTOMETRICALLY DERIVED MASSES AND RADII OF THE PLANET AND STAR IN THE TrES-2 SYSTEM. AspJ 761, 53 (2012).

[Lillo-Box2014] Lillo-Box, J. et al. Kepler-91b: a planet at the end of its life. A&A 562, A109 (2014).

Python Module Index

p

Index

Symbols

A

B

C

D

E

F

G

I

L

O

P

Q

R

S

T

U