
TransitFit

Release 3.4.0

Joshua Hayes and collaborators

Dec 11, 2023

CONTENTS

1	Installation	3
2	Citing	5
3	Observational Data	7
3.1	Getting Started	7
3.2	Config Files	10
3.3	Limb-darkening	13
3.4	Detrending	15
3.5	Fitting large number of parameters	18
3.6	Allowing for TTVs	19
3.7	FAQs	19
3.8	API	20
	Python Module Index	55
	Index	57

TransitFit is a Python 3.X package for fitting multi-telescope, multi-filter, and multi-epoch exoplanetary transit observations. It uses the [batman](#) transit model and nested sampling routines from [dynesty](#).

TransitFit can include in its likelihood calculations the effect of host star characteristics and observation filter profiles on limb-darkening coefficients (LDCs), which we refer to as '*coupling*' the LDCs. It can also perform per-telescope detrending simultaneously with the fitting of other parameters.

See [Getting Started](#) for instructions on how to use TransitFit. You can also find more information in the [TransitFit paper](#)

INSTALLATION

TransitFit is compatible with Python 3.X installations. To run, TransitFit requires the following packages:

- numpy
- scipy
- pandas
- matplotlib
- corner
- [batman](#)
- [dynesty](#)
- [ldtk](#)

To install the most recent stable version, run:

```
pip install transitfit
```

Alternatively, you can install it direct from the source by downloading the project from the [GitHub page](#) and running:

```
pip install -e .
```


CITING

If you find TransitFit useful in your work, please cite the [accompanying paper](#). If you are using BibTeX, you can use the following citation in your .bib file:

```
@article{10.1093/mnras/stad3353,  
author = {Hayes, J J C and Priyadarshi, A and Kerins, E and Awiphan, S and McDonald, I,  
→and A-thano, N and Morgan, J S and Humpage, A and Charles, S and Wright, M and Joshi,  
→Y C and Jiang, Ing-Guey and Inyanya, T and Padjaroen, T and Munsaket, P and  
→Chuanraksasat, P and Komonjinda, S and Kittara, P and Dhillon, V S and Marsh, T R and  
→Reichart, D E and Poshyachinda, S and The SPEARNET Collaboration},  
title = "{TransitFit: combined multi-instrument exoplanet transit fitting for JWST, HST,  
→and ground-based transmission spectroscopy studies}",  
journal = {Monthly Notices of the Royal Astronomical Society},  
pages = {stad3353},  
year = {2023},  
month = {11},  
abstract = "{We present TransitFit, a package designed to fit exoplanetary transit  
→light-curves. TransitFit offers multi-epoch, multi-wavelength fitting of multi-  
→telescope transit data. TransitFit allows per-telescope detrending to be performed  
→simultaneously with transit parameter fitting, including custom detrending. Host limb  
→darkening can be fitted using prior conditioning from stellar atmosphere models. We  
→demonstrate TransitFit in a number of contexts. We model multi-telescope broadband  
→optical data from the ground-based SPEARNET survey of the low-density hot-Neptune WASP-  
→127 b and compare results to a previously published higher spectral resolution GTC/  
→OSIRIS transmission spectrum. Using TransitFit, we fit 26 transit epochs by TESS to  
→recover improved ephemeris of the hot-Jupiter WASP-91 b and a transit depth determined  
→to a precision of 111 ppm. We use TransitFit to conduct an investigation into the  
→contested presence of TTV signatures in WASP-126 b using 180 transits observed by TESS,  
→concluding that there is no statistically significant evidence for such signatures  
→from observations spanning 27 TESS sectors. We fit HST observations of WASP-43 b,  
→demonstrating how TransitFit can use custom detrending algorithms to remove complex  
→baseline systematics. Lastly, we present a transmission spectrum of the atmosphere of  
→WASP-96 b constructed from simultaneous fitting of JWST NIRISS Early Release  
→Observations and archive HST WFC3 transit data. The transmission spectrum shows  
→generally good correspondence between spectral features present in both datasets,  
→despite very different detrending requirements.}",  
issn = {0035-8711},  
doi = {10.1093/mnras/stad3353},  
url = {https://doi.org/10.1093/mnras/stad3353},  
eprint = {https://academic.oup.com/mnras/advance-article-pdf/doi/10.1093/mnras/stad3353/  
→52799695/stad3353.pdf},
```

}

OBSERVATIONAL DATA

The observational data of WASP-127b obtained by SPEARNET and the TRANSITFIT transmission spectrum of WASP-96b can be downloaded from [here](#) and [here](#).

3.1 Getting Started

TransitFit operation is based around three *config files* (the *data path file*, the *priors file* and the *filter info file*) and a single Python wrapper function (`run_retrieval()`). There's a lot more going on under the hood but, for everyday use, you don't need to worry about that.

3.1.1 A note on time standards

TransitFit assumes that all time values are in BJD. If you are using BJD-2450000, or any variation on this, you will be okay as long as you are consistent. Using HJD, local time, or any time system that is not based on BJD will end in incorrect results and probably tears.

3.1.2 An illustration

Some toy observations

Let's assume we have observations of 3 transits of a planet with a 6-day period:

Observation 1

Taken with telescope A, using an R-band filter, on Day 0.

Observation 2

Taken with telescope A, using a filter with uniform transmission between 400nm and 500nm, on Day 12.

Observation 3

Taken with telescope B, using an R-band filter, on Day 42.

The config files for this might look something like

- 'input_data.csv':

Path,	Telescope,	Filter,	Epoch,	Detrending
/path/to/observation1,	0,	0,	0,	0
/path/to/observation2,	0,	1,	1,	0
/path/to/observation3,	1,	0,	2,	0

- 'priors.csv':

Parameter	Distribution	InputA	InputB	Filter
P,	gaussian,	6,	0.001,	
t0,	gaussian,	2457843.45246,	0.007,	
a,	gaussian,	7.64,	0.5,	
inc,	gaussian,	88.5,	1.2,	
rp,	uniform,	0.13,	0.19,	0
rp,	uniform,	0.13,	0,19,	1
ecc,	fixed,	0,	,	

- 'filter_profiles.csv':

Filter index,	InputA,	InputB
0,	R,	
1,	400,	500

See [here](#) for more information on exactly what these mean and how to set them up.

Running TransitFit on these data

Once we have the config files set up, it is incredibly easy to retrieve a best-fit model using TransitFit.

To fit these observations with basic linear detrending and a bog-standard quadratic limb darkening model without using the *LDC coupling offered by* TransitFit, run:

```
from transitfit import run_retrieval

results = run_retrieval('input_data.csv, priors.csv')
```

This is a simple approach which is useful for getting some preliminary results, as it is generally the fastest-running method. However, TransitFit can do better.

Let's now imagine we have decided to use a quadratic detrending model, and also want to take advantage of the *coupled LDC fitting*. To do this, we still use `run_retrieval()`, but have to specify a few more arguments, including providing filter profiles and host information.:

```
from transitfit import run_retrieval

# Set up the host info, using arbitrary values.
# These are all given in (value, uncertainty) tuples
host_T = (5450, 130) # Effective temperature in Kelvin
host_z = (0.32, 0.09) # The metallicity
host_r = (1.03, 0.05) # Host radius in solar radii - this MUST be supplied if the prior_
↳ for orbital separation is in AU.
host_logg = (4.5, 0.1) # log10(suface gravity) in cm/s2

# Set up the detrending model
# We want to use a quadratic (2nd order) model
detrending_models = [['nth order', 2]]

# Now we can run the retrieval!
results = run_retrieval('input_data.csv', 'priors.csv', 'filter_profiles.csv', # Config_
↳ paths

                        detrending_list=detrending_models, # Set up detrending models
                        ld_fit_method='coupled' # Turn on coupled LDC fitting
```

(continues on next page)

(continued from previous page)

```

→ # host params)      host_T=host_T, host_logg=host_logg, host_z=host_z, host_r=host_r

```

3.1.3 TransitFit outputs

TransitFit provides a variety of outputs. These are:

Output files

These are .csv files which record the best fit value and uncertainty for each parameter fitted. The base folder in which they are saved is controlled by the `results_output_folder` argument of `run_retrieval()`.

There are a few different versions of these:

- `'complete_output.csv'` - this contains the complete final results of the entire retrieval, collating results from all *stages of retrieval* if required.
- `summary_output_file` - this contains the best fit results from a specific stage of retrieval (e.g. if in *'folded mode'*, this would be the results for a specific filter (stage 1) or the results for fitting the folded curves (stage 2)). The name of this file can be specified with the `summary_file` argument of `run_retrieval()` and defaults to `'summary_output.csv'`.
- `full_output_file` - This contains every output from a given batched run, including indication of which batch the results come from. The name of this file can be specified with the `full_output_file` argument of `run_retrieval()` and defaults to `'full_output.csv'`.
- `modified_output.csv` - This contains different upper and lower bound of errors on the best fit values. It is recommended to use these results as it handles the asymmetric distribution of sampled parameters. We define upper limit on the best fit value as the 68.27 quantile of the weighted samples beyond the best fit value, and the lower limit as 31.73 quantile of weighted samples below the best fit.

Fitted light curves

These are .csv files for each input light curve, containing the normalised and detrended light curves, along with phase values and best-fit models. The columns for these are

1. **Time** - The time of observation
2. **Phase** - The phase of observation, setting mid-transit to a phase of 0.
3. **Normalised flux** - The detrended and normalised flux values
4. **Flux uncertainty** - The uncertainty on the normalised flux
5. **Best fit curve** - The normalised best-fit light curve.

By default these are saved in the `'./fitted_lightcurves'` folder, but this can be changed using the `final_lightcurve_folder` argument of `run_retrieval()`.

Plots

TransitFit has a few different plots that it provides. The default base folder for these is `'./plots'` and this can be set using `plot_folder='./plots'`. The different plot types are:

Fitted light curves

These are given for each individual light curve. Additionally, if running in *'folded mode'*, a folded curve for each filter is produced. These plots come in versions with and without error bars.

Posterior samples

These are made using corner and show the samples for each run of dynesty within a TransitFit retrieval.

Quick-look folded curves

Since running in *'folded mode'* can take some time, TransitFit provides a 'quick-look' plot for each filter after folding. This is there mostly so that you can be satisfied that the folding makes sense, rather than having to wait until the end to find out something has gone wrong.

3.1.4 Error Scaling

The likelihood function for the fitting includes an additional error-rescaling term as described in the [online documentation](#) for EMCEE2 (Foreman-Mackey et al. 2013). This allows for cases where the flux errors might be underestimated. This can be specified by adding 'error_scaling' while calling `run_retrieval()`. The limits of this parameter can also be specified by adding 'error_scaling_limits'.

3.1.5 Median Normalisation

In some cases, users may benefit from using median-normalisation. This can be specified by adding 'median_normalisation' while calling `run_retrieval()`. This normalises the raw lightcurves by the median value before attempting any fitting.

3.2 Config Files

There are three configuration files required by TransitFit. These are the *data input file*, the *priors file* and the *filter info file*. These are all fairly prescriptive in their format, and should be provided as .csv files with one header row.

For the purposes of illustration, we will be using an example where we have observations of 3 transits of a planet with a 6-day period:

Observation 1

Taken with telescope A, using an R-band filter, on Day 0.

Observation 2

Taken with telescope A, using a filter with uniform transmission between 400nm and 500nm, on Day 12.

Observation 3

Taken with telescope B, using an R-band filter, on Day 42.

We will also assume that we are going to only use one *detrending model*.

3.2.1 Indexing

In order to allow for full flexibility, TransitFit operates on a zero-indexed indexing system. Each light curve is identified by three indices: **telescope**, **filter**, and **epoch**. The actual ordering of these indices does not matter, as long as they are consistent across all light curves being fitted, and no values are skipped. A useful practice is to order observation epochs chronologically (0 being the first observation) and filters in either ascending or descending order of wavelength.

This approach makes it possible to easily fit (for example) two simultaneous observations with the same filter from two different sites, or single wavelength, multi-epoch observations from a single observatory, or any other combination you can think of.

Since TransitFit offers the ability to use multiple different detrending models simultaneously, a **detrending method** index is also required. This is particularly useful in situations where space- and ground-based observations are being combined, since space-based observations often require something more complex than an nth-order approach. This is discussed in more depth [here](#).

So, to summarise, each transit observation in the dataset that you are fitting is identified by a **telescope**, **filter**, and **epoch** index and the detrending model is controlled by a **detrending method** index.

3.2.2 Data input file

This file is used to direct `TransitFit` to the light curve observations that you want to fit. It is also where the *indexing scheme* for the light curves is defined. Each transit observation to be fitted should be a separate text (.txt or .csv) file, with three columns: time (in BJD), flux, and uncertainty on flux. Note that the light curves do not have to be normalised or detrended, as this is *something that TransitFit can handle*. The data input file should have 5 columns:

1. **Path**: this is the absolute or relative path to the data file for each light curve.
2. **Telescope index**: The index associated with the telescope used to take this observation.
3. **Filter index**: The index which identifies the filter used in this observation.
4. **Epoch index**: The index which identifies the epoch of this transit. Note that this does not reflect the number of transits that have passed. For example, if you have two observations of a planet with a 5 day orbit, taken a year apart, the indices would be 0 and 1, **not** 0 and 72.
5. **Detrending method index**: This is the index used to choose which detrending method you want to use. See [here](#) for more info.

In the case of our example above, the `input_data.csv` file will look something like:

Path,	Telescope,	Filter,	Epoch,	Detrending
/path/to/observation1,	0,	0,	0,	0
/path/to/observation2,	0,	1,	1,	0
/path/to/observation3,	1,	0,	2,	0

3.2.3 Priors

This file determines which physical parameters are to be fitted by `TransitFit`, and the distribution from which samples are to be drawn from for each. It can also be used to manually fix a parameter to a specific value. This file should also have 5 columns:

1. **Parameter**: The parameters which can be set using the priors file are
 - **P**: Period of the orbit, in BJD
 - **t0**: time of inferior conjunction in BJD
 - **a** : semi-major axis. This can be given in units of either host-radii or AU. If given in AU, then `host_r` must be specified in `run_retrieval()` to allow for a conversion to host-radii.
 - **inc**: inclination of the orbit in degrees (Defaults to 90 degrees if not provided). `TransitFit` automatically sets an upper limit of 90 degrees.
 - **ecc**: eccentricity of the orbit (defaults to 0 if not provided)
 - **w**: longitude of periastron (in degrees) (Defaults to 90 degrees if not provided)
 - **rp**: Planet radius in stellar radii (i.e. R_p/R_*). **Note**: if you have multiple filters that you want to fit **rp** for, you will have to provide a prior for *each* filter.
 - **{q0, q1, q2, q3}** : Kipping q parameters for limb darkening. Most of the time you will not need to set these, but if you want to run a retrieval without fitting for limb darkening (if, for example, you fitted for these another way), then you can set them here by specifying a 'fixed' distribution. Note that you will also have to set `ld_fit_method='off'` in the arguments of `run_retrieval()`.

2. **Distribution:** The distribution that samples will be drawn from. This can be any of:
 - **uniform** - uses a uniform, box-shaped prior
 - **gaussian** - uses a Gaussian prior
 - **fixed** - the parameter won't be fitted and will be fixed at a user-specified value.
3. **Input A:** The use of this column depends on the distribution being used:
 - If **uniform**: provide the **lower bound** of the uniform distribution.
 - If **gaussian**: provide the **mean** of the Gaussian distribution.
 - If **fixed**: provide the value to fix the parameter at.
4. **Input B:** The use of this column depends on the distribution being used:
 - If **uniform**: provide the **upper bound** of the uniform distribution.
 - If **gaussian**: provide the **standard deviation** of the Gaussian distribution.
 - If **fixed**: this input is not used and anything here will be ignored.
5. **Filter index:** If a parameter varies with wavelength (i.e. **rp** and limb-darkening coefficients), the filter index must be supplied for each instance in the priors file, making sure to follow the indexing set out in the data paths and filter info files.

So, for our example observations, if we assume a circular orbit (i.e. don't fit for **ecc** and **w**), our 'priors.csv' file might look something like:

Parameter	Distribution	InputA	InputB	Filter
P	gaussian	6	0.001	
t0	gaussian	2457843.45246	0.007	
a	gaussian	7.64	0.5	
inc	gaussian	88.5	1.2	
rp	uniform	0.13	0.19	0
rp	uniform	0.13	0.19	1
ecc	fixed	0		

When setting up your priors, we recommend that you use a uniform distribution for **rp** so that you don't inadvertently bias the values, especially if you're doing spectroscopy work.

3.2.4 Filter profiles

This file is used to specify the filter profiles that observations were made at, and is only required if you are using TransitFit's *ability to couple LDCs across wavelengths*.

TransitFit can deal with either uniform box filters (useful for narrow-band spectroscopy), or full filter response functions. It comes pre-packaged with a set of standard filters:

- Johnson-Cousins *UVRIB*
- SLOAN-SDSS *u'g'r'i'z'*
- The *TESS* filter
- The *Kepler* filter

If you want to use your own filter profile, you can provide a .csv with 2 columns: wavelength in nm, and filter transmission, either as a fraction or percentage (TransitFit will detect which).

The filter info file requires 3 columns:

1. **Filter index:** the index of the filter, ensuring consistency with the other input files.
2. **Input A:**
 - For a uniform box filter, provide the **lowest wavelength** not blocked by the filter in nanometres.
 - The name of one of the provided filter profiles: any of: U, V, R, I, B, u', g', r', i', z', TESS, Kepler.
 - The path to a user-provided filter profile
3. **Input B:**
 - For a uniform box filter, provide the **highest wavelength** not blocked by the filter in nanometres.
 - For anything else, this column is ignored.

So, for our example, `filter_profiles.csv` would look like:

Filter index,	InputA,	InputB
0,	R,	
1,	400,	500

3.3 Limb-darkening

`TransitFit` was built with two primary motivations. First, to facilitate transmission spectroscopy surveys using observations from heterogeneous telescopes, and second, to allow the user to fit light curves while accounting for the effects that filter profiles and host parameters have on the LDCs, which we refer to as ‘coupling’ the LDCs. We will discuss the latter here.

The full justification for and impacts of this approach can be found in the [accompanying paper](#) but, in short, not including host parameters and filter profiles in likelihood calculations of LDCs can lead to biases in your measurements of R_p/R_\star of tens of percent. By including this, `TransitFit` has made it easy to conduct robust transmission spectroscopy studies using observations from heterogeneous sources.

3.3.1 Calculating LDC likelihoods

`TransitFit` uses the [Limb Darkening Toolkit \(LDTk\)](#) to calculate the likelihood values of sets of LDCs given the host characteristics and filter profiles. These values are then included in the likelihood calculations of transit models during retrieval.

In order to use this feature, the user must provide the following:

- Host temperature
- Host z
- Host $\log(g)$
- A filter profile for each filter used in the observations.

The first three are provided as arguments to `run_retrieval()`, and the filter profiles are specified using the *filter profiles input file*.

Available limb-darkening models

Typically, stellar intensity profiles are described by analytical functions $I_\lambda(\mu)$, where μ is the cosine of the angle between the line of sight and the emergent intensity. μ can also be expressed as $\mu = \sqrt{1 - r^2}$ where r is the unit-normalised radial coordinate on the stellar disk, and as such, all limb-darkening models must be valid for $0 \leq \mu < 1$.

There are 5 limb darkening models provided by TransitFit, which can be selected using the `limb_darkening_model` argument in `run_retrieval()`. These are:

- **'linear'** - the linear law given by

$$\frac{I(\mu)}{I(1)} = 1 - u_{0,l}(1 - \mu)$$

- **'quadratic'** - the quadratic law given by

$$\frac{I(\mu)}{I(1)} = 1 - u_{0,q}(1 - \mu) - u_{1,q}(1 - \mu)^2$$

- **'squareroot'** - the square-root law given by

$$\frac{I(\mu)}{I(1)} = 1 - u_{0,\text{sqrt}}(1 - \mu) - u_{1,\text{sqrt}}(1 - \sqrt{\mu})$$

- **'power2'** - the power-2 law given by

$$\frac{I(\mu)}{I(1)} = 1 - u_{0,p2}(1 - \mu^{u_{1,p2}})$$

- **'nonlinear'** - the non-linear law given by

$$\begin{aligned} \frac{I(\mu)}{I(1)} = & 1 - u_{0,\text{nl}}(1 - \mu^{1/2}) - u_{1,\text{nl}}(1 - \mu) \\ & - u_{2,\text{nl}}(1 - \mu^{3/2}) - u_{3,\text{nl}}(1 - \mu^2). \end{aligned}$$

where each of u_0 , u_1 , u_2 , and u_3 are the limb-darkening coefficients to be fitted. With the exception of the non-linear law, all of these models are constrained to physically-allowed values by the method in [Kipping \(2013\)](#), which we have extended to include the power-2 law.

LDC Fitting modes

TransitFit offers three modes for LDC fitting, which can be selected using the `ld_fit_method` argument in `run_retrieval()`:

- **'independent'**

This is the traditional approach of fitting LDCs for each filter separately. TransitFit still uses the [Kipping parameterisations](#), but LDTk is not used to couple LDCs across filters.

- **'coupled'**

Using the Kipping parameterisations, each LDC is fitted as a free parameter, with LDTk being used to estimate the likelihood of sets of LDCs, using information on the host star and the observation filters. To use the coupled mode, the filter response file should have atleast 4 individual datapoints. TransitFit also provides the functionality to use the uncertainty multiplier from LDTk. This can be specified by adding `'ldtk_uncertainty_multiplier'` while calling `run_retrieval()`.

- 'single'

When fitting with multiple wavebands, the number of parameters required to be fitted can increase dramatically. The 'single' LDC fitting mode freely fitting LDC for only one filter, and uses LDTk to extrapolate LDC values for the remaining filters. The i -th coefficient of a filter f , is calculated as

$$c_{i,f} = u_i \times \frac{\langle c_{i,f} \rangle}{\langle u_i \rangle}$$

where u_i is the sampled value of the i -th LDC in the actively fitted filter, and $\langle c_{i,f} \rangle$ and $\langle u_i \rangle$ are the maximum likelihood values initially suggested by LDTk.

3.4 Detrending

TransitFit has the capability to detrend light curves simultaneously with fitting physical parameters, and can handle using both n th-order polynomial and user-specified detrending functions. It is able to fit multiple detrending models for different observations at once, which is particularly useful when combining observations from different telescopes which have known systematic properties.

For n th-order detrending, we assume that the detrending is additive, and that the detrended flux, $\mathbf{D}(\mathbf{t})$, is given by

$$\mathbf{D}(\mathbf{t}) = \mathbf{F}(\mathbf{t}) - \mathbf{d}(\mathbf{t})$$

where $\mathbf{F}(\mathbf{t})$ is the raw flux and $\mathbf{d}(\mathbf{t})$ is the detrending function. However, if you have a more complicated detrending function which has multiplicative elements, or value which depend on the actual flux, TransitFit can use a custom model to do this.

We will look here at how to get TransitFit to use the different types of detrending, and show a simple example of setting up a custom detrending function.

3.4.1 Basic detrending syntax

Setting up TransitFit to use different detrending models is simple and uses the `detrending_list` kwarg in `run_retrieval()`. This is a list of the different detrending methods to be used, along with any required details. The *detrending indices* given in the *data input file* refer to the index of the methods in this list.

Detrending methods

The available types of detrending are:

Nth order

To use a polynomial of order n , the entry to `detrending_list` should be given as `['nth order', n]`. The n th order polynomials used by TransitFit are designed to be flux-conserving at the time of conjunction, and are of the form

$$d(t_i) = \sum_{j=1}^n \left[a_j (t_i - t_0)^j \right]$$

The full derivation of this can be found in the [paper](#)

Custom function

Using a custom function requires a little more information. By default, all parameters are assumed to be global: that is, there is a single value for each parameter which applies to *all light curves with this detrending model*. There are situations where some parameters in a detrending function should not be fitted globally. We define three cases of this:

- “**telescope dependent**” parameters - If a parameter is telescope dependent, then a different value will be fitted for each telescope index;
- “**wavelength dependent**” parameters - If a parameter is filter dependent, then a different value will be fitted for each filter index;
- “**epoch dependent**” parameters - If a parameter is epoch dependent, then a different value will be fitted for each epoch index.

Custom detrending functions must take a `LightCurve()` as their first argument, and each argument after that must be a float. It must return the detrended flux values. Aside from this, there are no major restrictions to the type of detrending you can use.

Let’s assume that we want to use the following arbitrary detrending function:

```
def f(lightcurve, a, b, c, t0, P):
    # t0 is the time of conjunction and P is the period

    times = lightcurve.times

    detrending_vals = times - a * exp(-b * times) + c
    detrended_flux = lightcurve.flux - detrending_vals
    return detrended_flux
```

and that `c` is some wavelength dependent parameter.

The general syntax to use for a custom detrending function `f()` is:

```
['custom', f, [telescope dependent parameters], [wavelength dependent parameters], ↵
↵[epoch dependent parameters]]
```

To specify that a parameter is telescope-, wavelength-, or epoch-dependent, add the index of the relevant argument to the appropriate list. In our example, our entry for `c` being wavelength dependent would be:

```
['custom', f, [], [3], []]
```

No detrending

To not detrend a light curve, use `['off']` in your `detrending_list`

Setting limits on detrending coefficients

By default, all detrending coefficients are fitted using a uniform prior of ± 10 . Obviously this is not always ideal, so you can specify the range over which these priors should be fitted using the `detrending_limits` argument in `run_retrieval()`. **Note:** all the detrending coefficients in a given model will be bound to the same range.

To use custom ranges in your detrending models, you use a list where each entry is `[lower, upper]` for the detrending methods.

An example

Let's again consider our *toy model* with three observations. We shall assume that we want to apply a quadratic detrending model to one, the custom detrending model above to another, and that the last one has already been detrended in pre-processing. We will also change the coefficient bounds. We first need to edit our 'input_data.csv' to:

Path,	Telescope,	Filter,	Epoch,	Detrending
/path/to/observation1,	0,	0,	0,	0
/path/to/observation2,	0,	1,	1,	1
/path/to/observation3,	1,	0,	2,	2

and then our full input code, using the coupled LDC fitting, becomes:

```
from transitfit import run_retrieval

# Set up the custom detrending function
def f(times, a, b, c, t0, P):
    return times - a * exp(-b * times) + c

# Set up the host info, using arbitrary values.
# These are all given in (value, uncertainty) tuples
host_T = (5450, 130) # Effective temperature in Kelvin
host_z = (0.32, 0.09) # The metalicity
host_r = (1.03, 0.05) # Host radius in solar radii - this MUST be supplied if the prior_
↳ for orbital separation is in AU.
host_logg = (4.5, 0.1) # log10(surface gravity) in cm/s2

# Set up the detrending models
detrending_models = [['nth order', 2], # This is detrending index 0
                    ['custom', f, [3], [], []], # This is detrending index 1
                    ['off']] # This is detrending index 2

# Set the detrending coefficient bounds
detrending_limits = [[-10, 10], # bounds for model 0
                    [-3, 20], # bounds for model 1
                    [0.2, 4.8]] # bounds for model 2

# Now we can run the retrieval!
results = run_retrieval('input_data.csv', 'priors.csv', 'filter_profiles.csv', # Config_
↳ paths
                      detrending_list=detrending_models, # Set up detrending models
                      detrending_limits=detrending_limits # Set the detrending limits
                      ld_fit_method='coupled' # Turn on coupled LDC fitting
                      host_T=host_T, host_logg=host_logg, host_z=host_z, host_r=host_r_
↳ # host params)
```

In case of a single detrending model, please ensure that the format is maintained as::

```
detrending_models = [['custom', f, [3], [], []]]
```

3.5 Fitting large number of parameters

In an ideal world, a transmission spectrum would be made up of very many light curve observations, at many filters and epochs. Fitting all of these observations simultaneously would result in a very high-dimensional parameter space, which leads to instabilities in the nested sampling.

In order to avoid this, TransitFit has modes for dealing with large numbers of parameter sets: 'batched', and 'folded', which can be set with the `fitting_mode` argument of `run_retrieval()`. There is also 'all' mode, which can be set to manually force all parameters to be fitted simultaneously.

3.5.1 'Batched' fitting

This mode groups the light curves by filter, and then splits the retrieval into multi-filter 'batches,' fitting each of these batches one at a time. The batches are chosen to allow a maximum number of parameters to be fitted simultaneously, which can be controlled with the `max_batch_parameters` argument of `run_retrieval()`. Final best-fit values are then calculated from weighted means of the best-fit values from each batch.

The batches are generally constructed to have at least one filter in common with at least one other batch. This ensures that there is still some coupling of information between the batches. The exception to this is when there is one filter in particular which has a very high number of observations. In this case, we recommend using the 'folded' mode.

It is suggested to run the model twice if using 'batched' mode. For the second run, use the output from first run as the priors for the wavelength independent parameters (P, t0, a, inc). The error limits, standard deviation on these priors should be narrowed down to not allow for much variation in individual batches/lightcurves.

3.5.2 'Folded' fitting

With the launch of large surveys such as *TESS*, many exoplanets have multiple-epoch observations in a single filter. TransitFit can make use of these through a two-step retrieval process.

1. TransitFit runs a retrieval on each filter independently, and uses the results to produce a phase-folded light curve for each filter.
2. TransitFit runs a standard multi-wavelength retrieval using the batched algorithm above.

This mode of retrieval allows for the production of high-quality folded light curves from non-detrended data, as well as providing a method where observations from long-term, single-waveband surveys such as *TESS* can be easily combined with single-epoch observations at multiple wavelengths, such as from ground-based spectrographic follow-up.

3.5.3 Automatic mode selection

By default, TransitFit will try and detect which fitting mode is most appropriate for your data. It first works out how many parameters are required to fit everything simultaneously, which we will call `max_n_params`. Then if:

1. `max_n_params <= max_batch_parameters` - all parameters are fitted simultaneously ('all' mode).
2. If any filter has 3 or more epochs observed, then 'folded' mode is used.
3. Otherwise, 'batched' mode is used.

3.6 Allowing for TTVs

In its default mode, TransitFit assumes that there are no TTVs and fits a single t_0 and P that are assumed applicable to all transits. In the situation where TTVs are present, TransitFit can fit t_0 to each individual epoch, allowing then for O-C investigations.

In order to do this, set `allow_TTV=True` in the arguments of `run_retrieval()`. **Note:** TransitFit cannot automatically detect if there are TTVs present in the data. You must explicitly enable this mode.

When `allow_TTV=True`, TransitFit cannot fit for the period value, and this must be set as a fixed value in the *data input file*. In order to be consistent, we recommend that any investigation into TTVs in a system should have 2 stages:

1. Run TransitFit with `allow_TTV=False`.
2. Run TransitFit with `allow_TTV=True`, fixing the period at the best fit value from the first step.

3.7 FAQs

What prior distributions should I use for my rp values?

Obviously your use case will determine a lot of this, but, as some helpful rules of thumb, we recommend that you use a wide uniform prior for your rp values, rather than a Gaussian based on previous measurements.

Why do I have to use config files? Isn't that a bit outdated?

Fair point. We've based this around config files because when being used on hundreds of light curves at once, typing all the inputs into a list becomes very difficult to read and keep track of. In future updates, we might work on streamlining the API to allow inputs to be set directly within the code by the user.

How do I cite TransitFit?

If you find TransitFit useful in your work, please cite the [accompanying paper](#). If you are using BibTeX, you can use the following citation in your .bib file:

```
@article{10.1093/mnras/stad3353,
author = {Hayes, J J C and Priyadarshi, A and Kerins, E and Awiphan, S and McDonald,
↪ I and A-thano, N and Morgan, J S and Humpage, A and Charles, S and Wright, M and
↪ Joshi, Y C and Jiang, Ing-Guey and Inyanya, T and Padjaroen, T and Munsaket, P
↪ and Chuanraksasat, P and Komonjinda, S and Kittara, P and Dhillon, V S and Marsh,
↪ T R and Reichart, D E and Poshyachinda, S and The SPEARNET Collaboration},
title = "{TransitFit: combined multi-instrument exoplanet transit fitting for JWST,
↪ HST and ground-based transmission spectroscopy studies}",
journal = {Monthly Notices of the Royal Astronomical Society},
pages = {stad3353},
year = {2023},
month = {11},
abstract = "{We present TransitFit, a package designed to fit exoplanetary transit
↪ light-curves. TransitFit offers multi-epoch, multi-wavelength fitting of multi-
↪ telescope transit data. TransitFit allows per-telescope detrending to be
↪ performed simultaneously with transit parameter fitting, including custom
↪ detrending. Host limb darkening can be fitted using prior conditioning from
↪ stellar atmosphere models. We demonstrate TransitFit in a number of contexts. We
↪ model multi-telescope broadband optical data from the ground-based SPEARNET
↪ survey of the low-density hot-Neptune WASP-127 b and compare results to a
↪ previously published higher spectral resolution GTC/OSIRIS transmission spectrum.
↪ Using TransitFit, we fit 26 transit epochs by TESS to recover improved ephemeris
↪ of the hot-Jupiter WASP-91 b and a transit depth determined to a precision of"
```

(continues on next page)

(continued from previous page)

```

→111 ppm. We use TransitFit to conduct an investigation into the contested,
→presence of TTV signatures in WASP-126 b using 180 transits observed by TESS,
→concluding that there is no statistically significant evidence for such
→signatures from observations spanning 27 TESS sectors. We fit HST observations of
→WASP-43 b, demonstrating how TransitFit can use custom detrending algorithms to
→remove complex baseline systematics. Lastly, we present a transmission spectrum
→of the atmosphere of WASP-96 b constructed from simultaneous fitting of JWST
→NIRISS Early Release Observations and archive HST WFC3 transit data. The
→transmission spectrum shows generally good correspondence between spectral
→features present in both datasets, despite very different detrending requirements.
→}",
issn = {0035-8711},
doi = {10.1093/mnras/stad3353},
url = {https://doi.org/10.1093/mnras/stad3353},
eprint = {https://academic.oup.com/mnras/advance-article-pdf/doi/10.1093/mnras/
→stad3353/52799695/stad3353.pdf},

```

}

Why have you only included nested sampling? Why can't I use MCMC?

This comes down partly to the personal preference of the development team, but mostly because TransitFit often has to deal with high-dimensioned fitting problems. MCMC routines often struggle in this situation, especially when the posterior space can be fairly degenerate or spiky. Nested sampling can handle these situations in a more stable way.

I've found a bug - what can I do?

Raise the issue on the [GitHub page](#). Make sure to include as much information as possible, including any trace-back messages and information on your priors etc.

Can I contribute to the project?

Absolutely! TransitFit is an open-source project (GPL-3.0 licence) - please raise a pull request for any additions, changes, or improvements want to suggest.

3.8 API

This page details the methods and classes provided by the TransitFit module.

3.8.1 Top-Level Interface

A function which will run everything when given a path to light curve and priors!

```
transitfit._pipeline.run_retrieval(data_files, priors, filter_info=None, detrending_list=[['nth order', 1]],
                                  limb_darkening_model='quadratic', ld_fit_method='independent',
                                  fitting_mode='auto', max_batch_parameters=25, batch_overlap=2,
                                  host_T=None, host_logg=None, host_z=None, host_r=None,
                                  nlive=300, dlogz=None, maxiter=None, maxcall=None,
                                  dynesty_sample='rslice', dynesty_bounding='multi', normalise=True,
                                  detrend=True, results_output_folder='./output_parameters',
                                  final_lightcurve_folder='./fitted_lightcurves',
                                  summary_file='summary_output.csv',
                                  full_output_file='full_output.csv', plot_folder='./plots', plot=True,
                                  marker_color='dimgrey', line_color='black', ldtk_cache=None,
                                  ldtk_samples=20000, do_ld_mc=False, data_skiprows=0,
                                  allow_ttv=False, filter_delimiter=None, detrending_limits=None,
                                  normalise_limits=None, bin_data=False, cadence=2,
                                  binned_color='red', walks=100, slices=10, n_procs=1,
                                  check_batchsizes=False, median_normalisation=False,
                                  error_scaling=False, error_scaling_limits=None,
                                  ldtk_uncertainty_multiplier=1.0)
```

Runs a full retrieval of posteriors using nested sampling on a transit light curve or a set of transit light curves. For more guidance on the use of input files and structuring, see the TransitFit documentation.

Parameters

- **data_files** (*str*) – The path to the data input .csv file, which contains the paths to the light curve data.
- **priors** (*str*) – Path to a .csv file containing prior information on each variable to be fitted.
- **filter_info** (*str*, *optional*) – Path to a .csv file containing information on the wavelengths of the filters that observations were made at.

This is required if `ld_fit_method` is `'single'` or `'coupled'`. If not `None` and `host_T`, `host_logg` and `host_z` are not `None`, retrieval will include fitting realistic limb darkening parameters for the filters. Default is `None`.

- **detrending_list** (*array_like*, *shape* (`n_detrending_models`, 2)) – A list of different detrending models. Each entry should consist of a method and a second parameter dependent on the method. Accepted methods are

```
['nth order', order]
['custom', function, [global fit indices], [filter fit indices],
[epoch fit indices]]
['off', ]
```

`function` here is a custom detrending function. TransitFit assumes that the first argument to this function is times and that all other arguments are single-valued - TransitFit cannot fit list/array variables. If `'off'` is used, no detrending will be applied to a light curve using this model.

If a custom function is used, and some inputs to the function should not be fitted individually for each light curve, but should instead be shared either globally, within a given filter, or within a given epoch, the indices of where these fall within the arguments of the detrending function should be given as a list. If there are no indices to be given, then use an empty list: [] e.g. if the detrending function is given by:

```
foo(times, a, b, c, t0, P):  
    # do something
```

and `a` should be fitted globally, then the entry in the `method_list` would be `['custom', foo, [1], [], []]`.

- **limb_darkening_model** (*str*, *optional*) – The limb darkening model to use. Allowed models are
 - 'linear'
 - 'quadratic'
 - 'squareroot'
 - 'power2'
 - 'nonlinear'

With the exception of the non-linear model, all models are constrained by the method in Kipping (2013), which can be found at <https://arxiv.org/abs/1308.0009>. Default is 'quadratic'.

- **ld_fit_method** (*str*, *optional*) – Determines the mode of fitting of limb darkening parameters. The available modes are:
 - 'coupled' : all limb darkening parameters are fitted independently, but are coupled to a wavelength dependent model based on the host parameters through *ldkt*
 - 'single' : LD parameters are still tied to a model, but only the first filter is actively fitted. The remaining filters are estimated based off the ratios given by *ldtk* for a host with the given parameters. This mode is useful for a large number of filters, as 'coupled' or 'independent' fitting will lead to much higher computation times.
 - 'independent' : Each LD coefficient is fitted separately for each filter, with no coupling to the *ldtk* models.
 - 'off' : no limb darkening fitting will occur. If using this mode, it is strongly recommended to set values for the Kipping *q* parameters using the priors file.

Default is 'independent'

- **fitting_mode** ({'auto', 'all', '2_stage', 'folded', 'batched'}, *optional*) – The approach TransitFit takes towards limiting the number of parameters being simultaneously fitted. The available modes are: - 'auto' : Will calculate the number of parameters required to fit

all the data simultaneously. If this is less than `max_parameters`, will set to 'all' mode, else will set to 'folded' if at least one filter has at least 3 epochs in it. Otherwise will set to 'batched'

- 'all' : Fits all parameters simultaneously, with no folding or batching of curves. Should be used with caution when fitting very large ($\sim < 30$) numbers of parameters.
- '2_stage' : Fits in 2 stages, first detrending the light curves and then fitting the detrended curves simultaneously, using the 'batched' approach if required.
- 'folded' : Useful for fitting curves with multiple epochs for each filter. TransitFit will fit each filter separately and produce a period-folded light curve for each filter, before fitting these simultaneously, using the 'batched' approach if required.
- 'batched' : Useful for large numbers of light curves with relatively few shared filters, so 'folded' loses large amounts of multi-epoch information. This mode splits the filters

into sets of overlapping batches, runs each batch and uses the weighted means of each batch to produce a final result.

Default is 'auto'.

- **max_batch_parameters** (*int*, *optional*) – The maximum number of parameters to use in a single retrieval when using 'folded' or 'batched' fitting modes. Default is 25.
- **batch_overlap** (*in*, *optional*) – The number of filters or epochs to overlap adjacent batches by where possible. This ensures that adjacent batches share information. Default is 2.
- **host_T** (*tuple* or *None*, *optional*) – The effective temperature of the host star, in Kelvin. Should be given as a (value, uncertainty) pair. Required if `ld_fit_method` is 'single' or 'coupled'. Default is None.
- **host_logg** (*tuple* or *None*, *optional*) – The \log_{10} of the surface gravity of the host star, with gravity measured in cm/s^2 . Should be given as a (value, uncertainty) pair. Required if `ld_fit_method` is 'single' or 'coupled'. Default is None
- **host_z** (*tuple* or *None*, *optional*) – The metallicity of the host, given as a (value, uncertainty) pair. Required if `ld_fit_method` is 'single' or 'coupled'. Default is None
- **host_r** (*tuple* or *None*, *optional*) – The host radius in Solar radii, given as a (value, uncertainty) pair. Required for conversion of host-planet separation from AU to host radii
- **nlive** (*int*, *optional*) – The number of live points to use in the nested sampling retrieval.
- **normalise** (*bool*, *optional*) – If True, will assume that the light curves have not been normalised and will fit normalisation constants within the retrieval. The range to fit normalisation constants `c_n` are automatically detected using

$$1/f_{\text{max}} \leq c_n \leq 1/f_{\text{min}}$$

as the default range, where `f_min` and `f_max` are the minimum and maximum flux values for a given light curve. Default is True.

- **detrend** (*bool*, *optional*) – If False, no detrending will be attempted, even if specified by detrending list. Default is True
- **dlogz** (*float*, *optional*) – Retrieval iteration will stop when the estimated contribution of the remaining prior volume to the total evidence falls below this threshold. Explicitly, the stopping criterion is $\ln(z + z_{\text{est}}) - \ln(z) < \text{dlogz}$, where z is the current evidence from all saved samples and z_{est} is the estimated contribution from the remaining volume. The default is $1e-3 * (\text{nlive} - 1) + 0.01$.
- **maxiter** (*int* or *None*, *optional*) – The maximum number of iterations to run. If None, will continue until stopping criterion is reached. Default is None.
- **maxcall** (*int* or *None*, *optional*) – The maximum number of likelihood calls in retrieval. If None, will continue until stopping criterion is reached. Default is None.
- **dynesty_sample** (*str*, *optional*) – Method used to sample uniformly within the likelihood constraint, conditioned on the provided bounds. Unique methods available are: uniform sampling within the bounds ('unif'), random walks with fixed proposals ('rwalk'), random walks with variable (“staggering”) proposals ('rstagger'), multivariate slice sampling along preferred orientations ('slice'), “random” slice sampling along all orientations ('rslice'), “Hamiltonian” slices along random trajectories ('hslice'), and any callable function which follows the pattern of the sample methods defined in `dynesty.sampling`. ‘auto’ selects the sampling method based on the dimensionality of the problem (from `ndim`). When `ndim < 10`, this defaults to 'unif'. When $10 \leq \text{ndim} \leq 20$, this defaults to 'rwalk'.

When `ndim > 20`, this defaults to `'hslice'` if a gradient is provided and `'slice'` otherwise. `'rstagger'` and `'rslice'` are provided as alternatives for `'rwalk'` and `'slice'`, respectively. Default is `'rslice'`.

- **`dynesty_bounding`** (`{'none', 'single', 'multi', 'balls', 'cubes'}`, optional) – The decomposition to use in sampling. Default is `'multi'`
- **`results_output_folder`** (*str*, optional) – Folder to save results to. TransitFit will create subfolders within this if folded or batched runs are being used. Default is `'./output_parameters'`
- **`fitted_lightcurve_folder`** (*str*, optional) – The folder to save fitted light curves to. These files contain the normalised and detrended light curves, as well as the best fit curve. Default is `'./fitted_lightcurves'`
- **`summary_file`** (*str*, optional) – The file to save the summarised final parameter results to. These are calculated by taking weighted averages over any batched fitting. Default is `'summary_output.csv'`
- **`full_output_file`** (*str*, optional) – The file to save the full, non-summarised results to. This file gives the results for each batch, without averaging over batches to get summarised results. Default is `'full_output.csv'`
- **`plot_folder`** (*str*, optional) – Path to folder to save plots to. Default is `'./plots'`
- **`plot`** (*bool*, optional) – If True, will plot all fitted light curves within the fitting routine, including any from partial fitting (eg, single filter modes). Default is True.
- **`marker_color`** (*matplotlib color*, optional) – The colour to plot data points on plots. Default is `'dimgray'`.
- **`line_colour`** (*matplotlib color*, optional) – The colour to plot best fit light curves on plots. Default is `'black'`.
- **`ldtk_cache`** (*str*, optional) – This is the path to cache LDTK files to. If not specified, will default to the LDTK default.
- **`ldtk_samples`** (*int*, optional) – Controls the number of samples taken by PyLDTk when calculating LDCs when using `'coupled'` or `'single'` modes for limb darkening fitting. Default is 200000
- **`do_ld_mc`** (*bool*, optional) – If True, will use MCMC sampling to more accurately estimate the uncertainty on initial limb darkening parameters provided by PyLDTk. Default is False.
- **`data_skiprows`** (*int*, optional) – The number of rows to skip when reading in light curve data from a .txt file. Default is 0.
- **`allow_ttv`** (*bool*, optional) – If True, will fit `t0` for each epoch individually. Default is False.
- **`filter_delimiter`** (*str*, optional) – The delimiter in filter profile files. Default is None, which will lead to pandas trying to auto detect the delimiter.
- **`detrending_limits`** (*list*, optional) – The bounds on detrending coefficients, given as (lower, upper) pair for each detrending method. If not provided, will default to ± 10
- **`bin_data`** (*bool*, optional) – If True, any folded light curves will be plotted with data binned to an observing cadence given by *cadence*. Default is False.
- **`cadence`** (*float*, optional) – The observing cadence, in minutes, to bin data to if *bin_data* is True. Default is 2 (mirroring TESS observations)

- **binned_color** (*str*, *optional*) – The color to use for binned data. Default is 'red'.
- **n_procs** (*int*, *optional*) – The maximum number of processes to use when running batches. If >1, will run batches in parallel. Default is 1.
- **median_normalisation** (*bool*, *optional*) – Normalising lightcurves by the median of the flux value reduces the runtime in many cases. Default is False.
- **error_scaling** (*bool*, *optional*) – If True, scales the errorbars in the lightcurves following <https://emcee.readthedocs.io/en/stable/tutorials/line/>
- **error_scaling_limits** (*list*, *optional*) – If error_scaling=True, this is the limit of the parameter.
- **ldtk_uncertainty_multiplier** (*float*, *optional*) – (From LDTK:) The uncertainty multiplier is a subjective factor that defines how strongly the LD profile (or the prior created from it) constrains the final analysis (that is, how much we trust the stellar atmosphere models used to create the profiles.)

Returns

results – The results returned by `Retriever.run_dynesty()`

Return type

`dict`

3.8.2 Retriever

Object which actually handles the retrieval process

```
class transitfit.retriever.Retriever(data_files, priors, n_telescopes, n_filters, n_epochs,
                                     filter_info=None, detrending_list=[['nth order', 1]],
                                     limb_darkening_model='quadratic', host_T=None,
                                     host_logg=None, host_z=None, host_r=None, ldtk_cache=None,
                                     n_ld_samples=20000, do_ld_mc=False, data_skiprows=0,
                                     allow_ttv=False, filter_delimiter=None, detrending_limits=None,
                                     normalise=True, normalise_limits=None, detrend=True,
                                     median_normalisation=False, error_scaling=False,
                                     error_scaling_limits=None, ldtk_uncertainty_multiplier=1.0)
```

Bases: `object`

Handles the nested sampling retrieval and acts as a high-level interface

Parameters

- **data_files** (*str*) – Path to the data input config file
- **priors** (*str*) – Path to the priors config file
- **n_telescopes** (*int*) – The number of different telescopes used in the data set
- **n_filters** (*int*) – The number of different filters used in the data set
- **n_epochs** (*int*) – The number of different epochs used in the data set
- **filter_info** (*str*, *optional*) – The path to the filter profiles config file
- **detrending_list** (*list*, *optional*) – The different detrending models to use
- **limb_darkening_model** (*str*) – The limb darkening model to use

- **host_T** (*tuple* or *None*, *optional*) – The effective temperature of the host star, in Kelvin. Should be given as a (value, uncertainty) pair. Required if `ld_fit_method` is 'single' or 'coupled'. Default is `None`.
- **host_logg** (*tuple* or *None*, *optional*) – The `log10` of the surface gravity of the host star, with gravity measured in `cm/s2`. Should be given as a (value, uncertainty) pair. Required if `ld_fit_method` is 'single' or 'coupled'. Default is `None`.
- **host_z** (*tuple* or *None*, *optional*) – The metallicity of the host, given as a (value, uncertainty) pair. Required if `ld_fit_method` is 'single' or 'coupled'. Default is `None`.
- **host_r** (*tuple* or *None*, *optional*) – The host radius in Solar radii, given as a (value, uncertainty) pair. Required for conversion of host-planet separation from AU to host radii.
- **ldtk_cache** (*str*, *optional*) – This is the path to cache LDTK files to. If not specified, will default to the LDTK default.
- **n_ld_samples** (*int*, *optional*) – Controls the number of samples taken by PyLDTk when calculating LDCs when using 'coupled' or 'single' modes for limb darkening fitting. Default is 20000.
- **do_ld_mc** (*bool*, *optional*) – If True, will use MCMC sampling to more accurately estimate the uncertainty on initial limb darkening parameters provided by PyLDTk. Default is False.
- **data_skiprows** (*int*, *optional*) – The number of rows to skip when reading in light curve data from a .txt file. Default is 0.
- **allow_ttv** (*bool*, *optional*) – If True, will fit `t0` for each epoch individually. Default is False.
- **filter_delimiter** (*str*, *optional*) – The delimiter in filter profile files. Default is `None`, which will lead to pandas trying to auto detect the delimiter.
- **detrending_limits** (*list*, *optional*) – The bounds on detrending coefficients, given as (lower, upper) pair for each detrending method. If not provided, will default to ± 10 .
- **error_scaling** (*bool*, *optional*) – If True, scales the errorbars in the lightcurves following <https://emcee.readthedocs.io/en/stable/tutorials/line/>.
- **error_scaling_limits** (*list*, *optional*) – If `error_scaling=True`, this is the limit of the parameter.
- **ldtk_uncertainty_multiplier** (*float*, *optional*) – (From LDTK:) The uncertainty multiplier is a subjective factor that defines how strongly the LD profile (or the prior created from it) constrains the final analysis (that is, how much we trust the stellar atmosphere models used to create the profiles.)

_calculate_n_params(*lightcurves*, *indices*, *ld_fit_method*, *normalise*, *detrend*)

Calculates the number of parameters which would be fitted for a given set of filter and epoch indices

This function exists because it's much faster than repeatedly making `PriorInfos` for different combos.

Parameters

indices (*tuple*) – The tuple of indices to consider. Must be given as (`telescope_indices`, `filter_indices`, `epoch_indices`)

_fold_lightcurves(*results*, *priors*, *lightcurves*)

Produces a set of folded lightcurves which can then be fitted across filters

Parameters

- **results** (*array_like*, *shape* (*n_filters*,)) – Each entry should be a list of results objects for each batch
- **priors** (*array_like*, *shape* (*n_filters*,)) – Each entry should be a list of Prior-Info objects for each batch
- **lightcurves** (*array_like*, *shape* (*n_filters*,)) – The lightcurves. Each entry should be a list of LightCurve arrays used for the batches within each filter

Returns

- **folded_lightcurves** (*np.array*, *shape* (*1*, *n_filters*, *1*)) – All the lightcurves, with each filter folded onto one epoch.
- **folded_P** (*float*) – The period that the lightcurves are folded with
- **folded_t0** (*float*) – The t0 that all lightcurves are centred on.

_format_indices (*indices*)

If passed a set of indices, checks they are usable. If indices is None, sets them to cover all of the possible values

_full_to_subset_index (*subset_indices*, *full_index*)

Converts an index which uses the notation of full parameter space to a subset. Useful for converting between overall indexing and indexing within a batch

Parameters

- **subset_indices** (*tuple*) – The indices which define the full subset of light curves
- **full_index** (*array_like*, *shape* (*3*,)) – The full-notation index to be converted

_get_detrending_subset (*indices*)

Pulls out the detrending indices for the given telescope, filter and epoch indices (given as tuple), assuming that we are working on all input lightcurves, not a folded version (though that shouldn't be using this function)

_get_folding_batches (*max_parameters*, *ld_fit_method*, *detrend*, *normalise*, *overlap=2*, *random_order=True*)

Splits all_lightcurves into single-filter, multi-epoch batches to be fitted, which will allow us to produce folded lightcurves. This includes the option to have batches overlapping so that they share some info.

Parameters

- **max_parameters** (*int*) – The maximum number of parameters to have in a single batch
- **ld_fit_method** (*str*) – The limb darkening fit method
- **detrend** (*bool*) – If True, detrending will be used
- **normalise** (*bool*) – If true, normalisation will be used
- **overlap** (*int*, *optional*) – The number of epochs to overlap in each batch. This will be adhered to where possible. Default is 2.
- **random_order** (*bool*, *optional*) – If True, will shuffle the epochs before batching to reduce correlations from grouping them. Default is True

Returns

batches – Each entry in the array is a list of batches for the given filter.

Return type

array_like, *shape* (*n_filters*,)

`_get_lightcurve_subset`(*lightcurves*, *indices*)

Pulls out the subset of lightcurves given by the indices.

`_get_non_folding_batches`(*lightcurves*, *max_parameters*, *ld_fit_method*, *detrend*, *normalise*, *overlap*=2, *error_scaling*=False)

Splits lightcurves into batches by filter, attempting to ensure that batches do not require more than *max_parameters* to fit. Each batch will contain every light curve in the filter, which trumps *max_parameters*. Where possible, the filter batches will overlap, allowing each batch to share information on some level.

Parameters

- **`lightcurves`** (*array_like*, *shape* (*n_telescopes*, *n_filters*, *n_epochs*)) – The LightCurves
- **`max_parameters`** (*int*) – The maximum number of parameters to have in a single batch
- **`ld_fit_method`** (*str*) – The limb darkening fit method
- **`detrend`** (*bool*) – If True, detrending will be used
- **`normalise`** (*bool*) – If true, normalisation will be used
- **`overlap`** (*int*, *optional*) – The number of epochs to overlap in each batch. This will be adhered to where possible. Default is 2.
- **`lightcurves`** – The lightcurves to be batched.

Returns

`batches` – The final batches. Each entry is a tuple of (telescope indices, filter indices, epoch indices) using the indices of lightcurves. Each of these batches can then be passed to `_get_priors_and_curves`.

Return type

array_like, *shape* (*n_batches*)

`_get_priors_and_curves`(*lightcurves*, *ld_fit_method*, *indices*=None, *detrend*=True, *normalise*=True, *folded*=False, *folded_P*=None, *folded_t0*=None, *suppress_warnings*=False)

Generates a prior info for a particular run:

Parameters

- **`lightcurves`** (*array_like*) – An array of the light curves which will be globally considered for fitting.
- **`ld_fit_method`** ({*'independent'*, *'single'*, *'coupled'*, *'off'*}) – The mode to fit limb darkening coefficients with.
- **`indices`** (*tuple* or *None*) – If None, will fit all light curves. Otherwise, supply relevant indices of lightcurves to fit as a tuple: (*telescope_indices*, *filter_indices*, *epoch_indices*)
- **`detrend`** (*bool*, *optional*) – If True, will initialise detrending fitting. Default is True.
- **`normalise`** (*bool*, *optional*) – If True, will initialise normalisation fitting. Default is True.
- **`folded`** (*bool*, *optional*) – Set to True if using folded light curves (functionally only one epoch). Also turns off detrending and normalisation fitting. Default is False.
- **`folded_P`** (*float*, *optional*) – Required if folded is True. This is the period that the light curves are folded to
- **`folded_t0`** (*float*, *optional*) – Required if folded is True. This is the t0 that the light curves are folded to. In the case where *ttv* mode is used (each epoch has been fitted to a

different t0), then this should be the t0 of the first epoch (the one which everything else is folded back to)

Returns

- **priors** (*PriorInfo*) – The fully initialised *PriorInfo* object
- **lightcurves** – The *LightCurves* in the correct format, with detrending and normalisation initialised

_get_unique_indices(*indices*)

When given a tuple of indices of all light curves to consider, gets all the unique values

_run_batched_retrieval(*lightcurves, batches, ld_fit_method, detrend, normalise, maxiter, maxcall, sample, nlive, dlogz, full_return=False, folded=False, folded_P=None, folded_t0=None, output_folder='./output_parameters', summary_file='summary_output.csv', full_output_file='full_output.csv', lightcurve_folder='./fitted_lightcurves', plot=True, plot_folder='./plots', marker_color='dimgrey', line_color='black', bound='multi', filter_idx=None, walks=100, slices=10, n_procs=1*)

Runs a retrieval using the given batches

lightcurves

[array_like, shape (n_telescopes, n_filters, n_epochs)] The full lightcurves array to be retrieved.

full_return

[bool, optional] If True will return all_results, all_priors, all_lightcurves. If False, will just return all_results. Default is False

_run_dynesty(*lightcurves, priors, maxiter=None, maxcall=None, sample='auto', nlive=300, dlogz=None, bound='multi', plot_folder='./plots', walks=100, slices=10*)

Runs dynesty on the given lightcurves with the given priors. Returns the result.

Returns

- **results**
- **ndof**

_run_folded_retrieval(*ld_fit_method, detrend, normalise, maxiter, maxcall, sample, nlive, dlogz, output_folder='./output_parameters', summary_file='summary_output.csv', full_output_file='full_output.csv', lightcurve_folder='./fitted_lightcurves', plot=True, plot_folder='./plots', marker_color='dimgrey', line_color='black', max_parameters=25, overlap=2, bound='multi', walks=100, slices=10, n_procs=1*)

For each filter, runs retrieval, then produces a phase-folded lightcurve. Then runs retrieval across wavelengths on the folded curves.

_run_full_retrieval(*ld_fit_method, detrend, normalise, maxiter, maxcall, sample, nlive, dlogz, output_folder='./output_parameters', summary_file='summary_output.csv', full_output_file='full_output.csv', lightcurve_folder='./fitted_lightcurves', plot=True, plot_folder='./plots', marker_color='dimgrey', line_color='black', bound='multi', walks=100, slices=10*)

Runs full retrieval with no folding/batching etc. Just a straight forward dynesty run.

_subset_to_full_index(*subset_indices, subset_index*)

Converts an index from notation within a batch to the full indexing. Inverse of `_full_to_subset_index`

Parameters

- **subset_indices** (*tuple*) – The indices which define the full subset of light curves
- **subset_index** (*array_like, shape (3,)*) – The subset-notation index to be converted

detrending_limits

detrending_limits = np.array(detrending_limits) if not detrending_limits.ndim == 2:

raise ValueError(f'Detrending limits should be provided as a list of length {len(self.detrending_info)} where each entry is the [lower, upper] limits on each method.')

if not detrending_limits.shape[1] == 2:

raise ValueError(f'Detrending limits should be provided as a list of length {len(self.detrending_info)} where each entry is the [lower, upper] limits on each method.')

run_retrieval(*ld_fit_method='independent', fitting_mode='auto', max_parameters=25, maxiter=None, maxcall=None, sample='auto', nlive=300, dlogz=None, plot=True, output_folder='./output_parameters', lightcurve_folder='./fitted_lightcurves', summary_file='summary_output.csv', full_output_file='full_output.csv', plot_folder='./plots', marker_color='dimgray', line_color='black', bound='multi', normalise=True, detrend=True, overlap=2, bin_data=True, cadence=2, binned_color='red', walks=100, slices=10, n_procs=1*)

Runs dynesty on the data. Different modes exist and can be specified using the kwargs.

Parameters

- **ld_fit_method** ({*'coupled', 'single', 'independent', 'off'*}, *optional*) – Determines the mode of fitting of limb darkening parameters. The available modes are:
 - *'coupled'* : all limb darkening parameters are fitted independently, but are coupled to a wavelength dependent model based on the host parameters through *ldtk*
 - *'single'* : LD parameters are still tied to a model, but only the first filter is actively fitted. The remaining filters are estimated based off the ratios given by *ldtk* for a host with the given parameters. This mode is useful for a large number of filters, as *'coupled'* or *'independent'* fitting will lead to much higher computation times.
 - *'independent'* : Each LD coefficient is fitted separately for each filter, with no coupling to the *ldtk* models.
 - *'off'* : Will use the fixed value provided in the input file

Default is *'independent'*

- **fitting_mode** ({*'auto', 'all', 'folded', 'batched'*}, *optional*) – Determines if the fitting algorithm is limited by *max_parameters*. If the number of parameters to be fitted exceeds *max_parameters*, then the retrieval will split into fitting each filter independently, phase-folding the detrended light curves to produce a single light curve for each filter and then fitting these phase-folded curves simultaneously. If *fitting_mode* is *'auto'*, then the mode used will be determined automatically. If *fitting_mode* is *'all'*, then all light curves will be attempted to be fitted simultaneously, regardless of the value of *max_parameters*. If *fitting_mode* is *'folded'*, then the folding approach will be used. Default is *'auto'*.
- **max_parameters** (*int, optional*) – The maximum number of parameters to use in a single retrieval. Default is 25.
- **maxiter** (*int or None, optional*) – The maximum number of iterations to run. If *None*, will continue until stopping criterion is reached. Default is *None*.

- **maxcall** (*int* or *None*, optional) – The maximum number of likelihood calls in retrieval. If *None*, will continue until stopping criterion is reached. Default is *None*.
- **sample** (*str*, optional) – Method used to sample uniformly within the likelihood constraint, conditioned on the provided bounds. Unique methods available are: uniform sampling within the bounds ('unif'), random walks with fixed proposals ('rwalk'), random walks with variable ("staggering") proposals ('rstagger'), multivariate slice sampling along preferred orientations ('slice'), "random" slice sampling along all orientations ('rslice'), "Hamiltonian" slices along random trajectories ('hslice'), and any callable function which follows the pattern of the sample methods defined in `dynesty.sampling`. 'auto' selects the sampling method based on the dimensionality of the problem (from `ndim`). When `ndim < 10`, this defaults to 'unif'. When `10 <= ndim <= 20`, this defaults to 'rwalk'. When `ndim > 20`, this defaults to 'hslice' if a gradient is provided and 'slice' otherwise. 'rstagger' and 'rslice' are provided as alternatives for 'rwalk' and 'slice', respectively. Default is 'auto'.
- **nlive** (*int*, optional) – The number of live points to use in the nested sampling retrieval. Default is 300.
- **dlogz** (*float*, optional) – Retrieval iteration will stop when the estimated contribution of the remaining prior volume to the total evidence falls below this threshold. Explicitly, the stopping criterion is $\ln(z + z_{est}) - \ln(z) < dlogz$, where z is the current evidence from all saved samples and z_{est} is the estimated contribution from the remaining volume. The default is $1e-3 * (nlive - 1) + 0.01$.

`transitfit.retriever._run_batch(x)`

Subprocess to run a batch

3.8.3 Prior handling

Object to handle and deal with prior info for retrieval

class `transitfit.priorinfo.PriorInfo`(*default_dict*, *limb_dark*, *n_telescopes*, *n_filters*, *n_epochs*, *allow_ttv=False*, *lightcurves=None*)

Bases: `object`

Object to deal with anything involving priors

Parameters

- **default_dict** (*dictionary*) – Dictionary containing default parameter values.
- **limb_dark** (*str*) – The limb darkening model to use.
- **n_telescopes** (*int*) – The number of different telescopes used in the data set
- **n_filters** (*int*) – The number of different filters used in the data set
- **n_epochs** (*int*) – The number of different epochs used in the data set
- **allow_ttv** (*bool*, optional) – If True, will fit t_0 to each transit. Default is False.
- **lightcurves** (*array_like*, *shape*(*n_telescopes*, *n_filters*, *n_epochs*)) – Array of the light curves. If there is no observation for a particular telescope, filter, epoch combination, then the value should be set to *None*.

_convert_unit_cube(*cube*)

Takes the unit cube provided by `dynesty` (all values between 0 and 1) and converts them into physical values

_interpret_final_results(*results*)

Generates a dictionary of results and a dictionary of errors from the final results of a run

_interpret_param_array(array)

Interprets the parameter cube generated by dynesty and returns them in a format usable by the Likelihood-Calculator

add_gaussian_fit_param(name, mean, stdev, telescope_idx=None, filter_idx=None, epoch_idx=None)

Adds a new parameter which will be fitted with a Gaussian prior

add_uniform_fit_param(name, low_lim, high_lim, telescope_idx=None, filter_idx=None, epoch_idx=None)

Adds a new parameter which will be fitted uniformly in the range given by low_lim and high_lim

fit_detrrending(lightcurves, method_list, method_index_array, limits=None)

Initialises detrrending

fit_limb_darkening(fit_method='independent', host_T=None, host_logg=None, host_z=None, filters=None, n_samples=20000, do_mc=False, cache_path=None, ldtk_uncertainty_multiplier=1.0)

Initialises fitting of limb darkening parameters, either independently or coupled across wavebands.

Parameters

- **fit_method** (str, optional) – Determines the mode of fitting of LD parameters. The available modes are:
 - 'coupled' : all limb darkening parameters are fitted independently, but are coupled to a wavelength dependent model based on the host parameters
 - 'single' : LD parameters are still tied to a model, but only the first filter is actively fitted. The remaining filters are estimated based off the ratios given by ldtk for a host with the given parameters. This mode is useful for a large number of filters, as 'coupled' or 'independent' fitting will lead to much higher computation times.
 - 'independent' : Each LD coefficient is fitted separately for each filter, with no coupling to the ldtk models.The default is 'independent'.
- **host_T** (tuple or None, optional) – The effective temperature of the host star, in Kelvin, given as a (value, uncertainty) pair. Must be provided if fit_method is 'coupled' or 'single'. Default is None.
- **host_logg** (tuple or None, optional or None, optional) – The log₁₀ of the surface gravity of the host star, with gravity measured in cm/s². Should be given as a (value, uncertainty) pair. Must be provided if fit_method is 'coupled' or 'single'. Default is None.
- **host_z** (tuple or None, optional) – The metallicity of the host, given as a (value, uncertainty) pair. Must be provided if fit_method is 'coupled' or 'single'. Default is None.
- **filters** (array_like or None, optional) – The set of filters, given in [low, high] limits for the wavelengths with the wavelengths given in nanometers. The ordering of the filters should correspond to the filter_idx parameter used elsewhere. Must be provided if fit_method is 'coupled' or 'single'. Default is None.
- **n_samples** (int, optional) – The number of limb darkening profiles to create. Passed to ldtk.LDPSetCreator.create_profiles(). Default is 20000.
- **do_mc** (bool, optional) – If True, will use MCMC to estimate coefficient uncertainties more accurately. Default is False.

- **cache_path** (*str*, *optional*) – This is the path to cache LDTK files to. If not specified, will default to the LDTK default
- **ldtk_uncertainty_multiplier** (*float*, *optional*) – (From LDTK:) The uncertainty multiplier is a subjective factor that defines how strongly the LD profile (or the prior created from it) constrains the final analysis (that is, how much we trust the stellar atmosphere models used to create the profiles.)

fit_normalisation(*lightcurves*, *normalise_limits*)

When run, the Retriever will fit normalisation of the data as a free parameter.

Parameters

lightcurves (np.array of `LightCurve`'s, shape (n_telescopes, n_filters, n_epochs)) – The array of `LightCurves` to be normalised. We can use the value of fluxes to estimate a normalisation constant (c_n) range for each light curve. We use

$$1/f_{\text{median}} - 1 \leq c_n \leq 1/f_{\text{median}} + 1$$

as the default range, where `f_median` is the median flux value.

get_latex_friendly_labels()

Adds a \$ around the underscored text to make the latex displays work

set_error_scaling(*lightcurves*, *scaling_limits*, *method_index_array*)

transitfit.priorinfo.setup_priors(*P*, *t0*, *a*, *rp*, *inc*, *ecc*, *w*, *limb_dark*, *n_telescopes*, *n_filters*, *n_epochs*, *q0=None*, *q1=None*, *q2=None*, *q3=None*, *allow_ttv=False*, *lightcurves=None*, *error_scaling=False*)

Factory function to initialise a `PriorInfo` object

`qX` should either be a single value or a list of values length `n_filters`

3.8.4 Limb darkening handling

Class to handle limb darkening

class transitfit._limb_darkening_handler.**LimbDarkeningHandler**(*default_model*, *low_lim=-5*, *high_lim=5*)

Bases: `object`

The `LimbDarkeningHandler` is designed to convert fitting parameters in the range [0,1] and convert them into physically allowed values for limb darkening coefficients for different models. This conversion is based on Kipping 2013 <https://arxiv.org/abs/1308.0009>.

Parameters

- **default_model** (*str*) –

The default limb darkening model to use. Accepted values are

- 'linear'
- 'quadratic'
- 'squareroot'
- 'power2'
- 'nonlinear'

This model will be the default for conversions unless otherwise specified.

- **low_lim** (*float*, *optional*) – The lower limit to use in conversion in the case where there are open bounds on a coefficient (power2 and nonlinear models). Note that in order to conserve sampling density in all regions for the power2 model, you should set lower_lim=-high_lim. Default is -5
- **high_lim** (*float*, *optional*) – The upper limit to use in conversion in the case where there are open bounds on a coefficient (power2 and nonlinear models). Note that in order to conserve sampling density in all regions for the power2 model, you should set lower_lim=-high_lim. Default is 5

convert_qtou(*q, model=None)

Takes parameters q distributed between [0,1] and converts them into physical values for limb darkening coefficients.

Conversions for quadratic, square root, and logarithmic are from Kipping 2013 <https://arxiv.org/abs/1308.0009> for two-parameter limb darkening methods

Notes

This is the inverse of convert_utoq

convert_qtou_with_errors(q, q_err, model=None)

Converts kipping parameters with errors into physical LDCs with errors

convert_utoq(*u, model=None)

Takes actual values of the LD coefficients and converts them to a value q between [0,1].

Conversions for quadratic, square root, and logarithmic are from Kipping 2013 <https://arxiv.org/abs/1308.0009> for two-parameter limb darkening methods

Notes

This is the inverse of convert_qtou

get_required_coefficients(model=None)

Finds the number of coefficients required for a given model and returns a string name for each of the form 'uX' where X is a number.

Parameters

model (*str* or *None*, *optional*) – Model to get the coefficients for. Default is self.default_model

Returns

coefficient names – The coefficient names

Return type

list of *str*

initialise_ldtk(host_T, host_logg, host_z, filters, model=None, n_samples=20000, do_mc=False, cache_path=None, ldtk_uncertainty_multiplier=1.0)

Sets up an LDTKHandler to deal with interfacing between ldtk and TransitFit

Parameters

- **host_T** (*tuple*) – The effective temperature of the host star, in Kelvin, given as a (value, uncertainty) pair.

- **host_logg** (*tuple*) – The \log_{10} of the surface gravity of the host star, with gravity measured in cm/s^2 . Should be given as a (value, uncertainty) pair.
- **host_z** (*tuple*) – The metallicity of the host, given as a (value, uncertainty) pair.
- **filters** (*array_like*) – The set of filters, given in [low, high] limits for the wavelengths with the wavelengths given in nanometers. The ordering of the filters should correspond to the `filter_idx` parameter used elsewhere.
- **ld_method** (*str or None, optional*) – The model of limb darkening to use. Allowed values are 'linear', 'quadratic', 'squaresroot', 'power2', and 'nonlinear'. If None, will default to `self.default_model`. Default is None.
- **n_samples** (*int, optional*) – The number of limb darkening profiles to create. Passed to `ldtk.LDPSetCreator.create_profiles()`. Default is 20000.
- **do_mc** (*bool, optional*) – If True, will use MCMC to estimate coefficient uncertainties more accurately. Default is False.
- **cache_path** (*str, optional*) – This is the path to cache LDTK files to. If not specified, will default to the LDTK default
- **ldtk_uncertainty_multiplier** (*float, optional*) – (From LDTK:) The uncertainty multiplier is a subjective factor that defines how strongly the LD profile (or the prior created from it) constrains the final analysis (that is, how much we trust the stellar atmosphere models used to create the profiles.)

ldtk_estimate(*ld0_values, model=None*)

Uses LDTK to estimate the LDCs for all filters when given a set of LDCs for filter 0, based on the ratios between the best values found in initialisation.

Parameters

- **ld1_values** (*float or array_like*) – The LD parameters for filter 0
- **ld_model** (*str, optional*) – The limb darkening model to use. Defaults to `self.default_model`

Returns

all_ld_values – The estimated limb darkening parameters

Return type

array_like, shape (n_filters, n_coeffs)

ldtk_inlike(*coeffs, model=None*)

Uses LDTK to evaluate the log likelihood for a set of coefficients. Note that `coeffs` should be the values returned by `convert_qtoA` and NOT the unit values.

Parameters

- **coeffs** (*array_like, shape (n_filters, n_coeffs)*) – The coefficients to evaluate the log likelihood for.
- **ld_model** (*str, optional*) – The model to use. Defaults to `self.default_model`

Returns

Inlike – The log likelihood of the coefficients

Return type

float

3.8.5 LDTK interfacing

Class to handle limb darkening parameters through PyLDTK

```
class transitfit._ldtk_handler.LDTKHandler(host_T, host_logg, host_z, filters, ld_model='quadratic',  
                                         n_samples=20000, do_mc=False, cache_path=None,  
                                         ldtk_uncertainty_multiplier=1.0)
```

Bases: `object`

The LDTKHandler provides an easy way to interface Ldtk with TransitFit.

Parameters

- **host_T** (*tuple*) – The effective temperature of the host star, in Kelvin, given as a (value, uncertainty) pair.
- **host_logg** (*tuple*) – The log₁₀ of the surface gravity of the host star, with gravity measured in cm/s². Should be given as a (value, uncertainty) pair.
- **host_z** (*tuple*) – The metallicity of the host, given as a (value, uncertainty) pair.
- **filters** (*array_like*) – The set of filters, given in [low, high] limits for the wavelengths with the wavelengths given in nanometers if a uniform filter is to be used, or [[wavelength. . .], [transmission]] if a fully-defined profile is being used. The ordering of the filters should correspond to the filter_idx parameter used elsewhere.
- **ld_method** (*str, optional*) – The model of limb darkening to use. Allowed values are 'linear', 'quadratic', 'squareroot', 'power2', and 'nonlinear'. Default is 'quadratic'.
- **n_samples** (*int, optional*) – The number of limb darkening profiles to create. Passed to `ldtk.LDPSetCreator.create_profiles()`. Default is 20000.
- **do_mc** (*bool, optional*) – If True, will use MCMC to estimate coefficient uncertainties more accurately. Default is False.
- **cache_path** (*str, optional*) – This is the path to cache LDTK files to. If not specified, will default to the LDTK default
- **ldtk_uncertainty_multiplier** (*float, optional*) – (From LDTK:) The uncertainty multiplier is a subjective factor that defines how strongly the LD profile (or the prior created from it) constrains the final analysis (that is, how much we trust the stellar atmosphere models used to create the profiles.)

```
_extract_best_coeffs(ld_model, do_mc=False)
```

Extracts the best values for a limb darkening model for the filters

Parameters

ld_model (*str*) – The limb darkening model to obtain the values for.

Returns

- **coeffs** (*array_like, shape (n_filters, n_coeffs)*) – The coefficients for each filter
- **err** (*array_like, shape (n_filters, n_coeffs)*) – The uncertainty on each of the coefficients

```
estimate_values(ld0_values, ld_model)
```

If given a set of LD param values for filter 0, will estimate the LD parameters for all filters based on the ratios between the best values found in initialisation.

Parameters

- **ld1_values** (*float or array_like*) – The LD parameters for filter 0
- **ld_model** (*str*) – The limb darkening model to use

Returns

all_ld_values – The estimated limb darkening parameters

Return type

array_like, shape (n_filters, n_coeffs)

lnlike(*coeffs*, *ld_model*)

Evaluates the log likelihood for a set of coefficients

Parameters

- **coeffs** (array_like, shape (n_filters, n_coeffs)) – The coefficients to evaluate the log likelihood for.
- **ld_model** (str, optional) – The model to use. Defaults to self.default_model

3.8.6 Likelihood calculations

Module to calculate the likelihood of a set of parameters

class transitfit._likelihood.LikelihoodCalculator(*lightcurves*, *priors*)

Bases: `object`

Object to quickly calculate the likelihood of a set of parameters to fit a given set of light curves.

Parameters

- **lightcurves** (array_like, shape (n_telescopes, n_filters, n_epochs)) – An array of LightCurves. If no data exists for a point in the array then the entry should be *None*.
- **priors** (PriorInfo) – The PriorInfo object for retrieval

find_likelihood(*params*)

Finds the likelihood of a set of parameters

find_likelihood_parallel_processed(*cube*)

update_params(*telescope_idx*, *filter_idx*, *epoch_index*, *t0*=None, *P*=None, *rp*=None, *a*=None, *inc*=None, *ecc*=None, *w*=None, *limb_dark*=None, *u*=None)

Updates self.params with values given

3.8.7 Detrending

detrender.py

A module to deal with arbitrary detrending

class transitfit.detrender.DetrendingFunction(*function*, *method*=None)

Bases: `object`

The DetrendingFunction is designed to handle detrending with an arbitrary function.

Basically gets around dealing with varying number of args in a neat way

Parameters

function (*function*) – The detrending function. Must have signature of f(times, a, b, c,...,t0,P) where a, b, c etc are the detrending coefficients, t0 is the time of conjunction and P is period.

Some detrending functions for use with LightCurve

class transitfit.detrending_funcs.NthOrderDetrendingFunction(*order*)

Bases: `object`

Arbitrary order detrending function which conserves flux.

Parameters

order (*int*) – The detrending function order. Must be greater than 0.

3.8.8 Input and output

Object to deal with writing fitting results to files

class transitfit.output_handler.OutputHandler(*lightcurves*, *full_prior*, *host_r=None*)

Bases: `object`

Designed to handle writing outputs from retrieval to files

Parameters

- **lightcurves** (*array_like*, *shape* (*n_telescopes*, *n_filters*, *n_epochs*)) – The full lightcurves array to be retrieved.
- **full_prior** (`PriorInfo`) – The prior for the complete light curve dataset.

_batch_to_full_idx(*i*, *param_name*, *lightcurves*, *allow_ttv*)

Converts a batch index into a full index

Parameter

i

[*tuple*] (*batch_tid*, *batch_fid*, *batch_eid*)

rtype

(*tid*, *fid*, *eid*)

_initialise_batman(*all_lightcurves*)

Sets up batman so we can generate best-fit models for outputs

Parameters

all_lightcurves (*array_like*) – Array containing the global set of light curves

_initialise_best_model(*mode*, *global_prior*, *output_folder*, *summary_file*)

Sets up the best fit model used to output best fit

Parameters

- **mode** (*str*) – The fitting mode being used
- **global_prior** (`PriorInfo`) – The full_prior from the retriever. Used to determine array shapes.

Returns

best_model – A dictionary containing param arrays of the best values

Return type

`dict`

Notes

To do this, we go through all output parameter files and pull in the values. In folded mode, we pull in from the global summary first and add from the filter summaries after

`_initialise_dict_entry(d, param, prior=None)`

Initialises param in results dictionaries using ParamArray

If prior is not provided, defaults to self.full_prior

`_plot_data(phase, flux, flux_err, model_phase, model_curve, residuals, fname, title=None, folder='./plots', figsize=(12, 8), marker_color='dimgrey', line_color='black', phase_lims=None, bin_data=False, cadence=2, binned_colour='red')`

Plots the lightcurve and model consistently

Parameters

`cadence` (*float*) – The cadence to bin to in phase, *not* minutes

`_plot_samples(result, prior, fname, folder='./plots')`

Makes a corner plot of the samples from a result

`_print_results(result)`

Prints a results dict to terminal

`_quicksave_result(results, priors, lightcurves, base_output_path='./outputs', filter=None, batch=None)`

Quickly saves a batch result to file and pickle the Result and Prior objects

`_results_dict_to_dataframe(results_dict, batched)`

Converts a results dict to a pandas dataframe

`_save_results_dict(results_dict, path, batched)`

Saves a dict to csv

`add_best_u(best_dict, combined_dict)`

Given results dicts, adds in the u vals

Parameters

- **`best_dict`** (*dict*) – The dictionary of best results
- **`combined_dict`** (*dict*) – The full results

Returns

- **`best_dict`** (*dict*) – The same dictionary with the best u vals added.
- **`combined_dict`** (*dict*) – The same dictionary with the u vals added.

`combine_results_dicts(results_dicts)`

Combines the given results dicts into one dict

Parameters

`results_dicts` (*array_like*, *shape* (n_batches,)) – The results_dicts obtained from get_results_dict

Returns

`combined_dict` – The combined results dictionary. Each entry is a list of values from the results dictionaries - [[best value, median, 16th percentile, 84th percentile, stdev],...]

Return type

dict

get_best_vals(*results_dicts*, *priors*, *fit_ld=True*, *return_combined=True*)

Gets the best values for a set of runs from the given results dicts

Parameters

- **results_dicts** (*array_like*, *shape* (*n_batches*,)) – The results_dicts obtained from get_results_dict
- **fit_ld** (*bool*, *optional*) – Should be True if LDC fitting. Default is True
- **return_combined** (*bool*, *optional*) – If True, will return the results dicts combined into a single dict. Default is True.

Returns

- **best_vals** (*dict*) – Each entry is [best val, error]
- **combined_dict** (*dict*) – The combined results dictionary. Returned if return_combined is True

get_results_dict(*results*, *priors*, *lightcurves*)

Makes dictionaries of results from a single dynesty run.

Useful for putting results in a form which allows for easy summary etc.

Parameters

- **results** (*dict*) – Results from a single dynesty run
- **priors** (*PriorInfo*) – The priors for the run
- **lightcurves** (*array_like*, *shape* (*n_telescopes*, *n_filters*, *n_epochs*)) – The lightcurves for the run

Returns

results_dict – Each entry is [best value, median, 16th percentile, 84th percentile, stdev]

Return type

dict

plot_final_light_curves(*all_lightcurves*, *global_prior*, *folder='./plots'*, *figsize=(12, 8)*,
marker_color='dimgrey', *line_color='black'*, *plot_folded=True*, *titles=False*,
bin_data=True, *cadence=2*, *binned_color='red'*)

Plots the detrended light curves with the global best-fit model

Parameters

cadence (*float*, *optional*) – The cadence to bin to in minutes

save_complete_results(*mode*, *global_prior*, *output_folder*, *summary_file*)

Once all batches etc are run, collates all results and saves to csv

save_final_light_curves(*all_lightcurves*, *global_prior*, *folder='./final_light_curves'*, *folded=False*)

Saves the final curves and best fit model to file

Requires best model to be initialised

Parameters

all_lightcurves (*array_like*) – Array containing the global set of light curves - these should be raw and not normalised or detrended.

save_results(*results*, *priors*, *lightcurves*, *output_folder='./output_parameters'*,
summary_file='summary_output.csv', *full_output_file='full_output.csv'*,
samples_plot_folder='./plots', *folded=False*, *plot_posteriors=True*, *batch_idx=None*,
stage=1)

Saves results to .csv files

Parameters

- **results** (*array_like*, *shape* (*n_batches*,)) – The results from each run
- **priors** (*array_like*, *shape* (*n_batches*,)) – The priors for each run
- **lightcurves** (*array_like*, *shape* (*n_batches*,)) – The light curves for each run
- **fit_ld** (*bool*, *optional*) – Must be true if LDCs are fitted. Default is True
- **output_folder** (*str*, *optional*) – The folder to save output files to (not plots). Default is `./output_parameters`
- **summary_file** (*str*, *optional*) – The file name for the final output. This file only gives the averaged values, rather than individual values fitted within batches if there are any. Default is `summary_output.csv`
- **full_output_file** (*str*, *optional*) – The file name for the full output file. This file gives partial results from batches, rather than the averaged results. Default is `full_output.csv`

Returns

- **best_vals** (*dict*) – Each entry is [best val, error]
- **combined_dict** (*dict*) – The combined results dictionary. Each entry is a list of values from the results dictionaries - [[best value, median, 16th percentile, 84th percentile, stdev],...]

class transitfit.output_handler.**Results**(*sampler*)

Bases: `object`

Dynesty>v 1.1 doesn't allow setting attributes directly. This new class help in creating a new instance with attributes for easier handling.

class transitfit.output_handler.**ResultsException**(*sampler*)

Bases: `object`

Handles results (same as above) during exception raised by dynesty.

Module to deal with all the reading in of inputs, as well as printing final results to terminal

transitfit.io._read_data_csv(*path*, *usecols=None*)

Given a path to a csv with columns [time, flux, errors], will get all the data in a way which can be used by the Retriever

transitfit.io._read_data_txt(*path*, *skiprows=0*, *usecols=None*, *delimiter=' '*)

Reads a txt data file with columns

transitfit.io.get_filter_path(*input_str*, *unit*)

Checks if a path or name of a default provided filter is given.

Parameters

input (*str*) – The input string from the filter file input

Returns

path – The path to the relevant filter

Return type

`str`

`transitfit.io.parse_data_path_list(data_path_list)`

Parses a list of paths to data files and places them into an array which can be passed to `read_data_file_array`

Parameters

data_path_list (*array_like*, *shape* (n_light_curves, 5)) – The list of paths. Each row should contain [data path, telescope idx, filter idx, epoch idx, detrending idx]

Returns

- **data_path_array** (*array_like*, *shape* (num_filters, num_epochs, num_telescopes)) – The paths inserted into an array which can be used by TransitFit
- **detrending_index_array** (*array_like*, *shape* (num_filters, num_epochs, num_telescopes)) – The array of detrending model indices for each light curve

`transitfit.io.parse_filter_list(filter_list, delimiter=None, unit='nanometers')`

Parses a list of filter information into a usable form for the 'filters' argument in `PriorInfo.fit_limb_darkening`.

Parameters

- **filter_list** (*array_like*, *shape* (n_filters, 3)) – The information on the filters. Each row should contain [filter_index, low wavelength/file path, high wavelength] The filter indices should refer to the indices used in the priors file. Wavelengths should be in nm.
- **delimiter** (*str*, *optional*) – The delimiter used in filter profile files. Default is None, which automatically detects using `csv.Sniffer`

Returns

filter_info – The filter information pass to `PriorInfo.fit_limb_darkening`.

Return type

`np.array`, *shape* (n_filters, 2)

`transitfit.io.parse_priors_list(priors_list, n_telescopes, n_filters, n_epochs, ld_model, filter_indices=None, folded=False, folded_P=None, folded_t0=None, host_radius=None, allow_ttv=False, lightcurves=None, suppress_warnings=False, error_scaling=False)`

Parses a list of priors to produce a `PriorInfo` with all fitting parameters initialised.

Parameters

- **priors_list** (*array_like*, *shape* (X, 5)) – A list of prior information for each variable to be fitted. Can also set fixed values by setting the low and high values to *None* or *np.nan*. Each row should be of the form [key, mode, input A, input B, filter index]
- **n_telescopes** (*int*) – The number of different telescopes being used. Required so that simultaneous observations from different observatories can be used by TransitFit
- **n_filters** (*int*) – The number of different filters being used
- **n_epochs** (*int*) – The number of different epochs being used
- **ld_model** (*str*) – The limb darkening model to use
- **filter_indices** (*array_like*, *optional*) – If provided, will only initialise fitting for parameters which are relevant to the filter indices given. Note that this will result in a difference between the filter indices used at the top, user level and those used within this `PriorInfo`
- **folded** (*bool*, *optional*) – If True, will not initialise P or t0 from the priors list. Instead will use `folded_P` and `folded_t0` to set fixed values. Default is False
- **folded_P** (*float*, *optional*) – Required if `folded` is True. This is the period that the light curves are folded to

- **folded_t0** (*float*, *optional*) – Required if folded is True. This is the t0 that the light curves are folded to
- **host_radius** (*float*, *optional*) – The host radius in Solar radii. If this is provided, then will assume that the orbital separation is given in AU rather than host radii and will convert the values accordingly

Returns

priors – The fully initialised PriorInfo which can then be used in fitting.

Return type

PriorInfo

`transitfit.io.print_results(results, priorinfo, n_dof)`

Prints the results nicely to terminal

Parameters

- **results** (*dynesty.results.Results*) – The Dynesty results object, but must also have weights, cov and uncertainties as entries.
- **priorinfo** (`transitfit.priorinfo.PriorInfo`) – The PriorInfo object

`transitfit.io.read_data_file(path, skiprows=0, folder=None, usecols=None, delimiter=None)`

Reads a file in, assuming that it is either a:

.csv .txt

with columns in the order time, depth, errors. Note that TransitFit assumes BJD times.

Parameters

- **path** (*str*) – Full path to the file to be loaded
- **skiprows** (*int*, *optional*) – Number of rows to skip in reading txt file (to avoid headers)
- **folder** (*str* or *None*, *optional*) – If not None, this folder will be prepended to all the paths. Default is None.
- **usecols** (*int* or *sequence*, *optional*) – Which columns to read, with 0 being the first. For example, `usecols = (1, 4, 5)` will extract the 2nd, 5th and 6th columns. These should be given in the order time, flux, uncertainty. The default, None, results in all columns being read.

Returns

- **times** (*np.array*) – The times of the data series
- **flux** (*np.array*) – The flux
- **error** (*np.array*) – The uncertainty on the flux

`transitfit.io.read_data_path_array(data_path_array, skiprows=0)`

If passed an array of paths, will read in to produce an array of `LightCurve`s`

Parameters

data_path_array (*array_like*, *shape* (*n_telescopes*, *n_filters*, *n_epochs*)) – Array with paths to data to load

Returns

lightcurves – The data loaded and stored as an array of `LightCurves`

Return type

np.array of `LightCurve`s`, *shape* (*n_telescopes*, *n_filters*, *n_epochs*)

`transitfit.io.read_filter_info(path, delimiter=None, unit='nanometers')`

Reads in information on the filters from .csv file and puts them in a format which can be passed to the `filters` argument in `PriorInfo.fit_limb_darkening`.

Parameters

- **path** (*str*) – Path to the .csv file containing the information on the filters. This file should have three columns:

filter_idx | low_wl_or_path | high_wl |

Filters can either be specified as uniform between low_wl and high_wl, or a full profile can be passed through a file. Filter profile files must be two columns, giving wavelength and transmission fraction in range [0,1]. The filter indices should refer to the indices used in the priors file. All wavelengths should be in nm.

- **delimiter** (*str*, *optional*) – The delimiter used in filter profile files. Default is None, which automatically detects using `csv.Sniffer`

Returns

filter_info – The filter information pass to `PriorInfo.fit_limb_darkening`.

Return type

np.array, shape (n_filters, 2)

`transitfit.io.read_input_file(path, skiprows=0)`

Reads in a file with listed inputs and produces data arrays for use in retrieval.

Parameters

- **path** (*str*) – The path to the .csv file with the paths to input parameters and their telescope, filter, epoch, and detrending indices.

path | telescope | filter | epoch | detrending |

Returns

- **lightcurves** (np.array of `LightCurve``s, shape (n_telescopes, n_filters, n_epochs)) – The data loaded and stored as an array of `LightCurves`
- **detrending_index_array** (*array_like*, shape (n_telescopes, n_filters, n_epochs)) – The detrending indices for each lightcurve

`transitfit.io.read_priors_file(path, n_telescopes, n_filters, n_epochs, limb_dark='quadratic',
filter_indices=None, folded=False, folded_P=None, folded_t0=None,
host_radius=None, allow_ttv=None, lightcurves=None,
suppress_warnings=False, error_scaling=False)`

If given a csv file containing priors, will produce a `PriorInfo` object based off the given values

Parameters

- **path** (*str*) – Path to .csv file containing the priors.

key | mode | input_A | input_B | filter |

The available modes and the expected values of inputs A and B are:

- **'uniform':** input A should be lower limit, input B should be upper limit
- **'gaussian':** input_A should be mean, input_B should be standard deviation.
- **'fixed':** input_A should be the fixed value. input_B is not used and should be left blank.
- **n_telescopes** (*int*) – The number of different telescopes being used. Required so that simultaneous observations from different observatories can be used by TransitFit
- **n_filters** (*int*) – The number of different filters being used
- **n_epochs** (*int*) – The number of different epochs being used
- **limb_dark** (*str*, *optional*) –

The model of limb darkening you want to use. Accepted are

- linear
- quadratic
- squareroot
- power2
- nonlinear

Default is quadratic

- **filter_indices** (*array_like*, *optional*) – If provided, will only initialise fitting for parameters which are relevant to the filter indices given. Note that this will result in a difference between the filter indices used at the top, user level and those used within this PriorInfo
- **folded** (*bool*, *optional*) – If True, will not initialise P or t0 from the priors list. Instead will use folded_P and folded_t0 to set fixed values. Default is False
- **folded_P** (*float*, *optional*) – Required if folded is True. This is the period that the light curves are folded to
- **folded_t0** (*float*, *optional*) – Required if folded is True. This is the t0 that the light curves are folded to
- **host_radius** (*float*, *optional*) – The host radius in Solar radii. If this is provided, then will assume that the orbital separation is given in AU rather than host radii and will convert the values accordingly.

Notes

Detrending currently cannot be initialised in the prior file. It will be available as a kwarg in the pipeline function

```
transitfit.io.save_final_light_curves(lightcurves, priorinfo, results, folder='./final_light_curves',
                                     folded=False)
```

Applies detrending and normalisation to each light curve and saves to .csv

Parameters

- **lightcurves** (*array_like*, *shape* (*n_telescopes*, *n_filters*, *n_epochs*)) – An array of LightCurves. If no data exists for a point in the array then the entry should be *None*.

- **priorinfo** (`transitfit.priorinfo.PriorInfo`) – The PriorInfo object
- **results** (`dynesty.results.Results`) – The Dynesty results object, but must also have weights, cov and uncertainties as entries.
- **folder** (`str`, *optional*) – The folder to save the files to. Default is `./final_light_curves`
- **folded** (`bool`, *optional*) – If True, will assume that the lightcurves provided are folded and will change the filenames accordingly. Default is False

This python file takes `output_parameters` folder as input and provides asymmetric errors using the distribution of samples.

class `transitfit.error_analysis.ErrorLimits(output_parameters)`

Bases: `object`

Initializes the error limit analysis.

Parameters

output_parameters (`str`) – path of the `output_parameters` folder generated by TransitFit

get_errors()

Gets the upper and lower error bound on values.

handle_ttv()

Handles the case of TTV analysis.

`transitfit.error_analysis.get_quantiles_on_best_val(samples, weights, best_val)`

Generates lower and upper limit of errors for the best values. Sorts the samples and corresponding weights. Gets value of samples such that they encompass 68.27% of the samples by weight on both sides of the best value.

Parameters

- **samples** (`array`) – the sampled values for the parameter from dynesty
- **weights** (`array`) – weights of the samples
- **best_val** (`float`) – best value among the samples

Returns

the lower and upper error on the best value.

Return type

`tuple`

`transitfit.error_analysis.make_dict(val_dict, key, vals)`

makes dictionary using the given values. if the key already exists, then the values get appended.

Parameters

- **val_dict** (`dict`) – the dictionary where the values are to be added
- **key** (`str`) – the key of the value
- **vals** (`array`) – the values to be added

`transitfit.error_analysis.q_to_u(q, q_err_l, q_err_u)`

Converts `q` (Kipping parameters) to limb darkening parameters `u`. Handles only quadratic limb darkening currently.

Parameters

- **q** (`list`) – values of `q0`, `q1`
- **q_err_l** (`list`) – lower bound of errors on `q0`, `q1`

- **q_err_u** (*list*) – upper bound of errors on q0, q1

Returns

the values of u, lower bound on values, upper bound on values.

Return type

tuple

`transitfit.error_analysis.q_to_u_err(q, q_err)`

Converts error in q to errors in u parameters for quadratic limb-darkening

Parameters

- **q** (*list*) – values of q0, q1
- **q_err** (*list*) – errors on q0, q1

Returns

errors on u0, u1

Return type

list

3.8.9 Parameter handling

LightCurve objects for TransitFit

class `transitfit.lightcurve.LightCurve`(*times, flux, errors, telescope_idx=None, filter_idx=None, epoch_idx=None, curve_labels=None, telescope_array=None, filter_array=None, epoch_array=None*)

Bases: `object`

The transit data which we are trying to fit.

The LightCurve is designed to simplify dealing with detrending etc across multiple data sets. It allows us to clearly keep values pointed at specific data sets.

Parameters

- **times** (*array_like, shape (X,)*) – The times of observation
- **flux** (*array_like, shape (X,)*) – The fluxes
- **errors** (*array_like, shape (X,)*) – The absolute uncertainty on the fluxes
- **telescope_idx** (*int, optional*) – The telescope index associated with this light curve
- **filter_idx** (*int, optional*) – The filter index associated with this light curve
- **epoch_idx** (*int, optional*) – The epoch index associated with this light curve
- **curve_labels** (*array_like, shape (X,), optional*) – Used to identify if different data points come from different observations. Useful when combining curves if you want to undo that!
- **telescope_array** (*array_like, shape (X,), optional*) – Array of the telescope indices for each data point. Useful when dealing with combined curves.
- **filter_array** (*array_like, shape (X,), optional*) – Array of the filter indices for each data point. Useful when dealing with combined curves.
- **epoch_array** (*array_like, shape (X,), optional*) – Array of the epoch indices for each data point. Useful when dealing with combined curves.

bin(*cadence*, *residuals*=None)

Bins the light curve to a given observation cadence

Parameters

- **cadence** (*float*) – The observation cadence to bin to in the units of self.times
- **residuals** (*array_like*, *shape* (n_times,)) – If provided, will also bin any model residuals

Returns

- **time** (*array_like*) – The time centers of each bin
- **flux** (*array_like*) – The flux in each bin, calculated as the weighted mean of all flux values in the bin
- **err** (*array_like*) – The error on the flux of each bin

combine(*lightcurves*, *telescope_idx*=None, *filter_idx*=None, *epoch_idx*=None)

Combines the LightCurve with a list of other lightcurves which are passed to it and returns a new lightcurve containing all the data of the input curves. Note that you should probably only do this with lightcurves which have already been detrended, otherwise you might struggle to detrend the combined one.

create_detrended_LightCurve(*d_new*, *norm*, *escale*=None)

Creates a detrended LightCurve using detrend_flux() and returns it

decombine()

Splits a lightcurve according to its curve labels.

Inverse of combine

detrend_flux(*d_new*, *norm*=1, *force_normalise*=False)

When given a normalisation constant and some detrending parameters, will return the detrended flux and errors

Parameters

- **d** (*array_like*, *shape*(n_detrending_params,)) – Array of the detrending parameters to use
- **norm** (*float*, *optional*) – The normalisation constant to use. Default is 1
- **force_normalise** (*bool*, *optional*) – Override to allow normalisation to be forced if the LightCurve does not have normalisation initialised. Default is False.

Returns

- **detrend_flux** (*array_like*, *shape*(num_times)) – The detrended flux
- **detrended_errors** (*array_like*, *shape*(num_times)) – The errors on the detrended flux

estimate_normalisation_limits()

Estimates the range for fitting a normalisation constant, and also finds a reasonable first guess.

Returns

- **median_factor** (*float*) – The initial best guess factor
- **low_factor** (*float*) – The low limit on the normalisation factor
- **high_factor** (*float*) – The high limit on the normalisation factor

Notes

We use

$0.5/f_{\max} \leq c_n \leq 1.5/f_{\min}$

as the default range.

fold(*t0*, *period*, *base_t0*=None)

Folds the LightCurve so that all times are between $t0 - \text{period}/2$ and $t0 + \text{period}/2$.

If *base_t0* is provided, this will ensure that the folded lightcurve is centred on *base_t0*. This is to allow for *ttv* mode where the differences between the retrieved *t0* for each epoch must be accounted for.

returns a new LightCurve

get_phases(*t0*, *P*)

Converts times into phase given *t0* and *P* values, with *t0* at phase=0

save(*filepath*)

Saves the LightCurve to a .csv file

set_detrending(*method*, *order*=None, *function*=None, *method_idx*=None)

Sets detrending method

Parameters

- **method** (*str*) –

Accepted

- 'nth order'
- 'custom'

- **order** (*int*, *optional*) – The order of detrending to use if method is 'nth order'. Must be provided.
- **function** (*function*, *optional*) – If method is 'custom', this is the custom detrending function to use.

set_error_scaling(*scaling_limits*)

Turns on normalisation. Also estimates limits and a best initial guess.

Parameters

default_low (*float*, *optional*) – The lowest value to consider as a multiplicative normalisation constant. Default is 0.1.

Returns

- **median_factor** (*float*) – The initial best guess factor
- **low_factor** (*float*) – The low limit on the normalisation factor
- **high_factor** (*float*) – The high limit on the normalisation factor

Notes

We use

$0.5/f_{\text{max}} \leq c_n \leq 1.5/f_{\text{min}}$

as the default range, where f_{median} is the median flux value.

set_normalisation(*normalise_limits*)

Turns on normalisation. Also estimates limits and a best initial guess.

Parameters

default_low (*float*, *optional*) – The lowest value to consider as a multiplicative normalisation constant. Default is 0.1.

Returns

- **median_factor** (*float*) – The initial best guess factor
- **low_factor** (*float*) – The low limit on the normalisation factor
- **high_factor** (*float*) – The high limit on the normalisation factor

Notes

We use

$0.5/f_{\text{max}} \leq c_n \leq 1.5/f_{\text{min}}$

as the default range, where f_{median} is the median flux value.

split(*t0*, *P*, *t14*, *cutoff*=0.25, *window*=None)

Splits the LightCurve into multiple LightCurves containing a single transit. This is useful for dealing with multi-epoch observations which contain TTVs, or have long term periodic trends, since single-epoch observation trends can be approximated with polynomial fits.

Parameters

- **t0** (*float*) – The centre of a transit
- **P** (*float*) – The estimated period of the planet in days
- **t14** (*float*, *optional*) – The approximate transit duration in minutes.
- **cutoff** (*float*, *optional*) – If there are no data within $t14 * \text{cutoff}$ of $t0$, a period will be discarded. Default is 0.25
- **window** (*float*, *optional*) – If provided, data outside of the range $[t0 \pm (0.5 * t14) * \text{window}]$ will be discarded.

Returns

lightcurves – A list of LightCurve objects

Return type

list

Notes

Will reset the telescope, filter and epoch indices to None.

A class for parameters in TransitFit which can be retrieved. These are used by the PriorInfo to determine dimensionality etc.

```
class transitfit._params._GaussianParam(best, sigma, negative_allowed=True, clipped_gaussian=False)
```

Bases: `_Param`

```
from_unit_interval(u)
```

Function to convert value *u* in range (0,1], will convert to a value to be used by Batman

```
uniform_to_clipped_gaussian(u)
```

Function to convert value *u* in range (0,1], will convert to a value to be used by Batman

```
class transitfit._params._Param(value, uncertainty=None)
```

Bases: `object`

```
from_unit_interval(u)
```

```
class transitfit._params._UniformParam(low_lim, high_lim, negative_allowed=True)
```

Bases: `_Param`

```
from_unit_interval(u)
```

Function to convert value *u* in range (0,1], will convert to a value to be used by Batman

```
class transitfit._paramarray.ParamArray(name, shape, telescope_dependent, filter_dependent,  
epoch_dependent, default_value=None, lightcurves=None)
```

Bases: `object`

The `_ParamArray` is designed to handle parameters which are being fitted by TransitFit. It can deal with parameters which are being fitted over different combinations of telescope, filter, and epoch

If a parameter is lightcurve specific, providing lightcurves will ensure that only parameters where a lightcurve exists will be initialised

Parameters

- **name** (*str*) – The parameter name
- **shape** (*tuple*) – The shape of the array to use
- **telescope_dependent** (*bool*) – Set to True if the parameter varies across telescopes
- **filter_dependent** (*bool*) – Set to True if the parameter varies across filters
- **epoch_dependent** (*bool*) – Set to True if the parameter varies across epoch
- **default_value** (*float*) – The default value for the parameter
- **lightcurves** (*array-like, shape(shape), optional*) – The array of relevant light curves.

```
_generate_idx(telescope_idx, filter_idx, epoch_idx)
```

Takes the given indices and converts them into a usable index, omitting any that are given for variables that the ParamArray does not depend on and raises ValueErrors if None is provided when required.

```
add_gaussian_fit_param(mean, stdev, telescope_idx=None, filter_idx=None, epoch_idx=None,  
negative_allowed=True, clipped_gaussian=False)
```

Adds a gaussian sampled fitting parameter

add_uniform_fit_param(*low_lim, high_lim, telescope_idx=None, filter_idx=None, epoch_idx=None, negative_allowed=True*)

Adds a parameter to be fitted with uniform sampling

clipped_gaussian_transform(*u, telescope_idx=None, filter_idx=None, epoch_idx=None*)

Converts a value *u* in the range [0,1] to a physical value. Used in the dynesty routine

from_unit_interval(*u, telescope_idx=None, filter_idx=None, epoch_idx=None*)

Converts a value *u* in the range [0,1] to a physical value. Used in the dynesty routine

generate_blank_ParamArray()

Produces a ParamArray with the same telescope, filter, and epoch dependencies and shape, but with everything else initialised to None

get_flat_array()

Returns the array as a flat list. Useful for getting average values

get_value(*telescope_idx=None, filter_idx=None, epoch_idx=None*)

set_value(*value, telescope_idx=None, filter_idx=None, epoch_idx=None*)

Sets the array entry at the provided value

Value can be any object

3.8.10 General utility functions

This is a series of utility functions for TransitFit, including input validation

transitfit._utils.AU_to_host_radii(*a, R, a_err=0, R_err=0, calc_err=False*)

Converts a number in AU to a value in host radii when given the host radius *R* in Solar radii. Inverse of **host_radii_to_AU**.

transitfit._utils.calculate_logg(*host_mass, host_radius*)

Calculates $\log_{10}(g)$ for a host in units usable by TransitFit. *g* is in cm s^{-2}

Parameters

- **host_mass** (*tuple*) – The host mass in solar masses and the uncertainty
- **host_radius** (*tuple*) – The host radius in solar radii and the uncertainty

transitfit._utils.check_batches(*allow_ttv, data_files*)

Taken from `firefly/_utils.py`, with minor modifications. This was needed to prevent uneven batchsizes. Now it is not required in most of the cases, but has been kept as a failsafe.

Parameters

- **allow_ttv** (*bool*) – To allow for TTV or not.
- **data_files** (*str*) – The path to the data input .csv file, which contains the paths to the
- **data.** (*light curve*) –

Returns

The path to modified data input .csv file, which contains the paths to the light curve data. It removed some lightcurves randomly to take care of batching issue in uneven batches.

Return type

str

`transitfit._utils.estimate_t14(Rp, Rs, a, P)`

Estimates t14 in minutes, if P is in days

`transitfit._utils.get_covariance_matrix(results)`

Gets a covariance matrix from Dynesty results

Parameters

results (*dynesty.results.Results*) – The Dynesty results object, but must also have weights as an entry

Returns

cov – The covariance matrix for the results object.

Return type

np.array, shape (ndims, ndims)

`transitfit._utils.get_normalised_weights(results)`

Obtains normalised weights for the Dynesty results

Parameters

results (*dynesty.results.Results*) – The Dynesty results object

Returns

weights – The normalised weights for each sample set

Return type

np.array, shape (n_iterations,)

`transitfit._utils.host_radii_to_AU(a, R, a_err=0, R_err=0, calc_err=False)`

converts a separation in host radii into a separation in AU when given the host radius R in Solar radii. Inverse of AU_to_host_radii.

`transitfit._utils.split_lightcurve_file(path, t0, P, t14=20, cutoff=0.25, window=5, new_base_fname='split_curve')`

Split a light curve file into multiple single epoch files

Splits a multi-epoch lightcurve data file into multiple single-epoch files and saves these. This is useful for dealing with multi-epoch observations which contain TTVs, or have long term periodic trends, since single-epoch observation trends can be approximated with polynomial fits. New files are created of the form new_base_name_X

Parameters

- **path** (*str*) – The path to the light curve data to be split
- **t0** (*float*) – The centre of a transit
- **P** (*float*) – The estimated period of the planet in days
- **t14** (*float*, *optional*) – The approximate transit duration in minutes. Default is 20
- **cutoff** (*float*, *optional*) – If there are no data within t14 * cutoff of t0, a period will be discarded. Default is 0.25
- **window** (*float*, *optional*) – Data outside of the range $[t0 \pm (0.5 * t14) * window]$ will be discarded. Default is 5.
- **new_base_fname** (*str*, *optional*) – The base name for the new files, which will have numbers appended depending on the epoch. This can be used to specify a relative path for saving. Default is 'split_curve'.
- **return_names** (*bool*) –

Returns

paths – The paths to each of the new data files

Return type

array_like, shape (n_curves,)

`transitfit._utils.validate_variable_key(key)`

Checks that a key is valid for use with PriorInfo, and corrects when it is obvious what is meant. Raises KeyError if unable to correct.

`transitfit._utils.weighted_avg_and_std(values, weights, axis=-1, single_val=False)`

Calculates the weighted average and error on some data.

axis defaults to the last one (-1)

`time_conversions`

Module which uses astropy.time to convert between different time standards.

`transitfit.time_conversions.MJD_to_BJD(mjd_times, ra, dec, lat, lon, elevation, ra_unit='hourangle')`

Takes an array of times in MJD and converts to BJD, using observation and telescope data.

Parameters

- **mjd_times** (array_like, shape (N,)) – The times to be converted from MJD
- **ra** (tuple, length 3) – The right ascension of the observation in either (d, m, s) or (h, m, s) depending on if unit is 'deg' or 'hourangle'
- **dec** (tuple, length 3) – The declination of the observation in (d, m, s)
- **lat** (float or str) – The latitude of the observer in degrees or “dd:mm:ss”
- **log** (float or str) – The longitude of the observer in degrees or “dd:mm:ss”
- **elevation** (float) – The elevation of the observer in meters
- **unit** (str, optional) – The unit being used for ra. Either 'deg' for degrees or 'hourangle' for hour angle. Default is 'hourangle'.

Returns

bjd_times – The times in Barycentric Julian Date

Return type

array_like, shape (N,)

PYTHON MODULE INDEX

t

- `transitfit._ldtk_handler`, 36
- `transitfit._likelihood`, 37
- `transitfit._limb_darkening_handler`, 33
- `transitfit._paramarray`, 51
- `transitfit._params`, 51
- `transitfit._pipeline`, 21
- `transitfit._utils`, 52
- `transitfit.detrender`, 37
- `transitfit.detrending_funcs`, 38
- `transitfit.error_analysis`, 46
- `transitfit.io`, 41
- `transitfit.lightcurve`, 47
- `transitfit.output_handler`, 38
- `transitfit.priorinfo`, 31
- `transitfit.retriever`, 25
- `transitfit.time_conversions`, 54

Symbols

<code>_GaussianParam</code> (class in <code>transitfit._params</code>), 51	
<code>_Param</code> (class in <code>transitfit._params</code>), 51	
<code>_UniformParam</code> (class in <code>transitfit._params</code>), 51	
<code>_batch_to_full_idx()</code> (<code>transitfit.output_handler.OutputHandler</code> method), 38	
<code>_calculate_n_params()</code> (<code>transitfit.retriever.Retriever</code> method), 26	
<code>_convert_unit_cube()</code> (<code>transitfit.priorinfo.PriorInfo</code> method), 31	
<code>_extract_best_coeffs()</code> (<code>transitfit._ldtk_handler.LDTKHandler</code> method), 36	
<code>_fold_lightcurves()</code> (<code>transitfit.retriever.Retriever</code> method), 26	
<code>_format_indices()</code> (<code>transitfit.retriever.Retriever</code> method), 27	
<code>_full_to_subset_index()</code> (<code>transitfit.retriever.Retriever</code> method), 27	
<code>_generate_idx()</code> (<code>transitfit._paramarray.ParamArray</code> method), 51	
<code>_get_detrending_subset()</code> (<code>transitfit.retriever.Retriever</code> method), 27	
<code>_get_folding_batches()</code> (<code>transitfit.retriever.Retriever</code> method), 27	
<code>_get_lightcurve_subset()</code> (<code>transitfit.retriever.Retriever</code> method), 27	
<code>_get_non_folding_batches()</code> (<code>transitfit.retriever.Retriever</code> method), 28	
<code>_get_priors_and_curves()</code> (<code>transitfit.retriever.Retriever</code> method), 28	
<code>_get_unique_indices()</code> (<code>transitfit.retriever.Retriever</code> method), 29	
<code>_initialise_batman()</code> (<code>transitfit.output_handler.OutputHandler</code> method), 38	
<code>_initialise_best_model()</code> (<code>transitfit.output_handler.OutputHandler</code> method), 38	
<code>_initialise_dict_entry()</code> (<code>transitfit.output_handler.OutputHandler</code> method), 39	
<code>_interpret_final_results()</code> (<code>transitfit.priorinfo.PriorInfo</code> method), 31	
<code>_interpret_param_array()</code> (<code>transitfit.priorinfo.PriorInfo</code> method), 31	
<code>_plot_data()</code> (<code>transitfit.output_handler.OutputHandler</code> method), 39	
<code>_plot_samples()</code> (<code>transitfit.output_handler.OutputHandler</code> method), 39	
<code>_print_results()</code> (<code>transitfit.output_handler.OutputHandler</code> method), 39	
<code>_quicksave_result()</code> (<code>transitfit.output_handler.OutputHandler</code> method), 39	
<code>_read_data_csv()</code> (in module <code>transitfit.io</code>), 41	
<code>_read_data_txt()</code> (in module <code>transitfit.io</code>), 41	
<code>_results_dict_to_dataframe()</code> (<code>transitfit.output_handler.OutputHandler</code> method), 39	
<code>_run_batch()</code> (in module <code>transitfit.retriever</code>), 31	
<code>_run_batched_retrieval()</code> (<code>transitfit.retriever.Retriever</code> method), 29	
<code>_run_dynesty()</code> (<code>transitfit.retriever.Retriever</code> method), 29	
<code>_run_folded_retrieval()</code> (<code>transitfit.retriever.Retriever</code> method), 29	
<code>_run_full_retrieval()</code> (<code>transitfit.retriever.Retriever</code> method), 29	
<code>_save_results_dict()</code> (<code>transitfit.output_handler.OutputHandler</code> method), 39	
<code>_subset_to_full_index()</code> (<code>transitfit.retriever.Retriever</code> method), 29	
A	
<code>add_best_u()</code> (<code>transitfit.output_handler.OutputHandler</code> method), 39	
<code>add_gaussian_fit_param()</code> (<code>transitfit._paramarray.ParamArray</code> method), 51	
<code>add_gaussian_fit_param()</code> (<code>transit-</code>	

fit.priorinfo.PriorInfo method), 32
 add_uniform_fit_param() (*transitfit._paramarray.ParamArray* method), 51
 add_uniform_fit_param() (*transitfit.priorinfo.PriorInfo* method), 32
 AU_to_host_radii() (in module *transitfit._utils*), 52

B

bin() (*transitfit.lightcurve.LightCurve* method), 47

C

calculate_logg() (in module *transitfit._utils*), 52
 check_batches() (in module *transitfit._utils*), 52
 clipped_gaussian_transform() (*transitfit._paramarray.ParamArray* method), 52
 combine() (*transitfit.lightcurve.LightCurve* method), 48
 combine_results_dicts() (*transitfit.output_handler.OutputHandler* method), 39
 convert_qtoug() (*transitfit._limb_darkening_handler.LimbDarkeningHandler* method), 34
 convert_qtoug_with_errors() (*transitfit._limb_darkening_handler.LimbDarkeningHandler* method), 34
 convert_utoq() (*transitfit._limb_darkening_handler.LimbDarkeningHandler* method), 34
 create_detrended_LightCurve() (*transitfit.lightcurve.LightCurve* method), 48

D

decombine() (*transitfit.lightcurve.LightCurve* method), 48
 detrend_flux() (*transitfit.lightcurve.LightCurve* method), 48
 detrending_limits (*transitfit.retriever.Retriever* attribute), 30
 DetrendingFunction (class in *transitfit.detrender*), 37

E

ErrorLimits (class in *transitfit.error_analysis*), 46
 estimate_normalisation_limits() (*transitfit.lightcurve.LightCurve* method), 48
 estimate_t14() (in module *transitfit._utils*), 52
 estimate_values() (*transitfit._ldtk_handler.LDTHandler* method), 36

F

find_likelihood() (*transitfit._likelihood.LikelihoodCalculator* method), 37

find_likelihood_parallel_processed() (*transitfit._likelihood.LikelihoodCalculator* method), 37

fit_detrending() (*transitfit.priorinfo.PriorInfo* method), 32
 fit_limb_darkening() (*transitfit.priorinfo.PriorInfo* method), 32
 fit_normalisation() (*transitfit.priorinfo.PriorInfo* method), 33
 fold() (*transitfit.lightcurve.LightCurve* method), 49
 from_unit_interval() (*transitfit._paramarray.ParamArray* method), 52
 from_unit_interval() (*transitfit._params._GaussianParam* method), 51
 from_unit_interval() (*transitfit._params._Param* method), 51
 from_unit_interval() (*transitfit._params._UniformParam* method), 51

G

generate_blank_ParamArray() (*transitfit._paramarray.ParamArray* method), 52
 get_best_vals() (*transitfit.output_handler.OutputHandler* method), 39
 get_covariance_matrix() (in module *transitfit._utils*), 53
 get_errors() (*transitfit.error_analysis.ErrorLimits* method), 46
 get_filter_path() (in module *transitfit.io*), 41
 get_flat_array() (*transitfit._paramarray.ParamArray* method), 52
 get_latex_friendly_labels() (*transitfit.priorinfo.PriorInfo* method), 33
 get_normalised_weights() (in module *transitfit._utils*), 53
 get_phases() (*transitfit.lightcurve.LightCurve* method), 49
 get_quantiles_on_best_val() (in module *transitfit.error_analysis*), 46
 get_required_coefficients() (*transitfit._limb_darkening_handler.LimbDarkeningHandler* method), 34
 get_results_dict() (*transitfit.output_handler.OutputHandler* method), 40
 get_value() (*transitfit._paramarray.ParamArray* method), 52

H

handle_ttv() (*transitfit.error_analysis.ErrorLimits* method), 46
 host_radii_to_AU() (in module *transitfit._utils*), 53

- I**
- `initialise_ldtk()` (transitfit._limb_darkening_handler.LimbDarkeningHandler method), 34
- L**
- `ldtk_estimate()` (transitfit._limb_darkening_handler.LimbDarkeningHandler method), 35
- `ldtk_lnlike()` (transitfit._limb_darkening_handler.LimbDarkeningHandler method), 35
- `LDTKHandler` (class in transitfit._ldtk_handler), 36
- `LightCurve` (class in transitfit.lightcurve), 47
- `LikelihoodCalculator` (class in transitfit._likelihood), 37
- `LimbDarkeningHandler` (class in transitfit._limb_darkening_handler), 33
- `lnlike()` (transitfit._ldtk_handler.LDTKHandler method), 37
- M**
- `make_dict()` (in module transitfit.error_analysis), 46
- `MJD_to_BJD()` (in module transitfit.time_conversions), 54
- `module`
- `transitfit._ldtk_handler`, 36
 - `transitfit._likelihood`, 37
 - `transitfit._limb_darkening_handler`, 33
 - `transitfit._paramarray`, 51
 - `transitfit._params`, 51
 - `transitfit._pipeline`, 21
 - `transitfit._utils`, 52
 - `transitfit.detrender`, 37
 - `transitfit.detrending_funcs`, 38
 - `transitfit.error_analysis`, 46
 - `transitfit.io`, 41
 - `transitfit.lightcurve`, 47
 - `transitfit.output_handler`, 38
 - `transitfit.priorinfo`, 31
 - `transitfit.retriever`, 25
 - `transitfit.time_conversions`, 54
- N**
- `NthOrderDetrendingFunction` (class in transitfit.detrending_funcs), 38
- O**
- `OutputHandler` (class in transitfit.output_handler), 38
- P**
- `ParamArray` (class in transitfit._paramarray), 51
- `parse_data_path_list()` (in module transitfit.io), 41
- `parse_filter_list()` (in module transitfit.io), 42
- `parse_priors_list()` (in module transitfit.io), 42
- `plot_final_light_curves()` (transitfit.output_handler.OutputHandler method), 40
- `print_results()` (in module transitfit.io), 43
- `PriorInfo` (class in transitfit.priorinfo), 31
- Q**
- `q_to_u()` (in module transitfit.error_analysis), 46
- `q_to_u_err()` (in module transitfit.error_analysis), 47
- R**
- `read_data_file()` (in module transitfit.io), 43
- `read_data_path_array()` (in module transitfit.io), 43
- `read_filter_info()` (in module transitfit.io), 43
- `read_input_file()` (in module transitfit.io), 44
- `read_priors_file()` (in module transitfit.io), 44
- `Results` (class in transitfit.output_handler), 41
- `ResultsException` (class in transitfit.output_handler), 41
- `Retriever` (class in transitfit.retriever), 25
- `run_retrieval()` (in module transitfit._pipeline), 21
- `run_retrieval()` (transitfit.retriever.Retriever method), 30
- S**
- `save()` (transitfit.lightcurve.LightCurve method), 49
- `save_complete_results()` (transitfit.output_handler.OutputHandler method), 40
- `save_final_light_curves()` (in module transitfit.io), 45
- `save_final_light_curves()` (transitfit.output_handler.OutputHandler method), 40
- `save_results()` (transitfit.output_handler.OutputHandler method), 40
- `set_detrending()` (transitfit.lightcurve.LightCurve method), 49
- `set_error_scaling()` (transitfit.lightcurve.LightCurve method), 49
- `set_error_scaling()` (transitfit.priorinfo.PriorInfo method), 33
- `set_normalisation()` (transitfit.lightcurve.LightCurve method), 50
- `set_value()` (transitfit._paramarray.ParamArray method), 52
- `setup_priors()` (in module transitfit.priorinfo), 33
- `split()` (transitfit.lightcurve.LightCurve method), 50
- `split_lightcurve_file()` (in module transitfit._utils), 53

T

`transitfit._ldtk_handler`
 module, [36](#)
`transitfit._likelihood`
 module, [37](#)
`transitfit._limb_darkening_handler`
 module, [33](#)
`transitfit._paramarray`
 module, [51](#)
`transitfit._params`
 module, [51](#)
`transitfit._pipeline`
 module, [21](#)
`transitfit._utils`
 module, [52](#)
`transitfit.detrender`
 module, [37](#)
`transitfit.detrending_funcs`
 module, [38](#)
`transitfit.error_analysis`
 module, [46](#)
`transitfit.io`
 module, [41](#)
`transitfit.lightcurve`
 module, [47](#)
`transitfit.output_handler`
 module, [38](#)
`transitfit.priorinfo`
 module, [31](#)
`transitfit.retriever`
 module, [25](#)
`transitfit.time_conversions`
 module, [54](#)

U

`uniform_to_clipped_gaussian()` (*transitfit._params._GaussianParam* method), [51](#)
`update_params()` (*transitfit._likelihood.LikelihoodCalculator* method), [37](#)

V

`validate_variable_key()` (*in module transitfit._utils*), [54](#)

W

`weighted_avg_and_std()` (*in module transitfit._utils*), [54](#)