TLS Documentation

Michael Hippke, Rene Heller

Jun 04, 2020

Contents


Transit Least Squares

CHAPTER 1

Installation

TLS can be installed conveniently using pip:

```
pip install transitleastsquares
```

If you have multiple versions of Python and pip on your machine, make sure to use pip3. Try:

```
pip3 install transitleastsquares
```

The latest version can be pulled from github:

```
git clone https://github.com/hippke/tls.git
cd tls
python setup.py install
```

If the command `python` does not point to Python 3 on your machine, you can try to replace the last line with `python3 setup.py install`. If you don't have `git` on your machine, you can find installation instructions here.

1.1 Compatibility

TLS has been tested to work with Python 2.7, 3.5, 3.6, 3.7. It works on Python 2.7, but only in single-threaded mode.

CHAPTER 2

Changelog

This describes changes to TLS.

The versioning scheme is: major.minor.revision

major Will be increased when the API (the interface) changes in an incompatible way. Will be documented in this changelog.

minor Will be increased when adding functionality in a backwards-compatible manner. Will be documented in this changelog.

revision Will be increased for backwards-compatible bug fixes and very minor added functionality. Will not always be documented in this changelog.

2.1 Version 1.0.23 (12 March 2019)

Fixed A bug in the post-detection statistics which caused a delay in large dataset (e.g., Kepler K1)

2.2 Version 1.0.21 (14 February 2019)

Fixed A bug in `cleaned_array` which caused an error in case invalid `dy` values were supplied. Relevant for TESS FITS files. These should now run including uncertainties.

Added New statistics: `before_transit_count`, `in_transit_count`, `after_transit_count` yield the number of data points in a bin of length transit duration before, in and after the phase-folded transit.

Added New parameter `show_progress_bar` *(bool, default: True)* When set to `False`, no progress bar (using `tqdm`) will be shown

Added Python 2 compatibility. For now, it is only single-core.

2.3 Version 1.0.20 (11 February 2019)

Fixed A bug which erroneously interchanged *power* and *power_raw*

2.4 Version 1.0.19 (10 February 2019)

Fixed A bug in the calculation of statistics which caused TLS to stall in some cases

2.5 Version 1.0.17 (07 February 2019)

Fixed A bug in the calculation of the SNR statistic (post-fit statistics)

Changed Major code refactoring

Added Extensive test suite

Added Warnings for the user, e.g. in case inputs are dubious. Numpy warnings are now a bug (if any left) and no longer hidden.

Added Improved command line interface (now has its own command). Added all recent new functionality to the command line interface (all except custom transit shapes)

2.6 Version 1.0.16 (29 January 2019)

Fixed A bug which caused to return an empty SDE-ogram if very small uncertainties `dy` were provided.

Changed Switched linear interpolation code of model shapes to a numba implementation. It is 2x faster, 20ms –> 10ms which is currently irrelevant if the shape is calculated only once per light curve, but will become relevant when the compensation for morphological light-curve distortions will be implemented. Then, the shapes will be re-calculated many times for a range of periods. Another advantage is that the dependency on scipy can now be removed. Scipy is still required for testing, however.

2.7 Version 1.0.15 (27 January 2019)

Changed If no transits fits were performed during a search, a flat SDE-ogram and SDE=0 are returned, and a warning is raised. Previous behavior was to raise an exception and quit. This can happen if `transit_depth_min` is set to a large value (e.g., 1000 ppm) and the light curve is flat (e.g., Kepler-quality with good detrending and no transits), so that the threshold causes no transit fits to be performed.

Changed Only useful warnings are printed to the user console. Internal processing issues (e.g., NaN values) are now hidden.

Changed Catalog information (e.g., from the Kepler K2 EPIC catalog) which includes missing values now returns `NaN` values. Previously, `--` was returned. The `NaN` values must still be evaluated by the user before feeding them into a TLS model.

Changed Catalog information is now entirely pulled using AstroQuery, from Vizier (Kepler K1, K2) and MAST. Dependency to package `kplr` has been dropped. This increases reliability as the MAST API was unstable in the past.

Fixed A bug in the command-line version was fixed which caused the search to quit under certain circumstances.

2.8 Version 1.0.14. (24 January 2019)

Added Automatically run `cleaned_array` before performing a search

Added New return value: `results.transit_depths_uncertainties`

Added New parameter: `use_threads`

Changed `period_grid` limited to physically plausible values to avoid generating empty or extremely large grids

Removed `numpy.set_printoptions(threshold=numpy.nan)` which fails in numpy 1.16+ (the latest version as of 24 Jan 2019)

2.9 Version 1.0 (01 January 2018)

Initial release.

CHAPTER 3

Python Interface

This describes the Python interface to TLS.

3.1 Define data for a search

class transitleastsquares.**model**(*t*, *y*, *dy*)

t *(array)* Time series of the data (**in units of days**)

y *(array)* Flux series of the data, so that `1` is nominal flux (out of transit) and `0` is darkness. A transit may be represented by a flux of e.g., `0.99`

dy *(array, optional)* Measurement errors of the data

Note: TLS works best with a constant cadence. Variations in the cadence generally have a negligible impact on detection efficiency, but may result in incorrect transit duration estimates. Small variations, e.g. from the barycentering of the Kepler satellite, can usually be neglected.

Note: Gaps in the data during a transit may decrease detection efficiency. The effect becomes negligible for a large number of transits (e.g., 20), but may be relevant in case of a few (e.g., 3) transits. Then, sorting the data points in phase space may result in an asymmetric transit shape, reducing detection efficiency when using normal (symmetric) transit shape templates.

Note: The time series must be **in units of days**. This is not a chicanery, but a necessity based on the physical model which is used to reduce the parameter space. The unit *day* is a logic choice as orbital periods are typically given in *days*. The Kepler mission also used this unit.

3.2 Define parameters and run search

class transitleastsquares.**power**(*parameters*)

Parameters used for the period search grid and the transit duration search grid. All parameters are optional.

R_star *(float, default: 1.0)* Stellar radius (in units of solar radii)

R_star_min *(float, default: 0.13)* Minimum stellar radius to be considered (in units of solar radii)

R_star_max *(float, default: 3.5)* Maximum stellar radius to be considered (in units of solar radii)

M_star *(float, default: 1.0)* Stellar mass (in units of solar masses).

M_star_min *(float, default: 0.1)* Minimum stellar mass to be considered (in units of solar masses)

M_star_max *(float, default: 1.0)* Maximum stellar mass to be considered (in units of solar masses)

period_min *(float)* Minimum trial period (in units of days). If none is given, the limit is derived from the Roche limit

period_max *(float)* Maximum trial period (in units of days). Default: Half the duration of the time series

n_transits_min *(int, default: 2)* Minimum number of transits required. Overrules `period_max=time_span/n_transits_min`

Note: A larger range of stellar radius and mass allows for a wider variety of transits to be found at the expense of computational effort

---

Physical parameters to create a Mandel & Agol (2002) transit model using a subset of the batman module and syntax (Kreidberg 2015). Available defaults are described below.

per *(float)* Orbital period (in units of days). Default: X.

rp *(float)* Planet radius (in units of stellar radii). Default: X.

a *(float)* Semi-major axis (in units of stellar radii). Default: X.

inc *(float)* Orbital inclination (in degrees). Default: 90.

b *(float)* Orbital impact parameter as the sky-projected distance between the centre of the stellar disc and the centre of the planetary disc at conjunction. If set, overrules `inc=degrees(arccos(b/a)`. Default: 0.

ecc *(float)* Orbital eccentricity. Default: 0.

w *(float)* Argument of periapse (in degrees). Default: 90.

u *(array)* List of limb darkening coefficients. Default: [0.4804, 0.1867] (a G2V star in the Kepler bandpass).

limb_dark *(str)* Limb darkening model (choice of `nonlinear`, `quadratic`, `exponential`, `logarithmic`, `squareroot`, `linear`, `uniform`, or `power2`. Default: `quadratic`.

Available defaults for the physical parameters of the transit model. When set, the individual parameters are overruled.

transit_template *(str)* Choice of `default`, `grazing`, and `box`.

Parameters to balance detection efficiency and computational requirements:

duration_grid_step *(float, default: 1.1)* Grid step width between subsequent trial durations, so that $\mathrm{dur}_{n+1} = \mathrm{dur}_n \times \mathrm{duration\_grid\_step}$. With the default value of 1.1, each subsequent trial duration is longer by 10%

transit_depth_min *(float, default: 10 ppm)* Shallowest transit depth to be fitted. Transit depths down to half the transit_depth_min can be found at reduced sensitivity. A reasonable value should be estimated from the data to balance sensitivity and avoid fitting the noise floor. Overfitting may cause computational requirements larger by a factor of 10. For reference, the shallowest known transit is 11.9 ppm (Kepler-37b, Barclay et al. 2013)

oversampling_factor *(int, default: 3)* Oversampling of the period grid to avoid that the true period falls in between trial periods and is missed.

T0_fit_margin *(float, default: 0.01)* Acceptable error margin of the mid-transit time T0. Unit: fraction of the transit duration (0.01 is 1%). For small datasets (e.g., Kepler K2; generally: <10k datapoints), this can be set to 0 with minor speed penalty (seconds). Then, every single cadence is sampled. In data with many cadences, however, this can take very long and can have negligible benefits. As an example, consider a Kepler LC light curve of 60000 points, with a maximum fractional transit duration $T_{14}/P = 0.12$. The longest phase-folded transit signal to be tested is then 7200 points long. With Kepler noise, shifting this signal point-by-point is overkill. Shifting by 1% of the transit duration would result in shifts of 72 cadences for this specific signal.

Note: Higher `oversampling_factor` increases the detection efficiency at the cost of a linear increase in computational effort. Reasonable values may be 2-5 and should be tested empirically for the actual data. An upper limit can be found when the period step is smaller than the cadence, so that the error from shifting the model by one data point in phase dominates over the period trial shift. For a planet with a 365-day period orbiting a solar mass and radius star, this parity is reached for `oversampling_factor=9` at 30 min cadence (Kepler LC). Shorter periods have reduced oversampling benefits, as the cadence becomes a larger fraction of the period.

Parameters to adjust the computational load and the user experience:

use_threads *(int)* Number of parallel threads to be used. A processor like the Intel Core i7-8700K has 6 cores and can run 12 threads in parallel using hyperthreading. Setting `use_threads=12` will cause a full load. If no parameter is given, TLS determines the number of available threads and uses the maximum available (in this case: 12).

show_progress_bar *(bool, default: True)* When set to `False`, no progress bar (using `tqdm`) will be shown

---

Note: Multi-threading (‘‘use_threads>1‘) only works with TLS running on Python 3 as of now. On Python 2, TLS should work, but will fall back to single-core.

3.3 Return values

The TLS spectra:

periods *(array)* The period grid used in the search

power *(array)* The power spectrum per period as defined in the TLS paper. We recommend to use this spectrum to assess transit signals. It is the median-smoothed `power_raw` spectrum.

power_raw *(array)* The raw power spectrum (without median smoothing) as defined in the TLS paper

SR *(array)* Signal residue similar to the BLS SR

chi2 *(array)* Minimum chi-squared (χ^2) per period

chi2red *(array)* Minimum chi-squared per degree of freedom ($\chi^2_\nu = \chi^2/\nu$) per period, where $\nu = n - m$ with n as the number of observations, and $m = 4$ as the number of fitted parameters (period, T0, transit duration, transit depth).

The TLS statistics:

SDE *(float)* Maximum of `power`

SDE_raw *(float)* Maximum of `power_raw`

chi2_min *(float)* Minimum of `chi2`

chi2red_min *(float)* Minimum of `chi2red`

Additional transit statistics based on the `power` spectrum:

period *(float)* Period of the best-fit signal

period_uncertainty *(float)* Uncertainty of the best-fit period (half width at half maximum)

T0 *(float)* Mid-transit time of the first transit within the time series

duration *(float)* Best-fit transit duration

depth *(float)* Best-fit transit depth (measured at the transit bottom)

depth_mean *(tuple of floats)* Transit depth measured as the mean of all intransit points. The second value is the standard deviation of these points multiplied by the square root of the number of intransit points

depth_mean_even *(tuple of floats)* Mean depth and uncertainty of even transits (1, 3, . . .)

depth_mean_odd *(tuple of floats)* Mean depth and uncertainty of odd transits (2, 4, . . .)

rp_rs *(float)* Radius ratio of planet and star using the analytic equations from Heller 2019

transit_depths *(array)* Mean depth of each transit

transit_depths_uncertainties *(array)* Uncertainty (1-sigma) of the mean depth of each transit

snr *(float)* Signal-to-noise ratio. Definition: $\mathrm{SNR} = \frac{d}{\sigma_o} n^{1/2}$ with d as the mean transit depth, σ as the standard deviation of the out-of-transit points, and n as the number of intransit points (Pont et al. 2006)

snr_per_transit *(array)* Signal-to-noise ratio per individual transit

snr_pink_per_transit *(array)* Signal-to-pink-noise ratio per individual transit as defined in Pont et al. (2006)

odd_even_mismatch *(float)* Significance (in standard deviations) between odd and even transit depths. Example: A value of 5 represents a $5\,\sigma$ confidence that the odd and even depths have different depths

transit_times *(array)* The mid-transit time for each transit within the time series

per_transit_count *(array)* Number of data points during each unique transit

transit_count *(int)* The number of transits

distinct_transit_count *(int)* The number of transits with intransit data points

empty_transit_count *(int)* The number of transits with no intransit data points

FAP *(float)* The false alarm probability for the SDE assuming white noise. Returns NaN for FAP>0.1.

before_transit_count *(int)* * Number of data points in transit (phase-folded)

in_transit_count *(int)* Number of data points in a bin of length transit duration before transit (phase-folded)

after_transit_count *(int)* Number of data points in a bin of length transit duration after transit (phase-folded)

Time series model for visualization purpose:

model_lightcurve_time *(array)* Time series spanning `t`, but without gaps, and oversampled by a factor of 5

model_lightcurve_model *(array)* Model flux value of each point in `model_lightcurve_time`

Phase-folded model for visualization purpose:

folded_phase *(array)* Phase of each data point `y` when folded to `period` so that the transit is at `folded_phase=0.5`

folded_y *(array)* Data flux of each point

folded_dy *(array)* Data uncertainty of each point

model_folded_phase *(array)* Linear array `[0..1]` which can be used to plot the `model_folded_model`. This is a separate array from `folded_phase`, because the data may have gaps which would prevent plotting the complete model. This array here is complete.

model_folded_model *(array)* Model flux of each point in `model_folded_phase`

Note: The models are oversampled and calculated for each point in time and phase. This way, the models cover the entire time series (phase space), including gaps. Thus, these curves are not exact representations of the models used during the search. They are intended for visualization purposes.

3.4 Period grid

When searching for sine-like signals, e.g. using Fourier Transforms, it is optimal to uniformly sample the trial frequencies. This was also suggested for BLS (Kovács et al. 2002). However, when searching for transit signals, this is not optimal due to the transit duty cycle which changes as a function of the planetary period due to orbital mechanics. The optimal period grid, compared to a linear grid, reduces the workload (at the same detection efficiency) by a factor of a few. The optimal frequency sampling as a function of stellar mass and radius was derived by Ofir (2014) as

$$N_{\mathrm{freq,optimal}} = \left(f_{\mathrm{max}}^{1/3} - f_{\mathrm{min}}^{1/3} + \frac{A}{3} \right) \frac{3}{A}$$

with

$$A = \frac{(2\pi)^{2/3}}{\pi} \frac{R}{(GM)^{1/3}} \frac{1}{S \times OS}$$

where M and R are the stellar mass and radius, G is the gravitational constant, S is the time span of the dataset and OS is the oversampling parameter to ensure that the peak is not missed between frequency samples. The search edges can be found at the Roche limit,

$$f_{\mathrm{max}} = \frac{1}{2\pi} \sqrt{\frac{GM}{(3R)^3}}; f_{\mathrm{min}} = 2/S$$

`period_grid`(*parameters*)

R_star Stellar radius (in units of solar radii)

M_star Stellar mass (in units of solar masses)

time_span Duration of time series (in units of days)

period_min Minimum trial period (in units of days). Optional.

period_max Maximum trial period (in units of days). Optional.

oversampling_factor Default: 2. Optional.

Returns: a 1D array of float values representing a grid of trial periods in units of days.

Example usage:

```
from transitleastsquares import period_grid
periods = period_grid(R_star=1, M_star=1, time_span=400)
```

returns a period grid with 32172 values:

```
[200, 199.889, 199.779, ..., 0.601, 0.601, 0.601]
```

Note: TLS calls this function automatically to derive its period grid. Calling this function separately can be useful to employ a classical BLS search, e.g., using the astroPy BLS function.

Note: To avoid generating an infinitely large period_grid, parameters are auto-enforced to the ranges `0.1 < R_star < 10000` and `0.01 < M_star < 1000`. Some combinations of mostly implausible values, such as `R_star=1` with `M_star=5` yield empty period grids. If the grid size is less than 100 values, the function returns the default grid `R_star=M_star=1`. Very short time series (less than a few days of duration) default to a grid size with a span of 5 days.

3.5 Priors for stellar parameters

This function provides priors for stellar mass, radius, and limb darkening for stars observed during the Kepler K1, K2 and TESS missions. It is planned to extend this function for past and future missions such as CHEOPS and PLATO.

`catalog_info`(*EPIC_ID or TIC_ID*)

EPIC_ID *(int)* The EPIC catalog ID (K2, Ecliptic Plane Input Catalog)

TIC_ID *(int)* The TIC catalog ID (TESS Input Catalog)

KIC_ID *(int)* The Kepler Input Catalog ID (Kepler K1 Input Catalog)

Returns

ab *(tuple of floats)* Quadratic limb darkening parameters a, b

mass *(float)* Stellar mass (in units of solar masses)

mass_min *(float)* 1-sigma upper confidence interval on stellar mass (in units of solar mass)

mass_max *(float)* 1-sigma lower confidence interval on stellar mass (in units of solar mass)

radius *(float)* Stellar radius (in units of solar radii)

radius_min *(float)* 1-sigma upper confidence interval on stellar radius (in units of solar radii)

radius_max *(float)* 1-sigma lower confidence interval on stellar radius (in units of solar radii)

Note: The matching between the stellar parameter table and the limb darkening table is performed by first finding the nearest $T_{\rm eff}$, and subsequently the nearest $\log g$.

Note: Data sources:

K1 data are pulled from the catalog for Revised Stellar Properties of Kepler Targets (Mathur et al. 2017) with limb darkening coefficients from Claret et al. (2012, 2013). Data are pulled from Vizier using AstroQuery and matched to limb darkening values saved locally in a CSV file within the TLS package.

K2 data are collated from the K2 Ecliptic Plane Input Catalog (Huber et al. 2016) with limb darkening coefficients from Claret et al. (2012, 2013). Data are pulled from Vizier using AstroQuery and matched to limb darkening values saved locally in a CSV file within the TLS package.

TESS data are collated from the TESS Input Catalog (TIC, Stassun et al. 2018) with limb darkening coefficients from Claret et al. (2017). TIC data are pulled from MAST and matched to limb darkening values saved locally in a CSV file within the TLS package.

Warning: Upper and lower confidence intervals may be identical. Radius confidence interval may be identical to the radius. Values not available in the catalog are returned as `None`. When feeding these values to TLS, make sure to validate accordingly.

Example usage:

```
ab, R_star, R_star_min, R_star_max, M_star, M_star_min, M_star_max = catalog_
↪info(EPIC_ID=211611158)
print('Quadratic limb darkening a, b', ab[0], ab[1])
print('Stellar radius', R_star, '+', R_star_max, '-', R_star_min)
print('Stellar mass', M_star, '+', M_star_max, '-', M_star_min)
```

produces these results:

```
Quadratic limb darkening a, b 0.4899 0.1809
Stellar radius 1.055 + 0.12 - 0.1
Stellar mass 1.267 + 0.64 - 0.286
```

Note: Missing catalog entries will be returned as NaN values. These have to be treated on the user side.

3.6 Transit mask

Can be used to plot in-transit points in a different color, or to cleanse the data from a transit signal before a subsequent TLS run to search for further planets.

`transit_mask`(*t*, *period*, *duration*, *T0*)

t *(array)* Time series of the data (in units of days)

period *(float)* Transit period e.g. from results: `period`

duration *(float)* Transit duration e.g. from results: `duration`

T0 *(float)* Mid-transit of first transit e.g. from results: `T0`

Returns

intransit *(numpy array mask)* A numpy array mask (of True/False values) for each data point in the time series. `True` values are in-transit.

Example usage:

```
intransit = transit_mask(t, period, duration, T0)
print(intransit)
>>> [False False False ...]
plt.scatter(t[in_transit], y[in_transit], color='red')  # in-transit points in red
plt.scatter(t[~in_transit], y[~in_transit], color='blue')  # other points in blue
```

3.7 Data cleansing

TLS may not work correctly with corrupt data, such as arrays including values as NaN, None, infinite, or negative. Masked numpy arrays may also be problematic, e.g., when performing a `transit_mask`. When in doubt, it is recommended to clean the data from masks and non-floating point values. For this, TLS offers a convenience function:

`cleaned_array`(*t*, *y*, *dy*)

t *(array)* Time series of the data (in units of days)

y *(array)* Flux series of the data

dy *(array, optional)* Measurement errors of the data

Returns

Cleaned arrays, where values of type NaN, None, +-inf, and negative have been removed, as well as masks. Removed values make the output arrays shorter.

Example usage:

```
from transitleastsquares import cleaned_array
dirty_array = numpy.ones(10, dtype=object)
time_array = numpy.linspace(1, 10, 10)
dy_array = numpy.ones(10, dtype=object)
dirty_array[1] = None
dirty_array[2] = numpy.inf
dirty_array[3] = -numpy.inf
dirty_array[4] = numpy.nan
dirty_array[5] = -99
print(time_array)
print(dirty_array)

>>> [ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
>>> [1 None inf -inf nan -99 1 1 1 1]

t, y, dy = cleaned_array(time_array, dirty_array, dy_array)
print(t)
print(y)
>>> [ 1.  7.  8.  9. 10.]
>>> [1. 1. 1. 1. 1.]
```

3.8 Data resampling (binning)

TLS run times are strongly dependent on the amount of data. Very roughly, an increase in the data volume by one order of magnitude results in a run time increase of two orders of magnitude (see paper Figure 9).

For a first quick look, or for short cadence data, it may be adequate to down-sample (bin) the data. In general, binning is adequate if there are many data points between two phase grid points at the critical phase sampling.

To bin the data, TLS offers a convenience function:

resample(*t*, *y*, *dy*, *factor*)

t *(array)* Time series of the data (in units of days)

y *(array)* Flux series of the data

dy *(array, optional)* Measurement errors of the data

factor *(float, optional, default: 2.0)* Binning factor

Returns

Resampled arrays of length `len(t)*int(1/factor)`, where the flux (and optionally, dy) values are binned by linear interpolation.

Example usage:

```
from transitleastsquares import resample
time_new, flux_new = resample(time, flux, factor=3.0)
```

Note: Values of type (NaN, None, +-inf, negative, or empty) lead to undefined behavior. It is recommended to first use `cleaned_array` if needed.

CHAPTER 4

Command line interface

This describes the command line interface to TLS. After installation, you can call it from the command line.

4.1 Usage

Syntax:

```
transitleastsquares [-h] [-o OUTPUT] [-c CONFIG] lightcurve
```

Minimum example:

```
transitleastsquares test_data.csv
```

Maximum example:

```
transitleastsquares test_data.csv --config=tls_config.cfg --output=results.csv
```

Note: In the current TLS version, custom transit shapes can not be defined with the command line interface. If you have a use case for more complex searches using the command line interface, please open an issue on Github and I will add it to the next version.

4.2 Config file

Syntax:

```
[Grid]
R_star = 1
R_star_min = 0.8
```

(continues on next page)

(continued from previous page)

```
R_star_max = 1.2
M_star = 1
M_star_min = 0.8
M_star_max = 1.2
period_min = 0
period_max = 1e10
n_transits_min = 3

[Template]
transit_template = default

[Speed]
duration_grid_step = 1.1
transit_depth_min = 10e-6
oversampling_factor = 2
T0_fit_margin = 0.01
use_threads = 4

[File]
delimiter = ,
```

4.3 Output

After a successful TLS run, 2 files are generated: :statistics: `lightcurve filename + _statistics.csv` :SDE-ogram: `lightcurve filename + _power.csv`

CHAPTER 5

FAQ

Frequently asked questions.

5.1 Is TLS always better than BLS?

No, but almost always!

- For >99.9% of all known transiting planets, TLS recovers the transits at a higher significance than BLS, as a transit is almost always better fit to the data than a box. This assumes limb-darkening estimates from the usual catalogs to be used with TLS. If no limb-darkening estimates are available, TLS is better for >99% of the known planets.
- The remaining cases are mostly grazing transits (V-shaped), for which TLS offers a dedicated template to maximize sensitivity (see tutorials and documentation)
- A very few real-life cases are limb-darkened transits which are distorted by noise so that they occur to be more box-like than transit-like. On average, however, this is very rare. If you like to search for box-like transits, you can also use TLS and choose a trapezoid-shaped template (again, see tutorials and documentation). Using TLS over BLS to search for box-like transit makes sense, because TLS offers the optimal period grid and optimal duration grid (assuming you have prior information on stellar radius and mass from the K2, EPIC, TESS etc. catalog). Also, TLS searches for a *true* trapezoid (with steep ingress and egress), whereas BLS searches for an (unphysical) step function.
- In terms of recovery rate for a chosen false alarm rate: From an experiment of 10k white noise injection and retrievals, we find that for any threshold, the recovery rate of true positives (the planets) is *always* better for TLS, compared to BLS. For example, the recovery rate at a false alarm threshold of 1% is better for TLS than for BLS, and this holds for any other threshold, such as 0.1%, 0.01% etc. Setting the SDE threshold is similar to BLS (see next section).

5.2 False alarm probability

From an experiment with >10000 white noise-only TLS search runs, we can estimate false alarm probabilities (FAP) as follows:

1-FAP	SDE
0.9	5.7
0.95	6.1
0.99	7.0
0.999	8.3
0.9999	9.1

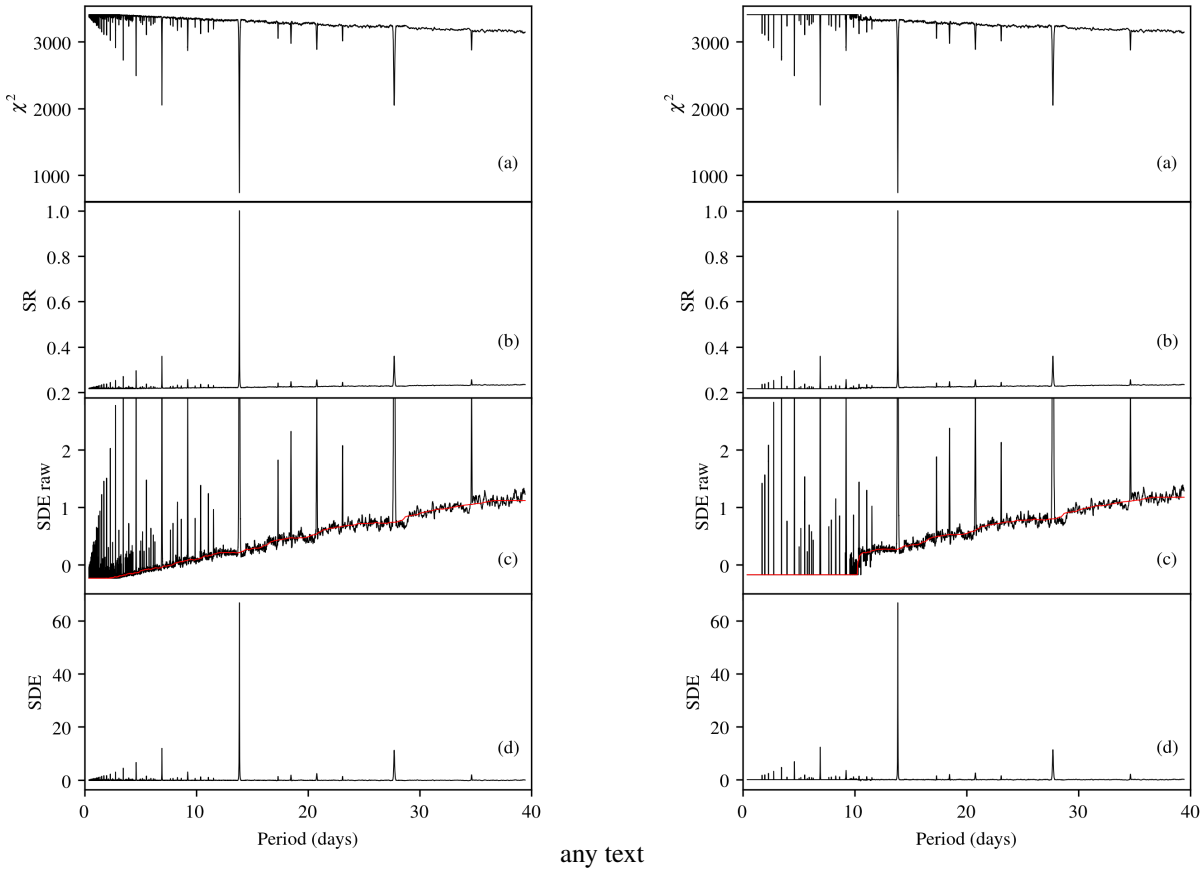
In noise-only data, 1% of the observed cases (1-FAP=0.99) had an SDE>7.0. If an SDE of 9.1 is observed in a data set, the probability of this happening from noise fluctuations is 0.01%. This assumes Gaussian white noise. Real data often has partially correlated (red) noise. Then, the FAP estimates are too optimistic, i.e., high SDE values will occur more often than measured in the experiment. Vice versa, the SDE values per given FAP value will be higher in red noise.

TLS returns the FAP value per SDE as `results.FAP` (see Python interface).

5.3 Truncation of the power spectrum

The Figure below (left panel) is taken from the paper (Figure 3) and shows (a): the χ^2 distribution (b): The signal residue (c): the raw signal detection efficiency and (d): the signal detection efficiency (SDE) used by TLS, smoothed with a walking median. This plot was made using the default parameters in TLS.

In the right panel, the only change is `transit_depth_min=200*10**-6`. That is, we decide not to fit any transits shallower than 200ppm (instead of 10ppm). As a consequence, no transits were fit for many short periods (these are smoother in phase space). The resulting spectrum contains maximum χ^2 values (where the signal is taken as unity) for many periods, resulting in SDE values of zero. With a lower baseline, the actual SDE peaks may be **higher** (remember: the SDE power spectrum is normalized to its standard deviation). Despite the higher peaks, the information content is lower, as true signals may be missed, and no additional information is introduced.



any text

5.4 How fast is TLS?

Very fast! It can search an entire unbinned Kepler K2 lightcurve (90 days, 4000 datapoints) for the best-fit limb-darkened transit model in a few seconds on a typical laptop computer.

In a typical K2 light curve (e.g., EPIC 201367065), TLS (default configuration) performs 3×10^8 light curve evaluations in the χ^2 sense, over 8500 trial periods, each including a transit and the out-of-transit part of the light curve.

A single phase-folded light curve evaluation calculates the squared residuals of the best-fit limb-darkened model light curve to the data points. It pulls the out-of-transit residuals from a cache (if re-usable from previous models) or calculates and caches them. In the end, it returns the χ^2 of this model to the main routine. One such individual model comparison consumes (on average) 230 ns of wall-clock time on one core of an Intel Core i5-6300U at 2.4 GHz.

The average number of in-transit points (in the phase-folded view), i.e. the transit duration in cadences, is 138 (in this example). Considering the out-of-transit points, almost 10^{13} squared-residuals-calculations would be required. Through careful evaluation of which out-of-transit points have previously been calculated and can be re-used, ~96% of these repetitive calculations can be avoided.

In Kepler K2 light curves, on average ~53% of the total compute time is required for phase-folding and sorting. Sorting is set to use numpy's *MergeSort* algorithm which is implemented in the C language. This is slightly faster than the more commonly used *QuickSort*, because phase-folded data is already partially sorted.

But: TLS is written in Python and JIT-compiled with numba. How much faster would a pure C or Fortran implementation be? Not much faster, if faster at all. The innermost numba-loop which calculates the residuals in the χ^2 sense has been measured with a throughput of 12.2 GFLOPs on a single core on an Intel Core i5-6300U at 2.4 GHz. The manufacturer spec-sheet gives a maximum of 16.9 GFLOPs per core at this clock speed, i.e. TLS pulls 72% of the

theoretical maximum. The remaining fraction is very difficult to pull, as it includes a relevant amount of I/O in the form of array shifts. It may be possible to shave off a few percent using hand-optimized assembly, but certainly not more than of order 10%.

5.5 Edge effect jitter correction

TLS fully compensates for the BLS edge effect jitter effect, which we discovered and described in our paper (Hippke & Heller 2019, appendix B). A visualization of this effect on the statistic is shown in an iPython tutorial, using synthetic data.

The original BLS implementation did not account for transit events occurring to be divided between the first and the last bin of the folded light curve. This was noted by Peter R. McCullough in 2002, and an updated version of BLS was made (*ee-bls.f*) to account for this edge effect. The patch is commonly realized by extending the phase array through appending the first bin once again at the end, so that a split transit is stitched together, and present once in full length. The disadvantage of this approach has apparently been ignored: The test statistic is affected by a small amount of additional noise. Depending on the trial period, a transit signal (if present) is sometimes partly located in the first and the second bin. The lower (in-transit) flux values from the first bin are appended at the end of the data, resulting in a change of the ratio between out-of-transit and in-transit flux.

There are phase-folded periods with one, two, or more than two bins which contain the in-transit flux. This causes a variation (over periods) of the summed noise floor, resulting in additional jitter in the test statistic. For typical Kepler light curves, the reduction in detection efficiency is comparable to a reduction in transit depth of ~0.1-1 %. TLS corrects this effect by subtracting the difference of the summed residuals between the patched and the non-patched phased data. In real data, the effect is usually overpowered by noise, and was thus ignored, but is nonetheless present.

5.6 Small period trial ranges

TLS can be parametrized to search over a restricted period range using `period_min` and `period_max`. TLS will then create an optimal period search grid in `[period_min, ..., period_max]`. If the range is very small, only a few periods would be tested. This works in the least-squares (χ^2) sense, i.e. it would detect the period with the smallest residuals for our transit model. With only a few period trials, however, no `power` spectrum can be created (sometimes called "SDE-ogram"). This is because `power` is normalized by its standard deviation, and a standard deviation of just a few (noisy) points is not meaningful. The most common detection criteria is the SDE, often required to be >9 for a signal to be considered interesting. As the SDE is located at the maximum of `power`, it can not be calculated without it. Thus, a small number of period trials are problematic. A large number of periods result in a robuster estimate of the `power` noise floor, and this in a robuster estimate of the height of the peak, the SDE.

TLS solves the issue of very small period ranges by requiring at least 100 trial periods, and extends the period range to its (large) defaults if the grid is too small based on the supplied parameters. Thus, if you set `period_min=365.2` and `period_max=365.3`, TLS will probably default to a larger range (depending on your stellar mass, radius, and oversampling parameter). This is displayed at the start of each TLS run:

```
Searching 18113 data points, 4726 periods from 0.602 to 27.867 days
```

You can use this information to increase your period search range accordingly.

Index