aspros Documentation

Release 0.0.dev11

Brett Morris

Dec 11, 2019

CONTENTS

I Getting Started 3

II aspros Documentation 7

1 Reference/API 11

Python Module Index 23

Index 25

: White dwarf planetesimal hunt with CHEOPS.

Part I

Getting Started

Let's start by importing the necessary packages

```
from aspros import simulate_lc, inject_transits

import astropy.units as u
from astropy.time import Time

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

Simulate a light curve with CHEOPS-like noise and window function:

```
seed = 42  # Makes random number generator reproducible
duration = 24 * u.hour
efficiency = 0.6
clean_lc = simulate_lc(duration, efficiency=efficiency, seed=seed)
```

Inject a transiting object with specified orbital properties:

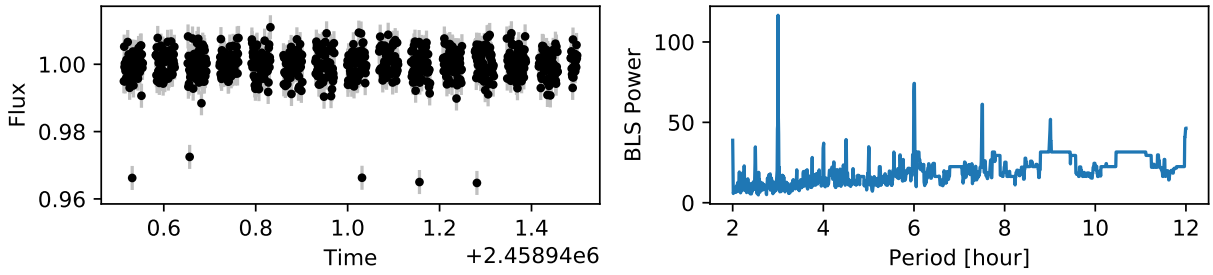
```
period = 3 * u.hour
epoch = Time('2020-04-01') + np.random.rand()*u.day
radius = 1500 * u.km
inc = 90 * u.deg

transit_lc = inject_transits(clean_lc, period, epoch, radius, inc)
```

Construct a Box Least Squares periodogram and inspect it for peaks:

```
periods = np.linspace(2, 12, 1500) * u.hour
results, bests, stats = transit_lc.bls(periods=periods, duration=2*u.min)

fig, ax = plt.subplots(1, 2, figsize=(8, 2))
transit_lc.plot(ax=ax[0])
ax[1].plot(results.period.to(u.hour), results.power)
ax[1].set_xlabel('Period [hour]')
ax[1].set_ylabel('BLS Power')
fig.tight_layout()
plt.show()
```



Part II

aspros Documentation

This is the documentation for aspros.

CHAPTER

ONE

REFERENCE/API

1.1 aspros Package

1.1.1 Functions

bls_peakfinder(results)	Find peaks in a Box Least Squares spectrum.
concatenate_light_curves(light_curve_list[, ...])	Combine multiple transit light curves into one TransitLightCurve object.
concatenate_transit_light_curves(...[, name])	Combine multiple transit light curves into one TransitLightCurve object.
inject_transits(lc, period, epoch, radius, inc)	Inject transits into a light curve.
period_to_a(period)	
simulate_lc(duration[, efficiency, seed, ...])	Simulate a white dwarf light curve observed with CHEOPS.
test(**kwargs)	Run the tests for the package.

bls_peakfinder

aspros.**bls_peakfinder**(*results*)

Find peaks in a Box Least Squares spectrum.

Parameters

results
[BoxLeastSquaresResults] BLS results

Returns

inds
[ndarray] Indices with the top powers, sorted in order of peak height

significance
[float] Ratio of the height of the tallest peak to the height of the second tallest peak

concatenate_light_curves

aspros.**concatenate_light_curves**(*light_curve_list*, *name=None*)

Combine multiple transit light curves into one TransitLightCurve object.

Parameters

light_curve_list
[list] List of TransitLightCurve objects

name
[str] Name of new light curve

Returns

tlc
[TransitLightCurve] Concatenated transit light curves

concatenate_transit_light_curves

aspros.**concatenate_transit_light_curves**(*light_curve_list*, *name=None*)
Combine multiple transit light curves into one TransitLightCurve object.

Parameters

light_curve_list
[list] List of TransitLightCurve objects

name
[str] Name of new light curve

Returns

tlc
[TransitLightCurve] Concatenated transit light curves

inject_transits

aspros.**inject_transits**(*lc*, *period*, *epoch*, *radius*, *inc*, *rstar=<Quantity 6378100. m>*, *a=None*)
Inject transits into a light curve.

Parameters

lc
[LightCurve] Light curve to inject transits into

period
[Quantity] Orbital period

epoch
[Time] Mid-transit time

radius
[Quantity] Planetesimal radius

inc
[Quantity] Orbital inclination

a
[float (optional)] Semi-major axis

rstar
[Quantity (optional)] Stellar radius

Returns

lc_transit
[LightCurve] Copy of the light curve with a transit injected

period_to_a

aspros.**period_to_a**(*period*)

simulate_lc

aspros.**simulate_lc**(*duration*, *efficiency=0.5*, *seed=None*, *noise_scale_factor=2*)
Simulate a white dwarf light curve observed with CHEOPS.

Parameters

duration
[Quantity] Duration of the simulated observations

efficiency
[float (optional)] Efficiency of the observations, defaults to 0.5.

seed
[int (optional)] Random seed

noise_scale_factor
[float (optional)] Scale up the observed noise by a factor of the estimated photon noise, defaults to 2.

Returns

lc
[LightCurve] Light curve of the object

test

aspros.**test**(***kwargs*)
Run the tests for the package.

This method builds arguments for and then calls `pytest.main`.

Parameters

package
[str, optional] The name of a specific package to test, e.g. 'io.fits' or 'utils'. Accepts comma separated string to specify multiple packages. If nothing is specified all default tests are run.

args
[str, optional] Additional arguments to be passed to `pytest.main` in the `args` keyword argument.

docs_path
[str, optional] The path to the documentation .rst files.

open_files
[bool, optional] Fail when any tests leave files open. Off by default, because this adds extra run time to the test suite. Requires the `psutil` package.

parallel
[int or 'auto', optional] When provided, run the tests in parallel on the specified number of CPUs. If parallel is `'auto'`, it will use the all the cores on the machine. Requires the `pytest-xdist` plugin.

pastebin
[('failed', 'all', None), optional] Convenience option for turning on py.test pastebin output. Set to 'failed' to upload info for failed tests, or 'all' to upload info for all tests.

pdb
[bool, optional] Turn on PDB post-mortem analysis for failing tests. Same as specifying `--pdb` in `args`.

pep8
[bool, optional] Turn on PEP8 checking via the pytest-pep8 plugin and disable normal tests. Same as specifying `--pep8 -k pep8` in `args`.

plugins
[list, optional] Plugins to be passed to `pytest.main` in the `plugins` keyword argument.

remote_data
[{'none', 'astropy', 'any'}, optional] Controls whether to run tests marked with @pytest.mark.remote_data. This can be set to run no tests with remote data (`none`), only ones that use data from http://data.astropy.org (`astropy`), or all tests that use remote data (`any`). The default is `none`.

repeat
[`int`, optional] If set, specifies how many times each test should be run. This is useful for diagnosing sporadic failures.

skip_docs
[`bool`, optional] When `True`, skips running the doctests in the .rst files.

test_path
[str, optional] Specify location to test by path. May be a single file or directory. Must be specified absolutely or relative to the calling directory.

verbose
[bool, optional] Convenience option to turn on verbose output from py.test. Passing True is the same as specifying `-v` in `args`.

1.1.2 Classes

`LightCurve`([times, fluxes, errors, ...])	Container object for light curves.
`LooseVersion`([vstring])	Version numbering for anarchists and software realists.
`TransitLightCurve`([times, fluxes, errors, ...])	Container for a single transit light curve.
`UnsupportedPythonError`	

LightCurve

class aspros.**LightCurve**(*times=None*, *fluxes=None*, *errors=None*, *quarters=None*, *name=None*)
Bases: object

Container object for light curves.

Parameters

times
[ndarray] Times in JD

fluxes
[ndarray] Fluxes (normalized or not)

errors
[ndarray] Uncertainties on the fluxes

quarters
[ndarray (optional)] Kepler Quarter for each flux

name
[str] Name this light curve (optional)

Attributes Summary

times_jd	Get the times in this light curve in JD.

Methods Summary

bls(self, periods, duration)	Compute Box Least Squares periodogram
delete_outliers(self)	
from_dir(path[, for_stsp])	Load light curve from numpy save files in dir
from_raw_fits(fits_paths[, name])	Load FITS files downloaded from MAST into the LightCurve object.
get_available_quarters(self)	Get which quarters are available in this LightCurve
get_quarter(self, quarter)	Get a copy of the data from within LightCurve during one Kepler quarter.
get_transit_light_curves(self, params[, plots])	For a light curve with transits only (i.e.
mask_in_transit(self, params[, ...])	Mask out the in-transit light curve based on transit parameters
mask_out_of_transit(self, params[, ...])	Mask out the out-of-transit light curve based on transit parameters
normalize_each_quarter(self[, rename, ...])	Use polynomial fit to each quarter to normalize the data.
phases(self, params)	
plot(self[, transit_params, ax, quarter, ...])	Plot light curve.
save_to(self, path[, overwrite, for_stsp])	Save times, fluxes, errors to new directory dirname in path
split_at_index(self, index)	Split the light curve into two light curves, at index

Attributes Documentation

times_jd
Get the times in this light curve in JD.

Returns

t_jd
[ndarray] Julian dates.

Methods Documentation

bls(*self*, *periods*, *duration*)
Compute Box Least Squares periodogram

Parameters

periods
[ndarray]

duration: '~astropy.units.Quantity'

Returns

results
[BoxLeastSquaresResults]

bests
[list]

stats
[dict]

delete_outliers(*self*)

classmethod from_dir(*path*, *for_stsp=False*)
Load light curve from numpy save files in dir

classmethod from_raw_fits(*fits_paths*, *name=None*)
Load FITS files downloaded from MAST into the LightCurve object.

Parameters

fits_paths
[list] List of paths to FITS files to read in

name
[str (optional)] Name of light curve

Returns

lc
[LightCurve] The light curve for the data in the fits files.

get_available_quarters(*self*)
Get which quarters are available in this LightCurve

Returns

qs
[list] List of unique quarters available.

get_quarter(*self*, *quarter*)
Get a copy of the data from within LightCurve during one Kepler quarter.

Parameters

quarter
[int] Kepler Quarter

Returns

lc
[LightCurve] Light curve from one Kepler Quarter

get_transit_light_curves(*self*, *params*, *plots=False*)
For a light curve with transits only (i.e. like one returned by LightCurve.mask_out_of_transit), split up the transits into their own light curves, return a list of TransitLightCurve objects.

Parameters

params
[TransitParams] Transit light curve parameters

plots
[bool] Make diagnostic plots.

Returns

transit_light_curves
[list] List of TransitLightCurve objects

mask_in_transit(*self*, *params*, *oot_duration_fraction=0.25*)
Mask out the in-transit light curve based on transit parameters

Parameters

params
[TransitParams] Transit light curve parameters. Requires that params.duration is defined.

oot_duration_fraction
[float (optional)] Fluxes from what fraction of a transit duration of the out-of-transit light curve should be included in the mask?

Returns

d
[dict] Inputs for a new LightCurve object with the mask applied.

mask_out_of_transit(*self*, *params*, *oot_duration_fraction=0.25*, *flip=False*)
Mask out the out-of-transit light curve based on transit parameters

Parameters

params
[TransitParams] Transit light curve parameters. Requires that `params.duration` is defined.

oot_duration_fraction
[float (optional)] Fluxes from what fraction of a transit duration of the out-of-transit light curve should be included in the mask?

flip
[bool (optional)] If `True`, mask in-transit rather than out-of-transit.

Returns

d
[dict] Inputs for a new `LightCurve` object with the mask applied.

normalize_each_quarter(*self*, *rename=None*, *polynomial_order=2*, *plots=False*)
Use polynomial fit to each quarter to normalize the data.

Parameters

rename
[str (optional)] New name of the light curve after normalization

polynomial_order
[int (optional)] Order of polynomial to fit to the out-of-transit fluxes. Default is 2.

plots
[bool (optional)] Show diagnostic plots after normalization.

phases(*self*, *params*)

plot(*self*, *transit_params=None*, *ax=None*, *quarter=None*, *show=False*, *phase=False*, ***kwargs*)
Plot light curve.

Parameters

transit_params
[`TransitParams` (optional)] Transit light curve parameters. Required if `phase` is `True`.

ax
[`Axes` (optional)] Axis to make plot on top of

quarter
[float (optional)] Plot only this Kepler quarter

show
[bool] If `True`, call `matplotlib.pyplot.show` after plot is made

phase
[bool] If `True`, map times in JD to orbital phases, which requires that `transit_params` be input also.

plot_kwargs
[dict] Keyword arguments to pass to `matplotlib` calls.

save_to(*self*, *path*, *overwrite=False*, *for_stsp=False*)
Save times, fluxes, errors to new directory `dirname` in `path`

split_at_index(*self*, *index*)
Split the light curve into two light curves, at `index`

TransitLightCurve

class aspros.**TransitLightCurve**(*times=None*, *fluxes=None*, *errors=None*, *quarters=None*, *name=None*)
Bases: `aspros.LightCurve`

Container for a single transit light curve. Subclass of `LightCurve`.

Parameters

times
[`ndarray`] Times in JD

fluxes
[`ndarray`] Fluxes (normalized or not)

errors
[`ndarray`] Uncertainties on the fluxes

quarters
[`ndarray` (optional)] Kepler Quarter for each flux

name
[str] Name this light curve (optional)

Methods Summary

`fit_linear_baseline`(self, params[, cadence, ...])	Find OOT portions of transit light curve using similar method to `LightCurve.mask_out_of_transit`, fit linear baseline to OOT.
`fit_polynomial_baseline`(self, params[, ...])	Find OOT portions of transit light curve using similar method to `LightCurve.mask_out_of_transit`, fit linear baseline to OOT
`from_dir`(path)	Load light curve from numpy save files in `path`
`remove_linear_baseline`(self, params[, ...])	Find OOT portions of transit light curve using similar method to `LightCurve.mask_out_of_transit`, fit linear baseline to OOT, divide whole light curve by that fit.
`remove_polynomial_baseline`(self, params[, ...])	Find OOT portions of transit light curve using similar method to `LightCurve.mask_out_of_transit`, fit polynomial baseline to OOT, divide whole light curve by that fit.
`scale_by_baseline`(self, linear_baseline_params)	

Continued on next page

Table 5 – continued from previous page

subtract_polynomial_baseline(self, params[, ...])	Find OOT portions of transit light curve using similar method to LightCurve.mask_out_of_transit, fit polynomial baseline to OOT, subtract whole light curve by that fit.

Methods Documentation

fit_linear_baseline(*self*, *params*, *cadence=<Quantity 1. min>*, *return_near_transit=False*, *plots=False*)

Find OOT portions of transit light curve using similar method to LightCurve.mask_out_of_transit, fit linear baseline to OOT.

Parameters

params
[TransitParams] Transit light curve parameters. Requires that params.duration is defined.

cadence
[Quantity (optional)] Length of the exposure time for each flux. Default is 1 min.

return_near_transit
[bool (optional)] Return the mask for times in-transit.

Returns

linear_baseline
[numpy.ndarray] Baseline trend of out-of-transit fluxes

near_transit
[numpy.ndarray (optional)] The mask for times in-transit.

fit_polynomial_baseline(*self*, *params*, *order=2*, *cadence=<Quantity 1. min>*, *plots=False*, *mask=None*)

Find OOT portions of transit light curve using similar method to LightCurve.mask_out_of_transit, fit linear baseline to OOT

classmethod from_dir(*path*)

Load light curve from numpy save files in path

remove_linear_baseline(*self*, *params*, *plots=False*, *cadence=<Quantity 1. min>*)

Find OOT portions of transit light curve using similar method to LightCurve.mask_out_of_transit, fit linear baseline to OOT, divide whole light curve by that fit.

Parameters

params
[TransitParams] Transit light curve parameters. Requires that params.duration is defined.

cadence
[Quantity (optional)] Length of the exposure time for each flux. Default is 1 min.

plots
[bool (optional)] Show diagnostic plots.

remove_polynomial_baseline(*self*, *params*, *plots=False*, *order=2*, *cadence=<Quantity 1. min>*)
Find OOT portions of transit light curve using similar method to `LightCurve.mask_out_of_transit`, fit polynomial baseline to OOT, divide whole light curve by that fit.

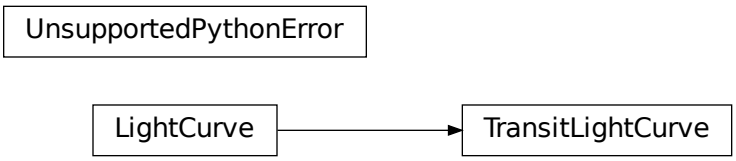
scale_by_baseline(*self*, *linear_baseline_params*)

subtract_polynomial_baseline(*self*, *params*, *plots=False*, *order=2*, *cadence=<Quantity 1. min>*)
Find OOT portions of transit light curve using similar method to `LightCurve.mask_out_of_transit`, fit polynomial baseline to OOT, subtract whole light curve by that fit.

UnsupportedPythonError

exception aspros.**UnsupportedPythonError**

1.1.3 Class Inheritance Diagram



PYTHON MODULE INDEX

a
aspros, 11

INDEX

A

aspros (*module*), 11

B

bls() (*aspros.LightCurve method*), 16
bls_peakfinder() (*in module aspros*), 11

C

concatenate_light_curves() (*in module aspros*), 11
concatenate_transit_light_curves() (*in module aspros*), 12

D

delete_outliers() (*aspros.LightCurve method*), 16

F

fit_linear_baseline() (*aspros.TransitLightCurve method*), 20
fit_polynomial_baseline() (*aspros.TransitLightCurve method*), 20
from_dir() (*aspros.LightCurve class method*), 16
from_dir() (*aspros.TransitLightCurve class method*), 20
from_raw_fits() (*aspros.LightCurve class method*), 16

G

get_available_quarters() (*aspros.LightCurve method*), 16
get_quarter() (*aspros.LightCurve method*), 17
get_transit_light_curves() (*aspros.LightCurve method*), 17

I

inject_transits() (*in module aspros*), 12

L

LightCurve (*class in aspros*), 15

M

mask_in_transit() (*aspros.LightCurve method*), 17
mask_out_of_transit() (*aspros.LightCurve method*), 17

N

normalize_each_quarter() (*aspros.LightCurve method*), 18

P

period_to_a() (*in module aspros*), 13
phases() (*aspros.LightCurve method*), 18
plot() (*aspros.LightCurve method*), 18

R

remove_linear_baseline() (*aspros.TransitLightCurve method*), 20
remove_polynomial_baseline() (*aspros.TransitLightCurve method*), 20

S

save_to() (*aspros.LightCurve method*), 19
scale_by_baseline() (*aspros.TransitLightCurve method*), 21
simulate_lc() (*in module aspros*), 13
split_at_index() (*aspros.LightCurve method*), 19
subtract_polynomial_baseline() (*aspros.TransitLightCurve method*), 21

T

test() (*in module aspros*), 13
times_jd (*aspros.LightCurve attribute*), 16
TransitLightCurve (*class in aspros*), 19

U

UnsupportedPythonError, 21