Draft version March 27, 2018
Typeset using LaTeX **twocolumn** style in AASTeX62

`lightkurve`: a Python open source package for NASA's Kepler, K2, and TESS data analysis

lightkurve contributors[1] and José Vinícius de Miranda Cardoso[2]
(AAS Journals Data Scientists collaboration)

[1] *The galatic startup*
[2] *Bay Area Environmental Research Institute*
Petaluma
California, USA

(Received January 1, 2018; Revised January 7, 2018; Accepted March 27, 2018)

Submitted to ApJ

ABSTRACT

This paper presents an open source package for NASA's Kepler, *K2*, and TESS.

Keywords: editorials, notices — miscellaneous — catalogs — surveys

1. INTRODUCTION

NASA's Kepler, *K2*, and TESS missions have been delivering high-precision time series data for a wide range of stellar types. Cite Borucki et al for Kepler, Howell et al for K2, XXX for TESS.

While these missions have powerful pipelines which deliver high-precision lightcurves for many objects (citation), analyses tailored for specific science cases can often optimize the analysis. For example: 1) a bigger aperture mask has been shown to yield additional planets in Kepler-XXX (cite Isabel), 2) custom tailored PSF photometry has been shown to disentangle the signals of stars in dense clusters (cite Libralato et al), 3) the official pipelines tend to be optimized for signals at the duration of typical planet transits (6 hours) and are not optimized for long-term trends such as those seen in AGN (cite Krista Lynn Smith).

Explain PyKE exists and how it differs. In this paper, we present a new Python package which makes the custom analysis of target easy. Based on AstroPy (cite Astropy).

Explain the structure of the paper and announce that we will show examples of the 3 key features: manipulating lightcurves, creating lightcurves, and removing systematics from lightcurves.

Corresponding author: Geert Barentsen
hello@geert.io

2. PACKAGE OVERVIEW

Maybe include a class diagram here, or a table of the modules or classes.

Maybe include a table or graph listing the key functions of the key classes, e.g. Lightcurve: cdpp, fold, stitch, plot TargetPixelFile: tolightcurve, plot

3. EXAMPLES OF KEY FEATURES

3.1. *Working with lightcurves*

flatten, fold, finding planets, from archive...

3.2. *Making your own lightcurves from target pixel files*

explain TPFs, aperture photometry, and PSF photometry

3.3. *Removing systematics from lightcurves*

We only intend to provide simple tools. Ideally, systematics are removed simultaneously with fitting a model (e.g. Montet and DFM 2015).

4. FUTURE WORK

Explain PSF photometry needs users and data-driven model capability.

We do not intend to implement transit fitting, more advanced detrending, etc. Instead, lightkurve intends to provide the building blocks needed to build or interact with such packages.

We intend to add many tutorials.

Explain how people can contribute.

5. CONCLUSIONS

we will discuss =¿ we have discussed

6. LIGHTCURVE BASICS

6.1. *The `LightCurve` and `KeplerLightCurve` classes*

The `LightCurve` class is a simple cointainer to store `numpy` arrays (hereafter, arrays) related to flux time-domain measurements.

The `LightCurve` object provides methods to store, process, and convert lightcurves. Table 1 contains a description of a subset of the methods.

Table 1. A subset of methods provided by the `LightCurve` class

Method	Short description
`stitch`	appends the attributes `flux`, `time`, and `flux_err` of other given `LighCurve` objects.
`flatten`	applies a Savitzky-Golay filter to capture low frequency flux variations which can be then removed in order to aid transit detection algorithms.
`fold`	folds a lightcurve at a given period and phase.
`bin`	bins a lightcurve using a block mean or median.
`cdpp`	computes the Combined Differential Photometric Precision (CDPP) metric, which is a proxy for the amount of scatter in the lightcurve signal.
`plot`	displays a lightcurve.

A `LightCurve` object can be instantiated by passing a `time` array, a `flux` array, and, optionally, a `flux_err` array which accounts for uncertainties in the `flux` measurements, i.e.,

```
>>> from lightkurve import LightCurve
>>> lc = LightCurve(time, flux)
```

The `KeplerLightCurve`class extends `LightCurve`by adding attributes to store metadata information such as channel number, quality flags, campaign or quarter number, kepler id, etc.

Additionally, `KeplerLightCurve` can be corrected for motion-dependent correlated noise using the `correct` method which will be discussed in Section 4.3.

6.2. *The `KeplerLightCurveFile` class*

The `KeplerLightCurveFile` class defines a structure to deal with lightcurve files from both NASA's Kepler and K2 missions.

To instantiate a `KeplerLightCurveFile` object, it is necessary to pass a `path` which represents the address

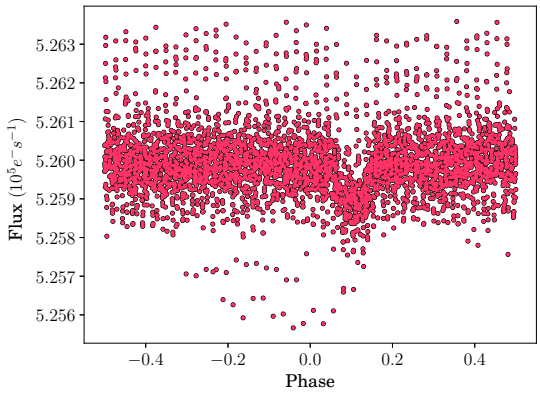

Figure 1. Folded lightcurve of target `KIC011904151` quarter 3, showing the transit signal of Kepler-10b.

(url or local path) of a lightcurve file in the fits (or compressed) format, and a `quality_bitmask` string which specifies quality flags of cadences that should be ignored.

One crucial method of the `KeplerLightCurveFile` class is `get_lightcurve` which returns a `KeplerLightCurve` object with the metadata provided by the corresponding `KeplerLightCurveFile`.

Therefore, one can, for example, perform the following series of operations in order to fold a lightcurve from the MAST archive

```
>>> lc_file = KeplerLightCurveFile("kplr011904151-200935
>>> klc = lc_file.PDCSAP_FLUX.fold(period=0.837495)
>>> klc.plot()
```

7. TARGET PIXEL FILE BASICS

7.1. *The `KeplerTargetPixelFile` class*

A `KeplerTargetPixelFile` object can be instantiated by passing a `path` (URL or local) of a target pixel file. Optionally, the user can elect to throw away frames that contain a specific flag by using the `quality_bitmask` argument.

`KeplerTargetPixelFile` offers a number of methods that range from getting raw aperture photometry lightcurves to data visualization.

For instance, the method `plot` can be used to visualize a given frame, which are depicted in Fig. 2.

```
>>> import numpy as np
>>> from lightkurve import KeplerTargetPixelFile
>>> tpf = KeplerTargetPixelFile("kplr008462852-201107313
>>> tpf.plot()
>>> tpf.plot(aperture_mask=tpf.flux[0] > np.nanmean(tpf.
```

In an image with n pixels, where the flux and the center positions of the i-th pixel are denoted as f_i and

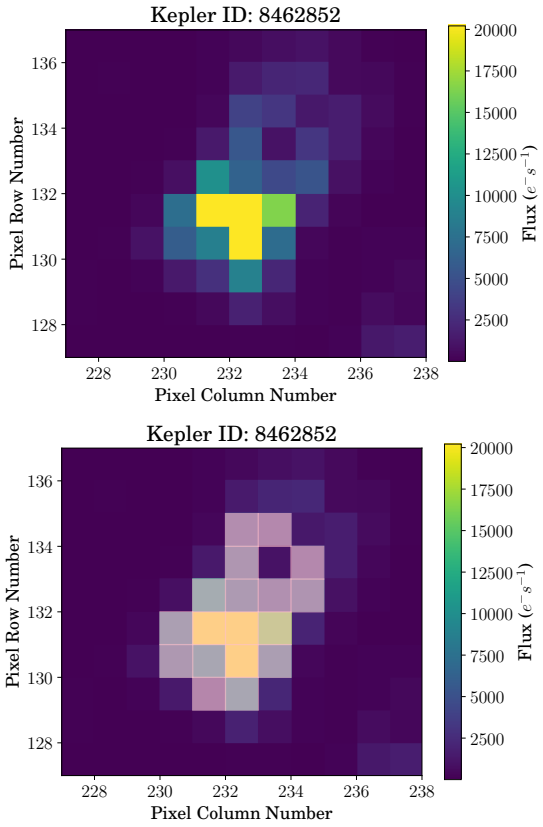

Figure 2. Displaying a given frame of a TPF using `plot`. Optionally, an `aperture_mask` can be passed which is highlighted on the right hand side.

(x_i, y_i), respectively, the centroids may be expressed as

$$x^\star = \frac{\sum_{i=1}^{n} f_i x_i}{\sum_{i=1}^{n} f_i}, \quad y^\star = \frac{\sum_{i=1}^{n} f_i y_i}{\sum_{i=1}^{n} f_i}. \tag{1}$$

In `lightkurve`, the centroids in every cadence can be computed as

```
>>> from lightkurve import KeplerTargetPixelFile
>>> tpf = KeplerTargetPixelFile('ktwo246199087-c12_lpd-targ.fits.gz')
>>> x_star, y_star = tpf.get_centroids()
```

8. TOOLS

8.1. *Cotrending basis vectors*

Cotrending basis vectors (CBVs) can remove global correlated systematics present in a given channel Smith et al. (2012).

Given a set of n CBVs, one is interested in finding a vector of n coefficients $\boldsymbol{\theta} = (\theta_1, \theta_2, ..., \theta_n)$ which minimizes some cost function between the SAP flux and the set of CBVs. The mathematical structure of the cost function is a direct consequence of the statistical assumptions made for data.

For instance, if one assumes that the data comes from an independent and identically distributed (iid) multivariate Gaussian distribution with mean $\sum_{j=1}^{n} \theta_j v_j(t)$ and known variance σ^2, then the cost function can be expressed as follows

$$\mathcal{C}(\boldsymbol{\theta}, f_{SAP}) = \sum_t \left(f_{SAP}(t) - \sum_{j=1}^{n} \theta_j v_j(t) \right)^2, \tag{2}$$

in which f_{SAP} is the SAP flux and v_j is the j-th CBV.

The maximum likelihood estimator for $\boldsymbol{\theta}$, $\boldsymbol{\theta}^\star$ can be expressed as

$$\boldsymbol{\theta}^\star = \arg\min_{\boldsymbol{\theta} \in \Theta} \mathcal{C}(\boldsymbol{\theta}, f_{SAP}). \tag{3}$$

However, Equation (2) is sensitive to outliers Ivezić et al. (2014), therefore, as a default behaviour in `lightkurve`, we use the following cost function

$$\mathcal{C}(\boldsymbol{\theta}, f_SAP) = \sum_t \left| f_{SAP}(t) - \sum_{j=1}^{n} \theta_j v_j(t) \right|. \tag{4}$$

Then, the CBV-corrected flux can be computed as

$$f_{CBV} = f_{SAP} - \sum_{j=1}^{n} \theta_j^\star v_j(t). \tag{5}$$

An example of SAP flux correction for target `KOI 8462852` (Tabby's star) can be written as follows

```
>>> from lightkurve.lightcurve import KeplerCBVCorrector
>>> cbv = KeplerCBVCorrector("kplr008462852-20110731332
>>> cbv_lc = cbv.correct(cbvs=[1,2])
```

Fig 3 illustrates the correction. The pink line has a shift from the green line because in `lightkurve` we do not account the flux lost outside of the aperture mask.

Improperly tuning the number of CBVs can cause over-/under-fitting. One way to identify a reasonable number of CBVs is to perform a grid search as shown in Fig (6).

In the same fashion, we can apply cotrending basis vector correction to $\mathcal{K}2$lightcurves.

Fig. 4 shows the detrended lightcurve after estimating the first nine coefficients for CBV correciton on $\mathcal{K}2$target `EPIC 201543306`. The selection of the number of CBVs is set by inspecting the grid search curve can be set through model comparison heuristics like AIC, BIC, or cross-validation Ivezić et al. (2014).

8.1.1. *Number of CBVs*

The number of CBVs will directly contribute to overfitting effects. One way to identify a reasonable number of CBVs is to perform a grid search as suggested in

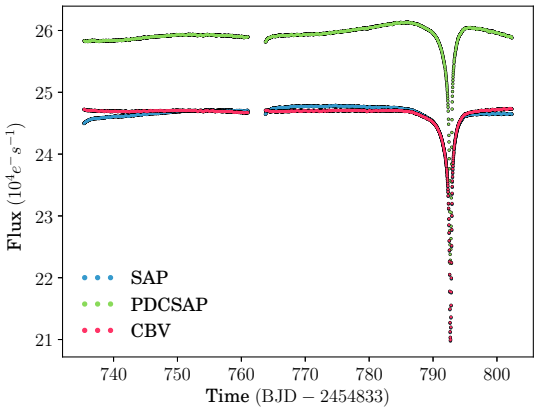

Figure 3. CBV correction applied on `KOI 8462852`

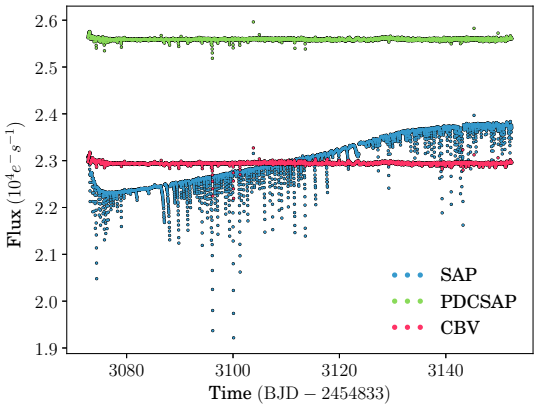

Figure 4. CBV correction applied on `EPIC 201543306`.

Fig (6), which shows the cost function as a function of the number of CBVs. Usually, as the number of CBVs increases, the value of the cost function decreases. And therefore, the user should empirically choose a number of CBVs which does not remove the astrophysical signal of interest [add reference].

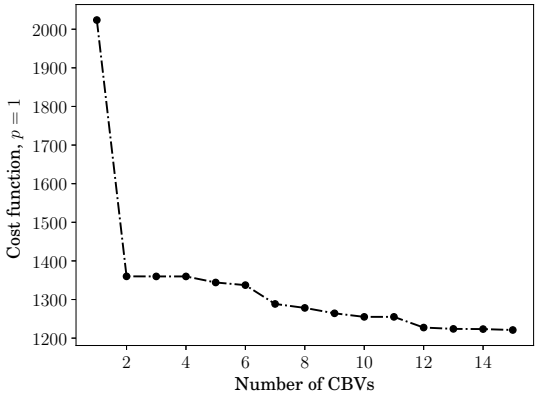

Figure 5. Grid search on the number of CBVs

An objective way of selecting the number of CBVs is to use Bayes' factors [add reference]. In the Bayes' factor setting, the selected number of CBVs is the one that provide the least gain in posterior probability, i.e., for all ordered pairs of CBVs, the Bayes factor selects $n^\star$ number of CBVs as follows

$$n^\star = \arg\min_n \frac{p_{n+1}}{p_n}, \tag{6}$$

in which p_n is the posterior probability evaluated at the Maximum A Posteriori Estimator (MAP) obtained using n CBVs.

A Laplacian prior with zero mean and variance 16 is the default prior density over the CBVs coefficients.

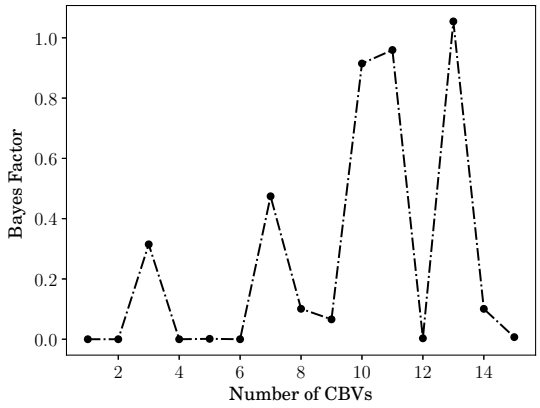

Figure 6. Bayes factors as a function of the number of CBVs for `EPIC 201543306`

8.2. *Point spread function photometry*

`lightkurve` contains routines to perform PSF photometry in TPFs which are implemented in the `psf` module.

Briefly, the PSF photometry problem that `lightkurve` solves can be formulated as follows. Given an image $\boldsymbol{y}$, with n pixels and m stars, and a PSF model $\lambda(\boldsymbol{\theta}) = \sum_{j=1}^{m} \lambda(\theta_j)$, find the best parameter vector (which encodes fluxes and center positions for m stars) $\boldsymbol{\theta}^\star = (\theta_1^\star, \theta_2^\star, ..., \theta_m^\star)$ that minimizes some cost (or loss) function $R(\lambda(\boldsymbol{\theta}), \boldsymbol{y})$ of assigning $\boldsymbol{\theta} = \boldsymbol{\theta}^\star$.

From a probabilistic point of view, one is often interested in minimizing the expected cost with respect to some probability distribution assigned to the data $\boldsymbol{y}$ and to the parameter vector $\boldsymbol{\theta}$, from which the cost function R naturally arises. The default assumption, made in `lightkurve`, on the data is that it follows a Poisson probability distribution, whereas the probability distribution on the parameter vector has to be assigned by the user using the `prior` argument. Using a uniform

prior for $\boldsymbol{\theta}$, the MAP estimator can be written as

$$\boldsymbol{\theta}^{\star}(\boldsymbol{y}) = \arg\min_{\boldsymbol{\theta}\in\Lambda} \sum_{i=1}^{n} \left(\sum_{j=1}^{m} \lambda_i(\boldsymbol{\theta}_j) - y_i \log \sum_{j=1}^{m} \lambda_i(\boldsymbol{\theta}_j) \right), \tag{7}$$

in which Λ is the support of $\boldsymbol{\theta}$.

The example below illustrates PSF photometry on the target `EPIC 246199087` (Trappist-1):

```
>>> from lightkurve import KeplerTargetPixelFile
>>> from lightkurve.psf import PRFPhotometry, SceneModel
>>> from oktopus import UniformPrior
>>> tpf = KeplerTargetPixelFile("ktwo246199087-c12_lpd-targ
>>> prf = tpf.get_prf_model()
>>> prior = UniformPrior(lb=[4e3, 990, 25, 1], ub=[2e4, 996, 30, 2e3])
>>> scene = SceneModel(prf=[prfs])
>>> phot = PRFPhotometry(scene_model=scene, prior=prior)
>>> results = phot.fit(tpf.flux + tpf.flux_bkg)
```

The photometric results are stored in a $c \times 4$ matrix, where c is the number of frames (cadences).

Another important aspect is the PSF model...

8.3. *Motion-dependent Correlated Noise*

Spacecraft-induced correlated noise remains one of the greatest hurdles to analyzing K2 lightcurves. Many algorithms have been developed to mitigate motion-dependent artifacts Vanderburg & Johnson (2014)[CITE K2SC and EVEREST]. In `lightkurve`, we implement an algorithm based off of the self-flat-field (SFF) presented in Vanderburg & Johnson (2014).

Basically, SFF works by decorrelating the simple aperture flux against the information on the spacecraft motion, obtained by computing the arclength using the centroids of the target.

```
from lightkurve import KeplerTargetPixelFile
tpf = KeplerTargetPixelFile("ktwo248667471-c14_lpd-targ.
lc = tpf.to_lightcurve()
centroids = tpf.get_centroids()
lcc = lc.correct(centroids[0], centroids[1])
```

We would like to express our gratitude...

Facilities: Kepler

Software: astropy

APPENDIX

A. APPENDIX INFORMATION

REFERENCES

Ivezić, Ž., Connelly, A. J., VanderPlas, J. T., & Gray, A. 2014, Statistics, Data Mining, and Machine Learning in Astronomy

Smith, J. C., Stumpe, M. C., Van Cleve, J. E., et al. 2012, PASP, 124, 1000, doi: 10.1086/667697

Vanderburg, A., & Johnson, J. A. 2014, Publications of the Astronomical Society of the Pacific, 126, 948, doi: 10.1086/678764