Draft version April 9, 2018
Typeset using LaTeX **twocolumn** style in AASTeX62

`lightkurve`: an open source Python 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 April 9, 2018)

Submitted to ApJ

ABSTRACT

*****CLH: Key messages are: WHY do users need to create their own products? (RB, asteroids, variable apertures, crowding etcetc). WHAT are we providing? (lightkurve, easy Kepler/K2/TESS access). HOW can they use it? (Beautiful examples) *****

- Performing custom photometry allows users to mitigate...
- However, with complex 'big' data from *Kepler*and *K2*...
- We present `lightkurve`, an open source package for analysis of flux time series using Python.
- It is a multipurpose, general tool to make producing time-series photometry from raw pixel data easier and more reproducible between different teams.
- `lightkurve` is designed to interface seemlessly with data from NASA's *Kepler*, *K2* and *TESS* missions.
- Using these tools it is simple to create corrected time-series photometry from raw pixel data from any of these missions.
- Easy to interact with raw data products and produce fantastic light curves
- Open source, modable
- Supremely reproducible.
- Go from raw data to planet in 5 lines of code.
- Excellent learning tool

*****CLH: What plots do we want? These should be our KILLER selling points. All graphs should be easy to understand as standalone figures. These are what will grab people! But let's keep it short and sweet.**

- **Beautiful TPF and masking**
- **Beautiful flattening?**
- **Beautiful motion detrending**

Corresponding author: Geert Barentsen
hello@geert.io

- **Beautiful CBV correction**
- **Beautiful planet finding**

****

1. INTRODUCTION

What is the background—

- Time series photometry is available for a wide variety of astrophysical purposes and in a wide variety of formats.
- This ranges from 30 year long time series of variable stars**?** to short time-series of hours for X-Ray objects **?**
- In particular, NASA's *Kepler*and *K2*missions have provided some of the most precise, long term monitoring of stars to date **?**.
- In the near future the *TESS*mission will delivering high-precision time series data for 90% of the sky, providing light curves for X millions of object **?**.
- *****CLH: What are the challenges we face in time-series astronomy? *****

What tools are currently available?—

- PyKE
- *****CLH: Geert says he can do this para *****
- However, no one package provide a simple, open source framework for manipulating time-series data that is general purpose.

What are we presenting—

- We present `lightkurve`as a general tool to use almost all time-series photometry, with a particular focus on *Kepler*, *K2* and *TESS*.
- While these missions have powerful pipelines which deliver high-precision light curves for many objects (citation), `lightkurve`allows bespoke analyses tailored for specific science cases.
- These might include custom aperture photometry, PSF photometry in crowded field and studies of long period transient events such as supernovae and AGN.
- `lightkurve` is not designed to replace NASA pipelines, but to allow users more flexibility when producing time-series for their unique science cases.
- We have designed `lightkurve`as a tool process this vast wealth of data easily and intuitively with many features and tools to remove reduce the overhead in using this data.
- By using these tools users have the advantage of easy reproducibility. By sharing the same tools and the same short scripts for producing their light curve products different teams will be more able to compare results.

How do you use lightkurve—

- designed to be flexible
- nuts and bolts
- open source
- easy data fetching
- easy api

What is the selling point of lightkurve—

- There are two sides to the `lightkurve`package. Firstly, `lightkurve`can be used as an extraction package for creating time-series photometry from astronomical images such as *Kepler*Target Pixel Files (TPFs) or *TESS*Full Frame Images (FFIs). This includes simple aperture photometry, PSF photometry and centroiding.
- Secondly, `lightkurve`can be used for analysis of time-series photometry. This includes motion detrending, CBV corrections, outlier rejection and period folding.
- Together these two sides can be combined to convert raw data from *Kepler*, *K2* and *TESS* to cleaned light curves of exoplanet candidates, supernovae and extra-galactic objects.
- One flexible system for all optical photometry
- Learning/teaching tool

Future resources?—

- This is `lightkurve`1.0
- Anticipate adding new features
- Easily extendible for users to add in new features
- There are already tutorials, which will be expanded

What's in this paper?—

- In this paper we discuss the basic components of `lightkurve`
- We will show three key components of analysis with `lightkurve`; manipulating lightcurves, creating lightcurves, and removing systematics from lightcurves.
- `lightkurve`has a full compliment of tutorials
- More details can be found in our documentation at link.

2. PACKAGE OVERVIEW

*****CLH: Ze can you do a beautiful diagram here? *****

2.1. *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.

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 **??**.

2.2. *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 (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()
```

2.3. *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. 1.

```
>>> 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 (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-t
>>> x_star, y_star = tpf.get_centroids()
```

*****CLH: This table I believe needs a little updating? There can also be a second table for TPF *****

2.4. *PyKE Tools*

- previously there was pyke, don't spend too long on this
- pyke is still available
- `lightkurve`uses only python
- new Python package which makes the custom analysis of target easy. Based on AstroPy (cite Astropy).

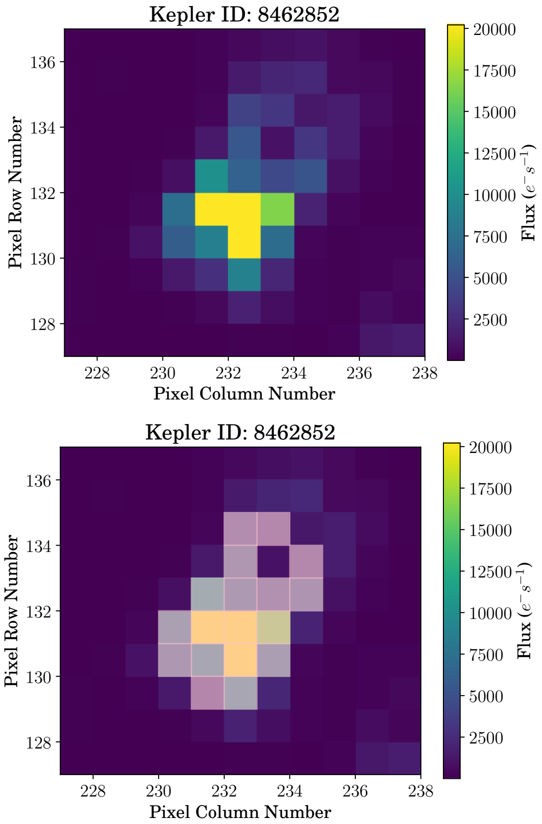

Figure 1. Displaying a given frame of a TPF using `plot`. Optionally, an `aperture_mask` can be passed which is highlighted on the right hand side.

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.

3. COMMON USE CASES

3.1. *Creating Custom Light Curves*

3.1.1. *Simple Aperture Photometry (SAP)*

- What is SAP?
- Why do SAP?s
- How do you do SAP?

3.1.2. *Point Spread Function (PSF) Photometry*

Point Spread Function (PSF) photometry is the de facto technique to process crowded-field images **??**. In context of Kepler and K2 missions, Libralato *et al* **?** have shown...

See a detailed explaination of PSF photometry in B The underlying principle of PSF photometry consists in modelling a given crowded image as a linear combination of individual PSFs and possibly a background model. On the PSF model itself, it is commonly assumed that the flux at an arbitrary pixel position increases linearly with the integrated flux **??**.

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

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

******CLH: If you want to include oktopus in this snippet you have to explain what it is to the reader:) ******

```
>>> from lightkurve import KeplerTargetPixelFile
>>> from lightkurve.psf import PRFPhotometry, Sc
>>> from oktopus import UniformPrior
>>> tpf = KeplerTargetPixelFile("ktwo246199087-c
>>> prf = tpf.get_prf_model()
>>> prior = UniformPrior(lb=[4e3, 990, 25, 1], u
>>> scene = SceneModel(prf=[prfs])
>>> phot = PRFPhotometry(scene_model=scene, pric
>>> 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).

3.2. *Correcting Common Systematics*

We provide tools to correct for two systematics that are common between targets on the same channel. We provide corrections using *Cotrending Basis Vectors* (CBVs) which mitigate systematics due to e.g. spacecraft heating. (See Appendix **??** for a detailed explanation of CBVs) We also provide a simple implementation of the *Self Flat Fielding* (SFF) method to correct for spacecraft motion. (See Appendix C for a detailed explanation of SFF)

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

3.2.1. *Correcting Spacecraft Motion using Lightkurve*

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).

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. (See Appendix C for a detailed explanation of SFF)

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

3.2.2. *Correcting Common Systematics with CBVs using Lightkurve*

*****CLH: Simple paragraph describing CBVs ***** Cotrending basis vectors (CBVs) can remove global correlated systematics present in a given channel Smith et al. (2012).

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

```
>>> from lightkurve.lightcurve import KeplerCBV(
>>> cbv = KeplerCBVCorrector("kplr008462852-201
>>> cbv_lc = cbv.correct(cbvs=[1,2])
```

Fig 2 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 (**??**). 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).

In the same fashion, we can apply cotrending basis vector correction to *K2*lightcurves.

3.3. *Recovering a planet signal*

3.4. *Light curves from large images*

4. EXAMPLE CODE SNIPPETS

4.1. *Quickly recover a planet*

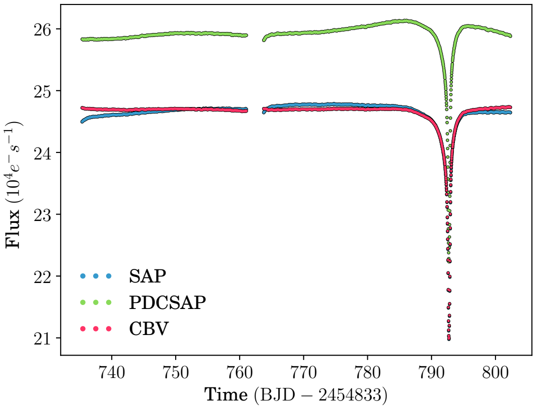

Figure 2. CBV correction applied on `KOI 8462852` *****CLH: This is great, really valuable plot. I would just change to have two lines with labels corrected and uncorrected. *****

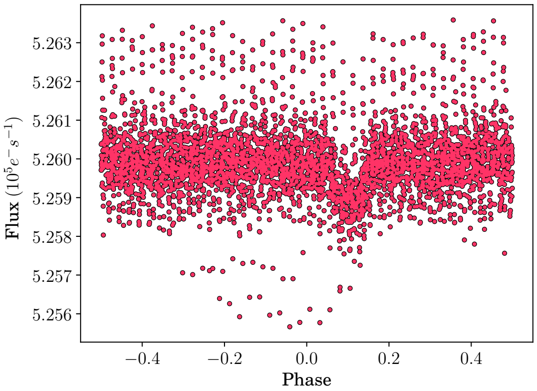

Figure 3. Folded lightcurve of target `KIC011904151` quarter 3, showing the transit signal of Kepler-10b.

4.2. *Correct a supernova light curve*

4.3. *Background Subtraction for AGN*

5. 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.

6. CONCLUSIONS

we will discuss =¿ we have discussed

APPENDIX

A. COTRENDING BASIS VECTORS

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

$$\mathcal{C}(\boldsymbol{\theta}, f_{SAP}) = \sum_t \left(f_{SAP}(t) - \sum_{j=1}^{n} \theta_j v_j(t) \right)^2 , \tag{A1}$$

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

$$\boldsymbol{\theta}^\star = \arg\min_{\boldsymbol{\theta} \in \Theta} \mathcal{C}(\boldsymbol{\theta}, f_{SAP}). \tag{A2}$$

However, Equation (A1) is sensitive to outliers Ivezić et al. (2014), therefore, as a default behaviour in `lightkurve`, we use the following cost function

$$\mathcal{C}(\boldsymbol{\theta}, f_S AP) = \sum_t \left| f_{SAP}(t) - \sum_{j=1}^{n} \theta_j v_j(t) \right| . \tag{A3}$$

Then, the CBV-corrected flux can be computed as

$$f_{CBV} = f_{SAP} - \sum_{j=1}^{n} \theta_j^\star v_j(t). \tag{A4}$$

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 Fig (**??**), 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].

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

$$n^\star = \arg\min_n \frac{p_{n+1}}{p_n}, \tag{A5}$$

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.

B. POINT SPREAD FUNCTION PHOTOMETRY

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

$$\boldsymbol{\theta}^{\star}(\boldsymbol{y}) = \arg\min_{\boldsymbol{\theta}\in\Lambda} \sum_{i=1}^{n}\left(\sum_{j=1}^{m}\lambda_i(\boldsymbol{\theta}_j) - y_i \log \sum_{j=1}^{m}\lambda_i(\boldsymbol{\theta}_j)\right), \tag{B6}$$

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

Another important aspect is the PSF model...

C. MOTION-DEPENDENT CORRELATED NOISE

*****CLH: Quick discussion of the SFF algorithm *****

We would like to express our gratitude... Funding sources

Facilities: Kepler

Software: astropy

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