Linearized Field Deblending: Point-spread Function Photometry for Impatient Astronomers

Christina Hedges\textsuperscript{1,2}, Rodrigo Luger\textsuperscript{3}, Jorge Martinez-Palomera\textsuperscript{1,2}, Jessie Dotson\textsuperscript{2}, and Geert Barentsen\textsuperscript{1,2}

\textsuperscript{1} Bay Area Environmental Research Institute, P.O. Box 25, Moffett Field, CA 94035, USA; christina.l.hedges@nasa.gov
\textsuperscript{2} NASA Ames Research Center, Moffett Field, CA, USA
\textsuperscript{3} Center for Computational Astrophysics, Flatiron Institute, New York, NY, USA

Received 2021 May 6; revised 2021 June 1; accepted 2021 June 2; published 2021 August 17

Abstract

NASA’s Kepler, K2, and TESS missions employ simple aperture photometry to derive time-series photometry, where an aperture is estimated for each star, and pixels containing each star are summed to create a single light curve. This method is simple, but in crowded fields, the derived time series can be highly contaminated. The alternate method of fitting a point-spread function (PSF) to the data is able to account for crowding but is computationally expensive. In this paper, we present a new approach to extracting photometry from these time-series missions that fits the PSF directly but makes simplifying assumptions in order to greatly reduce the computation expense. Our method fixes the scene of the field in each image, estimates the PSF shape of the instrument with a linear model, and allows only source flux and position to vary. We demonstrate that our method is able to separate the photometry from blended targets in the Kepler data set that are separated by less than a pixel. Our method is fast to compute and fully accounts for uncertainties from degeneracies due to crowding. We name the method described in this work linearized field deblending photometry. We demonstrate our method on the false-positive Kepler target KOI-608. We are able to separate the photometry of the two sources in the data and demonstrate that the contaminating transiting signal is consistent with a small, substellar companion with a radius of 2.67 \( R_J \) \( (0.27 R_{\odot}) \). Our method is equally applicable to extracting photometry from NASA’s TESS mission.

Unified Astronomy Thesaurus concepts: Time series analysis (1916); Photometry (1234); Computational methods (1965); Astronomy data analysis (1858)

1. Introduction

NASA’s Kepler mission provided photometry of over 400,000 stars (Borucki et al. 2010), revolutionizing the field of study of transiting exoplanets. Kepler relied on the Kepler pipeline to extract the photometry of these sources (Jenkins et al. 2010), which provided high-quality data products including (1) target pixel files (TPFs), (2) simple aperture photometry (SAP) light curves, and (3) presearch data conditioning SAP (PDCSAP) light curves (Stumpe et al. 2012). The TPFs consist of stacks of exposures containing pixel cutouts around a single star, usually covering \( \approx 5^2 \) pixels on the detector. The SAP light curves consist of time-series photometry using pipeline-generated apertures. The PDCSAP light curves are the SAP light curves corrected for instrument systematics using on-trending basis vectors (CBVs; see Smith et al. 2012). NASA’s TESS mission recently completed its prime mission, observing over 50 million stars (Ricker et al. 2015; Barclay et al. 2018), and uses similar methods to the Kepler pipeline to produce light curves.

In this paper, we present a new approach to extract photometry from data sets similar to those from the Kepler mission, which we name linearized field deblending (LFD) photometry. Rather than using SAP methods, we develop a method of modeling the astrophysical scene, where we create a simple, fast-to-evaluate model for the pixel response function (PRF) of each source on the detector and rely on the Gaia catalog to correctly place each source in the scene, fitting only for the flux in each source. We then simultaneously fit all sources in the image, which enables us to accurately account for crowding and contamination in the time series of each source. Our method employs a linear model and is able to accurately “deblend” crowded fields, fully accounting for uncertainties due to the crowding in the scene, so we name our method LFD. The LFD photometry is fast (in this work, we present a demonstration fitting 576 sources simultaneously in under 4 minutes on a personal laptop), requires few free parameters, and provides reasonable photometric errors, fully accounting for uncertainties due to crowding. As such, we suggest that LFD photometry is ideal for the impatient time-domain astronomer.

In Section 2 of this paper, we discuss the different approaches to photometry in the literature and the benefits of scene modeling. In Section 3, we discuss key decisions we have made to make our method more tractable. In Section 4, we present our model in the context of a demonstration target: KOI-608. In Section 5, we discuss the limitations of our model and the potential applications.

2. Background: Photometry

Performing high-precision photometry for an astronomical source from instruments like Kepler requires identifying which pixels contain signal for the source and combining them to maximize the signal-to-noise ratio (S/N) and precision of the photometry. The most common general-purpose methods are aperture and point-spread function (PSF) photometry. Difference imaging is an alternative method used when the primary goal is to identify variable sources. The Kepler mission itself employed aperture photometry, which is also favored by ground-based observatories. In this section, we briefly discuss common photometry methods.

2.1. Aperture Photometry

Aperture photometry is a process of selecting pixels to be summed in a stack of images in order to create time-series...
photometry; SAP sums all the pixels assigned to each source to create a single photometric time series. This method benefits from being simple and fast to apply to targets. However, in cases of severe crowding, it can be difficult to assign apertures, and a trade must be made between completeness of the source flux and contamination from neighbors.

The Kepler pipeline utilized SAP with excellent results (Morris et al. 2020). Two methods were used to identify candidate pixels for each source, one based on a model of the image and one based on the raw pixel data. The final aperture was chosen to maximize the S/N and minimize the combined differential photometric precision (CDPP) for each source (Smith et al. 2020). Crowding and contamination metrics were also provided.

Other methods use a modified version of this approach by assigning weights to each pixel in the aperture. For example, the photutils package enables users to build apertures from “partial” pixels by assigning pixel-level weights, enabling users to build, for example, circular apertures. Another example of a weighted aperture method is the method introduced by Naylor (1998), where the pixel weights are computed from a PSF model profile fitted to stars in the vicinity and scaled by the variance of each pixel. This method offers robust error estimations, and, assuming that the PSF profile does not vary significantly across subregions of the image, computing times can be tuned by fitting a lower number of PSF models in an image (Förster et al. 2016).

Tools can be used to create aperture photometry; for example, photutils provides general Python libraries for aperture photometry. The lightkurve package (Lightkurve Collaboration et al. 2018) can be used to create SAP from Kepler, K2, and TESS. The eleanor package (Feinstein et al. 2019) performs both SAP and weighted aperture photometry on sources observed in TESS full-frame images using a set of predefined simple and weighted apertures. Whether simple or weighted, aperture photometry usually provides a fast way to create time-series photometry once pixels have been assigned to apertures. Usually some component of scene modeling or point finding is required to assign pixels to apertures.

2.2. PSF Photometry

In PSF photometry, an estimate of the PSF shape is used, and both the flux and the position of the source are fit, either simultaneously or separately. Anderson & King (2000, 2006) demonstrated how to build a model for the “effective” PSF (ePSF) of an instrument and fit to retrieve the flux and position of sources for Hubble Space Telescope instruments. This ePSF is similar to the Kepler PRF presented in Bryson et al. (2010), which they defined as the optical PSF of the Kepler instrument, having been convolved with the instrument systematics, pixel sensitivities, and any intracadence motion and recorded on the detector. Figure 1 shows an example of the Kepler PRF created using dithered data. Whether using some estimate of the instrument PSF or a data-driven model of the ePSF/PRF, this method usually fits for both position and flux.

Several works have used PSF or PRF photometry on Kepler data. For example, Nardiello et al. (2015) derived PSF models, fit the flux and positions of stars neighboring their target star, removed these neighbors, and calculated the target photometry using both PSF and aperture photometry. Similarly, Libralato et al. (2016) estimated the PRF of the Kepler instrument and performed the same fit. Open-source tools such as photutils4 (Bradley et al. 2020) have enabled users to perform PSF photometry and reimplement algorithms from DAOPHOT (Stetson 1987) in Python. Because PSF photometry usually fits both position and flux and fits each star independently, it can sometimes be difficult to constrain a model in practice. Owing to these many parameters, PSF photometry is also much slower to compute than SAP.

Some works implement a simplified, less flexible version of PSF photometry. For example, Feinstein et al. (2019) implemented a simplified PSF photometry option on TESS data (which is similar in format to Kepler data). The locations of the stars are assumed to be known a priori. The PSF is assumed to be a 2D Gaussian. The two widths and orientation of the PSF are fit from an ensemble of sources. The height of the PSF is allowed to vary for each source. While computationally less intensive than other PSF implementations, this approach is not well suited to crowded regions due to the assumption of a Gaussian PSF and a static scene.

2.3. Difference Imaging

Difference imaging, or image subtraction, consists of aligning the images in a stack of observations to a reference frame, accounting for differences in seeing or PSF shape through kernel convolution, and subtracting images to find the difference. Difference imaging was first discussed in this way in Tomaney & Crotts (1996), and kernels were later improved in Alard & Lupton (1998) and Alard (2000). Difference imaging has been applied to the Kepler/K2 mission to extract light curves from extremely crowded fields such as the galactic center in Wang et al. (2016), who developed a causal pixel modeling approach to difference imaging and Zhu et al. (2017) and to open clusters in Soares-Furtado et al. (2017). This approach can be useful when looking for transient events, which appear as high-S/N events in difference imaging, where

---

4 See https://photutils.readthedocs.io/en/stable/psf.html.
quiescent phenomena do not. However, difference imaging usually provides only a difference in flux and therefore requires external information to retrieve the flux of a source in absolute terms. Additionally, difference imaging can be highly sensitive to the precision of the image registration.

2.4. LFD Photometry

In this work, we approach the problem by attempting to recreate the entire image of all stars at the pixel level simultaneously. In the PSF/PRF photometry approach, catalogs are often used to build the scene or a starting point, but in most cases, this input position is allowed to vary, and stars are fit either individually or in small groups. In LFD photometry, we instead (1) fix the locations of each source and (2) predict the flux in each source.

In the method we discuss in this work, a preexisting catalog is used that is more accurate than the pixel scale of the detector. We use Gaia EDR3, which provides extremely precise and complete locations for targets in Kepler’s sensitivity range. We build a PRF model for all sources, evaluate this model at the location of each source given by Gaia, and fit for the flux of each source simultaneously in an individual image. This process is repeated on a stack of images in order to obtain the time series. To account for the common motion of the “scene” (e.g., due to velocity aberration), we may allow the model to drift over time (see Section 4.5). We use Gaia G-band photometry as an estimate for the expected flux observed with Kepler, which provides the best match to the Kepler bandpass of any similarly complete survey. We assume that Gaia is complete to a Kepler magnitude of 18, and that any stars unresolved by Gaia due to crowding are indistinguishable from single stars in Kepler.

The LFD photometry is less flexible than PSF/PRF modeling, as we are not able to vary the positions of the sources independently. This can be seen as a strength of the method, as this prevents degeneracies if, for example, in the case of Kepler, a source is moving out of a TPF. In such a case, it can be difficult to distinguish between a source moving and a source changing flux. In LFD photometry, no source is allowed to move independently from the other sources, so we are able to break this degeneracy. Since all sources are fit simultaneously, we can retrieve the correct errors, marginalized over the uncertainty due to crowding. This approach is also faster than PSF/PRF photometry for a large number of sources, since there are fewer free parameters (i.e., no position parameters).

We propose that modeling the whole astronomical scene using our method is a highly useful tool in the era of Kepler and TESS.

In this work, we will demonstrate how to fit the PRF shape of a given stack of well-aligned images to use in LFD photometry (see Section 4). In our specific case of Kepler data, it would also be possible to utilize the measured Kepler PRF from Bryson et al. (2010), which was built from all sources on a single Kepler channel. In our method, we are essentially rederiving a slightly noisier estimate of the PRF by creating a model of it using fewer sources and allowing no flexibility in the shape as a function of position on the detector. We choose to do this so that we are able to (1) obtain the best-fitting local PRF; (2) obtain the polar coordinate model for the PRF, which is fast to compute and evaluate; and (3) demonstrate the general approach here, which is equally applicable to Kepler, K2, TESS, and beyond. In theory, it is possible to fit existing models in the ways described in this work (e.g., from Bryson et al. 2010, or from optical simulations) and use those to fit source weights, rather than building a data-driven model.

2.5. Detrending

Many light curves, from both the Kepler and TESS missions and from ground-based observatories, use postprocessing to correct instrumental effects. For example, the CBV method corrects for motion in the Kepler data due to velocity aberration and focus change due to spacecraft heating (Smith et al. 2012). Methods such as the self-flat-fielding (SFF) technique (Vanderburg & Johnson 2014) and pixel-level decorrelation (Luger et al. 2016) have been used to postprocess SAP light curves from the K2 mission (the second phase of the Kepler mission) to remove the high-frequency roll motion of the telescope. Wang et al. (2016) used causal pixel modeling in combination with a difference imaging approach to detrend the K2 spacecraft motion in the Campaign 9 Microlensing campaign. Aigrain et al. (2015) employed a Gaussian process to model and detrend the K2 roll motion in light curves derived using a circular aperture. These varied postprocessing approaches can improve the photometric precision of light curves generated using apertures, PSFs, or difference imaging. In this work, we demonstrate how to extract photometry but attempt no further postprocessing. However, the CBV detrending and SFF techniques to postprocess SAP light curves are just as applicable to light curves derived using LFD photometry and will similarly improve photometric precision.

3. Simplifying Assumptions Employed in LFD

In order to compute our model of the image quickly, we require a simple spatial model of the PRF that is efficient to fit and evaluate. To do this, we require simplifying assumptions. In LFD photometry, we undertake two key simplifying assumptions, described below.

In our method, we design a simple linear model of low-order, piecewise polynomials. The PSFs (and thus PRFs) tend to have a high dynamic range (tall peaks and wide, shallow wings), which can make fitting a linear model difficult. The PSFs are also narrow, usually covering just a few pixels. Other tools, such as photutils, create a fine grid of subpixels and then could many PSFs from the data on this fine (oversampled) grid. This finer grid can then be fit with some model or simply interpolated. However, fitting a linear model to this gridded data in order to evaluate it anywhere on the detector can still be difficult and expensive. Instead, we take the following approaches to simplify the problem.

3.1. Fitting in Polar Coordinates

The PSFs (and so PRFs) usually have some degree of natural radial symmetry, even in cases where they are elongated in one direction. Rather than fitting in Cartesian coordinates, where the PSF changes rapidly on small spatial scales, fitting in polar coordinates allows us to benefit from this radial symmetry and fit a smooth model. In Cartesian space, we would fit the flux of a PSF as a function of two parameters,

$$\delta x = x - x_0$$

and

$$\delta y = y - y_0.$$
where \( x \) and \( y \) denote the pixel position in each dimension, \( x_0 \) and \( y_0 \) denote the center of a given source, and \( \delta x \) and \( \delta y \) indicate the separation from the source in the \( x \) and \( y \) dimensions. Instead, in this work, we fit the flux as a function of

\[
\begin{align*}
  r &= (\delta x^2 + \delta y^2)^{\frac{1}{2}}, \\
  \phi &= \arctan(2(\delta y, \delta x)),
\end{align*}
\]

where \( r \) is the radial distance from the PSF center and \( \phi \) is the azimuth angle around the PSF. The top panels of Figure 2 show an example of data from the Kepler mission in each of these coordinate systems, demonstrating that the flux data from the telescope vary more smoothly in \( \phi \) and \( r \) than in \( \delta x \) and \( \delta y \).

### 3.1.1. Fit in Log of Flux

The PSFs have a large dynamic range and often have wings that are orders of magnitude fainter than their peaks. In the case of Kepler, channels at the edge of the focal plane have PSFs with highly extended wings (see the Kepler Instrument Handbook, KIH; Van Cleve & Caldwell 2016). As shown in Figures 1 and 2, the wings of the PRF are up to 2 orders of magnitude fainter than the center of the PRF. The flux varies steeply as a function of radial distance from the center of the PSF, following a power law (approximately \( 1/r^2 \)). As a simplifying assumption, we fit in the log of the flux, whereby we can fit this power law with a simple polynomial, meaning that we can create a simple linear model. We additionally benefit from log space enforcing that the model must always be positive.

### 3.2. Basis Splines

In this work, we use basis splines to model the PRF. Basis splines are nonparametric and model data as simple piecewise polynomials, which makes our PRF model linear and therefore fast to optimize and evaluate. A basis spline is defined by a number of “knots,” which are points where the piecewise polynomials meet. A basis spline is defined such that, if we specify a spline with a degree of 3, the first two derivatives of the function at these knots are continuous; i.e., at these knots, the piecewise polynomials must vary smoothly. We can create a design matrix of basis spline components that is prescribed by a number of knots in the \( \phi \) and \( r \) dimensions that controls the level of detail we are able to capture in our model. Increasing the number of knots increases the model complexity and computation time. We describe a basis spline in detail in Appendix C.

As we will use polar coordinates when fitting the PRF model, our basis spline must “wrap” in the \( \phi \) dimension (i.e., the value at 0° must be identical to the value at 360°). We include a simple implementation of how to create such a basis spline using Python in Appendix D of this work.

To create a basis spline model, we create a design matrix made up of vectors containing each part of the piecewise polynomial for a given variable. When this matrix is dotted with a vector of weights, we retrieve a smoothly varying model. See Appendix B for a general example of how to fit to find the best-fitting weights for the design matrix using linear least squares. In the sections below, we use the equations in Appendix B to fit for weights using several different design matrices.

### 4. Implementation

Our approach to modeling a stack of astronomical images from Kepler follows these stages.

1. Obtain the source positions and average \( G \)-band brightness in the scene using Gaia.
2. Build an estimate of the edge of the PRF in order to identify pixels that have a contribution from a single source (“uncontaminated” pixels).
3. Build a linear model for the PRF shape using uncontaminated pixels, which can be evaluated quickly at a given separation from a source center. Iteratively fit and mask out data to build an accurate model of the PRF shape.
4. Extract pixel time series of uncontaminated pixels, mean normalize, and model the change in flux due to common motion in the scene as a function of time.
5. Model the image stack as a combination of the PRF shape model and common motion model, where the weight for each source as a function of time gives the flux time series of the source.

These steps are described in detail below. In Section 4.7, we conclude with a diagram of the steps involved in our method.

We make the following implicit assumptions.

1. Our catalog (in this case, Gaia EDR3) is accurate, and we can ignore any errors in the location of each source in the scene. Extending this, we assume that the World Coordinate System solution for the image stack is accurate.
2. There is no distortion of the image due to, e.g., the optics of the telescope, such that PRF shapes do not vary greatly over the spatial scale of the images.
3. The motion in the scene is small, less than the width of the PSF, and smooth over time.
4. The motion in the scene is the same across all sources.
5. The intra- and interpixel sensitivity changes are small and average out across many sources.

We will discuss the limitations of these assumptions in the context of Kepler, K2, and TESS in Section 5. A table with descriptions of all of the variables used in this section is given in Appendix A in Table 2.

In this section, we discuss our implementation of the steps above in the context of Kepler object of interest KOI-608. This object is a false-positive planet candidate from the Kepler mission that shows a significant transiting signal. The Kepler data of KOI-608 contain two sources separated by just over a pixel on the detector. The first source (Gaia EDR3 2073618323728332544) has a Gaia $g$ magnitude of 16.855 and hosts a transiting signal. The second source (Gaia EDR3 2073618220649116160) has a Gaia $g$ magnitude of 14.723 and was the target of the Kepler survey. Owing to the detection of a significant centroid shift, indicating that the signal originated from a faint background source, KOI-608 was discounted as a single source, which are the pixels we can use to inform our PSF model. In this work, we use KOI-608 as a demonstration, our signal in KOI-608 is still consistent with being a small, substellar companion. While we use KOI-608 as a demonstration, our approach will work for any Kepler target. We summarize the steps in this section in a flowchart in Section 4.7.

4.1. Data Preparation

We obtain the Kepler pipeline-processed TPF data from the Mikulski Archive for Space Telescopes (MAST)\(^5\) for KOI-608 and download TPFs of 200 neighboring sources. We find in this case that 200 TPFs provide a reasonable balance between having enough sources to constrain the model and being close enough on the detector to have a similar PSF shape. In this proof of concept, we use only a single Kepler quarter. For each TPF, we query Gaia EDR3 (Gaia Collaboration et al. 2021) to obtain a catalog of all sources that were observed in the TPFs, down to a limiting magnitude of 18 in Gaia $g$ magnitude. We will denote that TPFs have $n$ pixels and $m$ cadences and that $l$ unique sources are observed. In our example, $n = 6978$ pixels, $m = 4442$ cadences, and $l = 576$ sources. We will label the image stack data $f$, which is a matrix with shape $(n, m)$.

In this work, we have matrices of quantities (for example, Cartesian separation from each source), which are 2D and have dimensions $(l, n)$ (number of sources by number of pixels). To use these quantities in our framework, we often “unwrap” these 2D matrices into 1D vectors. We build vectors containing the Cartesian separation from each source, $\delta x$ and $\delta y$, which have length $l \times n$ (number of sources multiplied by number of pixels). In this work, we use the notation vec(...) to denote “unwrapping” a matrix and mat(...) to denote the inverse (e.g., converting a vector of length $l \times n$ to a 2D matrix of shape $(l, n)$). Using Equations (3) and (4), we create matrices $\phi$ and $r$ with length $l \times n$.

4.2. Estimating the Edge of the PRF as a Function of Flux

In order to be efficient in future steps, we first estimate a circular aperture for each source. This initial aperture enables us to identify pixels that are likely to contain only flux from a single source, which are the pixels we can use to inform our PRF model. A reasonable circular aperture is neither too small (which omits flux at the edge of the PRF) nor too big (which causes us to overestimate crowding). To find a reasonable circular aperture as a function of flux, we build a basic model of the PRF shape. We then use this model to find the radius at which the recorded counts from a source would reach the noise limit of Kepler. We define this as the edge of the PRF.

To find the edge of the PRF, we fit the time-averaged (mean) flux of all frames, which we will denote as $\hat{f}$ and is a vector with length $n$. We model $\hat{f}$ as a simple polynomial in radial separation from each source $r$ (which is a vector with length $l \times n$) and our a priori estimate of the source flux. In this initial step, we will use the Gaia mean flux (phot_g_mean_flux in the Gaia EDR3 catalog) as our a priori flux estimate of each source. We will denote this estimate as $g$, which is a vector with length $l \times n$, where the value of $g$ is the same for all pixels for a given source.

We build a design matrix for this process using

\[
X' = \text{vec}\left(\begin{bmatrix} 1 & g & [1 \ r \ r^2] \end{bmatrix}\right),
\]

(5)

\[
X' = \begin{bmatrix} 1 & g & g^2 & r & gr & g^2r & r^2 & gr^2 & g^2r^2 \end{bmatrix}.
\]

(6)

where $X'$ is the design matrix for this initial step; vec(x) denotes the vectorization operation, which unrolls a matrix into a vector; and $X'$ is a 2D matrix and has shape $(l \times n, 9)$. Using the equations given in Appendix B, we then fit to find the best-fitting model flux for every pixel, which we will denote as $\hat{f}_0$ and is a vector of length $l \times n$. The hat symbol in this work denotes our best mean estimate of a quantity,

\[
\hat{f}_0 = X' \cdot \hat{w}_0,
\]

(7)

where the subscript zero denotes the model flux for the first-stage simple circular aperture and $\hat{w}_0$ is a vector of the best-fit coefficients for the circular aperture model with length 9. Equation (7) models the data as a simple circular PRF with a dependence only on the source flux and radius from the source. We then consider the edge of the PRF to be the radius at which this model is greater than some threshold. We find empirically that a value of $50e^{-s^{-1}}$ balances reasonably sized apertures that include the majority of the wings of the PSF. This allows us to place simple apertures around every source and identify “uncontaminated pixels” (pixels with contributions from a single source). Using this model, we identify 3418 pixels that are uncontaminated in our demonstration example.

4.3. Building a Linear Model of the PSF Shape

We use basis splines in polar coordinates to build a model for the shape of the PRF of the Kepler telescope. This model is simple, so it can be solved with linear least squares, yet it is powerful enough to completely capture the PSF shape. To employ basis splines, we build a design matrix of basis spline components based on the vectors $\phi$ and $r$, which have shape

\(^5\) https://archive.stsci.edu/kepler/
We denote these matrices as \( \Phi \) and \( R \), respectively. We build a spline design matrix \( \Phi \) with degree 3 and 15 knots, evenly spaced between \(-\pi\) and \(\pi\). This spline is “wrapped” such that the function has no discontinuity as a function of angle (see Appendix D). We build a third-degree spline design matrix \( R \) with 12 knots linearly spaced in radius squared between a radius of \(1^\circ\) and \(18^\circ\) (approximately one-fourth of a pixel and 4.5 pixels). Here \( \Phi \) has \( j_\phi \) components, making it a matrix of shape \((l \times n, j_\phi)\), and \( R \) has \( j_r \) components, making it a matrix of shape \((l \times n, j_r)\). Examples of the structure of \( \phi \) and \( r \) are given below. The values of \( j_\phi \) and \( j_r \) tune the resolution of the shape model and depend on the number of knots and degree of the b-spline,

\[
\Phi = \begin{bmatrix}
\Phi_{0,0} & \Phi_{1,0} & \cdots & \Phi_{j_\phi,0} \\
\Phi_{0,1} & \Phi_{1,1} & & \cdots & \Phi_{j_\phi,1} \\
\vdots & \vdots & \ddots & \ddots & \vdots \\
\Phi_{0,n \times l} & \Phi_{1,n \times l} & \cdots & \cdots & \Phi_{j_\phi,n \times l}
\end{bmatrix},
\]

(8)

where \( \Phi_{0,0} \) indicates a single value of the basis spline for a given pixel and component in the \( \phi \) dimension, and

\[
R = \begin{bmatrix}
R_{0,0} & R_{1,0} & \cdots & R_{j_r,0} \\
R_{0,1} & R_{1,1} & & \cdots & R_{j_r,1} \\
\vdots & \vdots & \ddots & \ddots & \vdots \\
R_{0,n \times l} & R_{1,n \times l} & \cdots & \cdots & R_{j_r,n \times l}
\end{bmatrix},
\]

(9)

where \( R_{0,0} \) indicates a single value of the basis spline for a given pixel and component in the \( r \) dimension. We take the dot products of every column vector in \( \Phi \) with every column vector in \( R \) and construct a new matrix,

\[
X = \begin{bmatrix}
\Phi_{0,0}R_{0,0} & \Phi_{1,0}R_{0,0} & \cdots & \Phi_{j_\phi,0}R_{0,0} \\
\Phi_{0,0}R_{0,1} & \Phi_{1,0}R_{0,1} & \cdots & \Phi_{j_\phi,0}R_{0,1} \\
\vdots & \vdots & \ddots & \ddots \\
\Phi_{0,n \times l}R_{0,0} & \Phi_{1,n \times l}R_{0,0} & \cdots & \Phi_{j_\phi,n \times l}R_{0,0} \\
\Phi_{0,0}R_{j_r,0} & \Phi_{1,0}R_{j_r,0} & \cdots & \Phi_{j_\phi,0}R_{j_r,0} \\
\Phi_{0,0}R_{j_r,1} & \Phi_{1,0}R_{j_r,1} & \cdots & \Phi_{j_\phi,0}R_{j_r,1} \\
\vdots & \vdots & \ddots & \ddots \\
\Phi_{0,n \times l}R_{j_r,0} & \Phi_{1,n \times l}R_{j_r,0} & \cdots & \Phi_{j_\phi,n \times l}R_{j_r,0} \\
\end{bmatrix},
\]

(10)

which is the design matrix for our PRF model, which now has shape \((l \times n, j_\phi \times j_r)\).

At the center of the PRF, there are naturally few sampled points. This is illustrated in Figure 2, where there are fewer points at low values of \( r \). Close to the center of the PRF, we do not have enough information to accurately fit for any \( \phi \) dependence. As such, we do not build the cross terms for \( r < 6^\circ \) (equivalent to 1.5 Kepler pixels), and we only use the terms from \( R \). At this limit, we have no \( \phi \) dependence. This means that while the tip of the PRF is allowed to vary as a function of \( r \), there is no azimuthal dependence.

We fit to find the optimal weights \( \hat{w} \) for the design matrix \( X \), fitting to the log of flux in each pixel in the vector of the pixel values in the mean image \( \bar{f} \), normalized by an estimate of the true flux for each source in each pixel. Using Equations (B2), (B3), and (B4) we fit

\[
y = \log_{10} \left( \frac{\bar{f} \hat{w}}{f} \right),
\]

(11)

where \( \hat{f} \) denotes the mean flux value of the pixels across all time, \( f \) indicates the a priori source flux estimate in each pixel, and \( \sigma_f \) indicates the error on \( \hat{f} \). We first use the Gaia flux estimate for source fluxes (denoted as \( g \) in the above sections), which provides our first estimate \( \hat{f}_g \), but we iterate and update this value as discussed below.

We solve

\[
y = X \cdot w
\]

(13)

using Equations (B3) and (B4) to find the best-fitting weights \( \hat{w} \). These weights provide the best-fit model to \( y \). We set the covariance of the data \((K_4)\) to a diagonal matrix, where the diagonal elements are the vector \( \sigma_y^2 \). Since we are working in log space, without including \( \sigma_y \), we would implicitly upweight fainter pixels, i.e., the wings of the PRF.

We use \( \Lambda \), a diagonal matrix, where the diagonal elements are set to a vector of ones and zeros to set the weights of our fit. We build \( \Lambda \) such that elements of \( y \) that are “contaminated” (contain flux from more than one source) are set to zero, and other elements are set to 1. This ensures that only uncontaminated pixels contribute to our fit.\(^6\) We use this masking to produce a model of the PRF shape. When we ultimately fit this model to obtain the flux of each source, we will not use this mask, and we will fit all pixels, including contaminated ones.

4.4. Estimating the Flux of Every Source

In this section, we attempt to find a vector of weights \( v \) containing the true flux values of every source in the image. The weights \( \hat{w} \) and design matrix \( X \) derived above can now be used to evaluate the “normalized” PRF shape,

\[
s = 10^X\hat{w},
\]

(14)

where \( s \) is a vector describing the normalized PRF model in each pixel, with shape \( l \times n \). To create the model of the scene, we then populate a sparse matrix \( S \) of shape \((l, n)\) with the values of \( s \),

\[
S = \text{mat}(s),
\]

(15)

where \( \text{mat} \) indicates the linear transformation that converts a column vector into a matrix (the inverse of vec).

Here \( S \) is mostly made up of zeros (or extremely small values) and valued only close to a source. It is now a matrix where, when dotted with some vector \( v \) of length \( l \) (representing the intrinsic flux of each source in the image), we are able to build an estimate for the brightness of every pixel in the image (i.e., the model of the scene),

\[
\hat{f} = S \cdot v,
\]

(16)

where \( \hat{f} \) denotes our mean estimate of the image data. We can now fit to find estimates of the values \( \hat{v} \) by inverting Equation (16) and using Equations (B3) and (B4). We find \( \hat{v} \) in a given image (note that the hat symbol denotes this as our

\(^6\) We note that while this procedure is mathematically correct, in practice, if attempting to recreate this work (for example, in Python), contaminated pixels should simply be masked arrays.
best mean estimate of the true weights $v_i$). This will yield an estimate of the flux value for every source. Note that we do not mask contaminated pixels when fitting for $\hat{v}_i$, so we naturally account for contamination.

In our framework, we perform an iterative process to better estimate the PRF shape and therefore the source flux. We fit the best-fit shape model (Equation (14)) and then fit to find the best-fitting PRF weights $\hat{v}_i$ in the mean frame $\bar{f}$. We evaluate $\bar{f}$ (Equation (16)) at all pixels in the mean frame and then mask pixels where the estimated flux would be less than $1 \times 10^{-4}$ s$^{-1}$ (where we assume we have reached the noise limit of the Kepler instrument.) We then update $\hat{y}_i$ with our updated estimate of the flux in each pixel ($\hat{g}_i$), update $\Lambda$ where necessary, and reject any significant outliers in our shape modeling. This enables us to iteratively remove “background” pixels, where there is not a significant contribution of flux. We perform this iteration three times. The final version of our model $\hat{f}$ is shown in Figure 2 for the mean frame. Though we start with a broad circular “aperture” to capture pixels that contain some flux, Figure 2 demonstrates that we are able to use this iteration to remove irrelevant pixels and evaluate the model only in the region directly related to the PRF.

Because we are modeling the entire scene in the image, $S$ includes every source and fits all sources simultaneously. This is powerful; it allows us to simultaneously include every source and

The model described here assumes that all PRFs have the same shape. We know this to be incorrect. Inter- and intrapixel sensitivity variations change the shape of the PRF depending on its location, but here we assume that these effects average out over a large enough number of sources. The PRF is also a function of the flux of the source. The Kepler detector is nonlinear (i.e., the recorded counts from a source are not linearly related to the flux of the source; see KHI). Extremely bright sources (close to the saturation limit) also exhibit “halos” from internal reflections within the telescope. Here we have ignored the changes in PRF shape as a function of source brightness, but the model presented here could be updated to include this dependence.

One of the key limitations in finding the flux of targets using Equations (B3) and (B4) is that there is no way to make a non-Gaussian prior on $v_i$. This means that in practice, sources are allowed to have negative flux values. For the most part, this is not a problem, but we find that there is a small number of cases of faint, crowded targets where source fluxes estimated with our method are negative, and so unphysical. In such cases, we can assume that our “scene” model is incorrect, and either the PRF model is incorrect or the catalog of sources is incomplete or incorrect. In our tests, these cases only occurred for faint, extremely crowded targets that fall off the edge of the detector. When iteratively fitting, we remove any targets from the model where the best-fit flux (\hat{v}_i) values go below zero and re-fit, assuming there is no source at that location. It would also be possible to instead place narrow priors on these targets and enforce that the flux must be close to the values reported by Gaia.

We can now find the best-fitting flux value for every source as a function of time, $\hat{v}_i$. For the $i$th image, we fit $f_i$ (the pixels in each $i$th frame) using Equations (B3) and (B4), and we find $\hat{v}_i$, the vector of best-fitting weights for each source, such that when dotted into $S$, they provide the best-fitting flux in every pixel (Equation (16)). We also find $\hat{K}_v$, the covariance of the weights of each of the PRFs, which provides the errors on $\hat{v}_i$. We assume that the covariance of the data ($K_{v,v}$) is well described by a simple diagonal matrix, where the diagonal terms are $\sigma_v^2$.

Figure 3 shows a demonstration of the model in Equation (16) for every frame (gray line). (While our model fits all sources simultaneously, we show one source as a demonstration.) Figure 3 shows that there is a significant long-term trend in the data. This trend is not astrophysical and is instead due to velocity aberration (see KHI for further information), which causes the source to drift slowly over the duration of an observation. This drift is usually on the order of a single pixel. Because our model in Equation (16) is incapable of moving to account for this motion, the long-term trend persists in our estimate of $\hat{v}_i$. For many science cases, such as searching for the short-term transits of exoplanets, this long-term trend is undesirable but could be removed simply (e.g., the Kepler pipeline’s CBV correction was designed to remove long-term trends from velocity aberration, among other effects). However, in cases where sources fall half on the detector (e.g., at the edge of TPFs), this drift will be degenerate with real long-term variability. As such, we will improve the model to account for “common” motion across sources in the scene.

The model we have built in this section is deliberately simplistic and demonstrates how to address scene motion using a simple low-order polynomial. In some applications, it may be prudent to use more informative vectors to build $A$. For example, including the Kepler CBVs in $A$ may help address issues such as focus change, and using spacecraft position/centroid information may allow the model to account for roll motion in the K2 data set. We leave this investigation as to the best way to construct $A$ for Kepler to further work.
4.5. Motion

In our model, we fit each cadence independently, despite the fact that each frame is highly correlated, and one frame will be highly predictive of the next. This is a reasonable approach; we want to obtain the flux of each source independently of the flux measured in neighboring frames, because a priori, we have no information about how a source should truly vary astrophysically. However, we would like to allow our model to take into account the motion of the scene for velocity aberration. To do this, we can update our model \( S \) and have the entire scene move in the same way that the data drift. We will assume that the motion is small (\(<\)the width of a pixel), so the model at any given frame can be estimated as the model in the mean frame, multiplied by some small-valued function, describing the change in the model.

We will make the following implicit assumptions: (1) the Kepler background is effectively zero and can be ignored; (2) sources, on average, do not vary astrophysically (an instantaneous astrophysical flux for each source is usually very close to the average over time); and (3) the motion is small and varies smoothly over a long time period. In this section, we will use the same model for the PRF from Section 4.3 and update the way we fit for the flux of each source to account for some time dependence.

To update the model for velocity aberration, we first bin the data function of time, creating a smaller number of time-averaged frames. We bin in this way for two reasons: first, to increase the S/N for the motion and average over astrophysical noise, and second, to decrease the amount of data that need to be fit and speed up our calculation of the model. To fit the drift, we divide the binned frames by the mean image \( \bar{f} \), normalizing each pixel time series to the average flux value. We will denote this binned, normalized flux as \( f_b \), defined as

\[
 f_b = B \cdot \left( \frac{f}{\bar{f}} \right),
\]

where

\[
 B = \begin{bmatrix}
 \frac{1}{b} & \cdots & \frac{1}{b} \\
 \frac{1}{b} & \cdots & \frac{1}{b} \\
 \frac{1}{b} & \cdots & \frac{1}{b} \\
 \end{bmatrix},
\]

where \( b \) is the number of cadences we sum to create each bin (in our example, we use \( b = 200 \), which provides a resolution of 4 days and is adequate to capture Kepler’s long-term motion). The matrix \( B \) is made up of entries of \( \frac{1}{b} \) for cadences we wish to bin. Now \( f_b \) is a matrix with shape \( (l \times n \times m/b) \). Assuming that most sources, on average, do not vary, \( f_b \) should only vary due to instrumental effects and velocity aberration.

To fit common motion in the scene, we will use another basis spline. In this case, unlike when fitting the PRF shape, there is no radial symmetry; the entire image drifts slowly in the \( x \) and \( y \) dimensions. As such, we will fit a basis spline in Cartesian coordinates.

Using the same methods discussed in Section 4.3, we build a simple 2D basis spline design matrix in each Cartesian dimension, \( \delta X \) and \( \delta Y \), and calculate all of the cross terms to build a design matrix, which we will denote as \( A_0 \),

\[
 \delta X = \begin{bmatrix}
 \delta X_{0,0} & \delta X_{0,1} & \cdots & \delta X_{j_x,0} \\
 \delta X_{1,0} & \delta X_{1,1} & \cdots & \delta X_{j_x,1} \\
 \vdots & \vdots & \ddots & \vdots \\
 \delta X_{n_x,0} & \delta X_{n_x,1} & \cdots & \delta X_{j_x,c} \\
 \end{bmatrix},
\]

(19)

where \( \delta X_{0,0} \) indicates a single value of the basis spline for the first pixel and component in the \( \delta x \) dimension, and

\[
 \delta Y = \begin{bmatrix}
 \delta Y_{0,0} & \delta Y_{0,1} & \cdots & \delta Y_{j_y,0} \\
 \delta Y_{1,0} & \delta Y_{1,1} & \cdots & \delta Y_{j_y,1} \\
 \vdots & \vdots & \ddots & \vdots \\
 \delta Y_{n_y,0} & \delta Y_{n_y,1} & \cdots & \delta Y_{j_y,c} \\
 \end{bmatrix},
\]

(20)

where \( \delta Y_{0,0} \) indicates a single value of the basis spline for the first pixel and component in the \( \delta y \) dimension. Here \( c \) is the length of each column vector,

\[
 c = l \times n \times m/b,
\]

(21)

and \( j_x \) and \( j_y \) are the number of basis spline components (column vectors) in \( \delta X \) and \( \delta Y \), respectively. We take the dot products of every column vector in \( \delta X \) with every column vector in \( \delta Y \) and construct a new matrix,

\[
 A_0 = \begin{bmatrix}
 \delta X_{0,0}\delta Y_{0,0} & \delta X_{0,0}\delta Y_{0,1} & \cdots & \delta X_{0,0}\delta Y_{j_y,0} \\
 \delta X_{0,1}\delta Y_{0,0} & \delta X_{0,1}\delta Y_{0,1} & \cdots & \delta X_{0,1}\delta Y_{j_y,1} \\
 \cdots & \cdots & \ddots & \cdots \\
 \delta X_{n_x,0}\delta Y_{0,0} & \delta X_{n_x,0}\delta Y_{0,1} & \cdots & \delta X_{n_x,0}\delta Y_{j_y,0} \\
 \end{bmatrix},
\]

(22)

which is the design matrix for our time-dependent model, which now has shape \( (l \times n \times m/b, j_x \times j_y) \). We then create a time-dependent design matrix \( A \) by taking the elementwise product of \( A_0 \) and a third-order polynomial in time \( t \), assuming that this is a reasonable model for the slow, long-term velocity aberration. This gives us a design matrix as a function of time,

\[
 A = \begin{bmatrix}
 A_0 & A_0\alpha t_b & A_0\alpha t_b^2 & A_0\alpha t_b^3 \\
 \end{bmatrix},
\]

(23)

which has shape \( \{l \times n \times m/b, j_x \times j_y \times 4\} \). Here \( t_b \) indicates a vector of the binned time values (corresponding to the average time at every element of \( f_b \) with length \( l \times n \times m/b \)), and \( \alpha \) indicates the elementwise product.

We then fit the model to the normalized, binned data vector \( f_b \) using Equations (B3) and (B4) to find \( \hat{u} \), the best-fitting weights of the time-dependent matrix \( A \).

The motion model at the \( i \)th frame is then simply

\[
 \mathbf{A}_i = \begin{bmatrix}
 A_0 & A_0\alpha t_i & A_0\alpha t_i^2 & A_0\alpha t_i^3 \\
 \end{bmatrix} \hat{u},
\]

(24)

where \( t_i \) indicates the time at the \( i \)th frame, and \( \mathbf{A}_i \) is a vector whose entries scale the mean PRF shape in a given frame \( \{s_i\} \) to the correct shape given the motion in each frame. We iteratively fit this model to the normalized \( f_b \), masking out sources that show a significant discrepancy from the motion model (suggesting that there is large-scale astrophysical variability).

To build the PRF shape model matrix \( S \), that accounts for this motion in the \( i \)th frame, we simply populate the matrix with
the elementwise multiplication of $s$ and $I_i$, \[ S_i = \text{mat}(s \cdot I_i). \] (25)

We then fit to find the best-fitting weights $v_i$ for each source, in each frame, using each unique $S_i$ model of the scene. We derive the flux for every source simultaneously, accounting for common motion in the scene.

Figure 4 shows the normalized, binned data $f_b$ as a function of the $\delta x$ and $\delta y$ dimensions alongside our best-fitting model. Figure 4 shows that in the case of our demonstration data, the sources are, on average, brighter in the top-left corner of $\delta x$, $\delta y$ space at the start of the image stack and in the bottom-right corner toward the end of the image stack (upper panels). This shows that the sources are drifting, over time, from one part of $\delta x$, $\delta y$ space to another. Our model (lower panels) accounts for this motion.

Figure 3 shows the results of fitting the model, including velocity aberration, in black. The long-term drift due to velocity aberration has been largely removed, leaving the astrophysical variability intact. While this has removed common motion from all sources, this method cannot account for local effects, such as a change in the sensitivity of the pixels. Spatially varying intrapixel sensitivity is a known effect in Kepler, which causes K2 systematic noise as the telescope roll motion moves the source over differentially sensitive parts of the pixel. As such, while this improves the photometry, it cannot completely account for this motion. Additionally, a more complete model would account for differential velocity aberration across the image. We find that this method is adequate to improve the photometry of a source that is half off the edge of the detector.

4.5.1. Focus Change

As discussed in Van Cleve & Caldwell (2016), the Kepler instrument underwent a significant focus change during observations, in particular during the start of data collection (or after any data collection gaps) as the telescope changed temperature. This focus change causes a significant shape change to the PRF and, in some cases, a significant source drift. The focus change can be seen in Figure 6 after the gaps in data collection as a significant “hook”-shaped systematic. In this work, we do not account for this focus change. Instrumental systematics such as this are well corrected by methods such as the Kepler pipeline’s CBV correction, so we assume that the focus change can be corrected in postprocessing steps. It would be possible to create a more flexible time-dependent model and use more time points (i.e., $b < 200$) to account for some of this short-term PRF shape change.

4.6. Sparsity

In the above sections, we have discussed a model that fits the scene in a given stack of Kepler images. For our test set, we are able to fit the PRF shape model and motion model and produce light curves of all 576 sources in under 4 minutes running on a personal laptop. This speed is only possible when we take advantage of the sparsity of the matrices described above. Many of the variables discussed above are extremely sparse, as pixels far from sources effectively contribute zero flux. By using scipy’s sparse library (Virtanen et al. 2020), we are able to capitalize on the sparsity of these matrices and avoid loading large arrays of zeros into memory, making it tractable to fit the scene for this data set on an average machine. The tools we have developed for this work are available in an open-source Python tool under an MIT License, and v1.0.0 is archived in Zenodo (Hedges & Martínez-Palomera 2021). This process is expandable to larger data sets; for example, it is possible to estimate the PRF model and source flux of every source in a Kepler full-frame image. For a single channel, this requires building a matrix of $\approx 100,000$ pixels and $\approx 13,250$ sources. Solving to find the PRF model and the weights of the sources then takes approximately 150 s for a single frame and requires 500 Mb of memory allocation. This demonstration shows that LFD photometry is scalable and applicable to larger data sets (for example, the TESS data set).

4.7. Diagram of the Method

Figure 5 shows an overview of the key stages involved in our method. Depending on the specific requirements of a given data set, users may choose to fit either with or without accounting for long-term trends. By following Sections 4.2, 4.3, and 4.4, users can build a simple photometry of sources (left side of flowchart in Figure 5), and by following Sections 4.2, 4.3, and 4.5, users can build photometry of sources accounting for long-term motion (right side of flowchart in Figure 5).

4.8. Results

In Sections 4.3, 4.4, and 4.5, we have discussed our method for modeling the scene in a stack of astronomical images,

7 Using a 2017 MacbookPro, 16 GB RAM, 2.9 GHz Intel Core i7 processor.
8 https://github.com/SSDataLab/psfmachine
solving for the flux of all sources. Our method derives the photometric light curves of each of the 576 sources in the scene, with errors. The LFD photometry is able to fit the instrument PRF and velocity aberration and then model the flux in each source as a function of time in under 4 minutes on a personal computer. In this section, we will discuss these results for the target of interest: KOI-608.

Figure 6 shows the Kepler TPF data for KOI-608 and the best-fit light curves for both targets in the data. Figure 7 shows the same light curves alongside the Kepler pipeline light curve for KOI-608, flattened with a simple Savitsky Golay filter, folded at the period of KOI-608, and binned. The transiting signal clearly originates from Gaia EDR3 2073618323728332544, and our method is able to separate both targets exceptionally well.

The transiting signal around Gaia EDR3 2073618323728332544 caused KOI-608 to be identified as a false-positive exoplanet candidate by the Kepler mission. It was discounted due to a centroid offset during transit. In this case, the transiting signal still remains a viable case for a transiting planet, with a depth potentially consistent with a very large planet or substellar companion and a transit-like shape.

The results of our transit model are shown in Table 1. Gaia DR3 quotes a parallax value for Gaia EDR3 2073618323728332544 of 0.342 \pm 0.267 mas. Since the parallax errors for this target are so large, the luminosity, mass, and radius errors are also broad. Despite these broad errors in stellar parameters, we obtain a radius estimate for the transiting object of 2.52^{+0.16}_{-0.14} R_{\text{Jup}}. This radius is consistent with a small substellar object around Gaia EDR3 2073618323728332544. This could imply that the transit is from either a very large planetary companion or a small substellar or stellar companion.

4.8.1. Comparison with PDCSAP Flux

The lightkurve Python package provides the ability to calculate an estimated CDPP metric, similar to the metric used by the Kepler pipeline to determine light-curve quality. The estimate_cdpp function of light-curve uses the simpler “sgCDPP proxy algorithm” discussed by Gilliland et al. (2011) and Van Cleve et al. (2016). This single numeric value can be used as a measure of the noise properties of the data. We compare the light curves obtained from LFD photometry with the PDCSAP light curves available for targets in our sample. We remove any targets where there is significant crowding, as our method separates crowded targets into individual light curves. Figure 8 shows the estimated CDPP for both sets. We find that in our test case, LFD photometry is able to reach a similar precision to the PDCSAP products and does not significantly increase or decrease the photometric noise of the time series compared to PDCSAP.

4.8.2. Validation

The methods described here have been demonstrated using KOI-608 but would be equally applicable to other Kepler objects of interest that were discounted due to a significant centroid shift to search for further planetary candidates. In the literature, planetary candidates are usually validated as true planets using a suite of different approaches (for example, the Kepler Robovetter, Thompson et al. 2018). Many of these approaches are invalid in the case of highly contaminated targets whose photometry is extracted in this way. Tests such as centroiding tests are not valid given our approach (centroiding tests are only applicable in the case of aperture photometry).

\footnote{Gaia EDR3 does not provide updated stellar parameters over previous data releases, so we use Gaia DR2 here.}
Further photometric observations at high spatial resolution are frequently employed to search for contaminating objects, but in this case, we are aware of a significant contaminant next to the object in question. Ground- or space-based follow-up to reobserve the transits would require a high S/N and spatial resolution to be able to separate these blended targets and resolve the small transit around the fainter background target. These factors would make it more difficult to confirm transiting signals from Kepler objects of interest analyzed in this way, but further observations with high-resolution imaging, high-resolution spectra, and/or large ground-based facilities may be able to confirm targets similar to the transiting signal around Gaia EDR3 2073618323728332544.

5. Discussion

In this work, we have presented a framework for modeling a field of stars from Kepler and demonstrated the power of the method to separate even highly contaminated sources. In this section, we will discuss some of the limitations and assumptions in our framework.

Our model is based on an input catalog of source positions built from the Gaia EDR3 catalog with proper motions accounted for. If this input catalog is missing sources, our model is not able to account for this, and our results will be inaccurate. We find this assumption to be valid for Kepler, where the pixel scale is much larger than the spatial resolution of Gaia. In this work, we have only accounted for sources in the Gaia EDR3 catalog down to 18th magnitude in Gaia and assumed that “missing” sources fainter than 18th magnitude will not adversely affect our results. This assumption could be relaxed in future work.

We have also assumed that there is no distortion over the image, and PSF shapes do not vary. For Kepler and TESS data, this assumption holds for reasonably small patches of the CCD (e.g., hundreds of pixels); however, the PSF shape is known to distort toward the edges of the focal plane (see, e.g., KIH or the TESS Instrument Handbook). The model presented in this work...
would be possible to analyze the residuals between the model and the data and search for significant deviations from the model that can be attributed to a shape change due to chromatic aberration (i.e., a source changing color).

For TESS, the PRF shape can change significantly from frame to frame due to intracadence motion caused by “jitter.” In our method, we assume that these changes to the PRF shape are small and common to all sources. We assume that these PRF shape changes cause systematics in the retrieved time series that can then be corrected by postprocessing (which is the approach used for SAP). In our method, we create a PRF model for the average frame, but it would also be possible build a unique PRF shape model at every frame. In practice, we find that this produces worse photometry, since estimating the PRF shape using a single frame provides a noisy estimate compared with the mean frame. For cases where there is a significant shape change and a large number of sources (e.g., perhaps large ground-based surveys affected by seeing), our assumptions may not hold.

We have assumed that the bulk motion in the scene is (1) less than the width of the PRF, (2) smooth over time, and (3) common across all sources. We require the motion to be less than the width of the PRF for our approach to be valid (Section 4.5), since we essentially assume that the PRF in any given frame can be modeled as the PRF in the mean frame, multiplied by some (small) factor. For large motion, this does not hold. In Kepler data, and largely in TESS, these three assumptions about motion are true. For K2 data, motion is small and common across sources, but the 6 hr roll motion breaks the assumption that motion is smooth over time. We suggest that this could potentially be alleviated by changing the time dependence in Equation (23) from a simple polynomial to a function of the centroid position, similar to the approach in Vanderburg & Johnson (2014).

Finally, we have largely assumed that the inter- and intrapixel sensitivity variations of the instrument are negligible in this work. For interpixel sensitivity variations, we assume the Kepler pipeline is able to accurately flat-field the data. However, the Kepler detector is known to exhibit significant intrapixel sensitivity variations. It is these variations that, when coupled with the significant roll motion of the Kepler spacecraft, cause the distinct “sawtooth” noise pattern in K2 data. When we build our PRF model, we are implicitly assuming that over a large number of sources, these variations average out, so that we can fit a smooth function to the PRF with the average intrapixel sensitivity. When we fit for the weights of a source, accounting for scene motion, we are not accounting for this sensitivity change across the pixel in the method described in this work. For work with Kepler, this may cause a long-term trend in the data that is not accounted for by our velocity aberration correction (e.g., see Figure 3). For Kepler, and for the purpose of finding short-period variability, our approach is reasonable. For K2 data, an additional correction for intrapixel sensitivity will be needed. Further investigation is needed to find the best approach to including intrapixel sensitivity in the model described here.

6. Summary

In this work, we have presented a method of obtaining photometry that we name LFD, where we estimate a model for the PRF of sources and then model the flux of every source simultaneously, fixing the source positions to those estimated by Gaia. The LFD photometry is general and fast. The method we have presented in this work is highly applicable to crowded sources, as we have demonstrated with KOI-608. We are able to produce accurate photometry of sources that are separated by 1 pixel on the detector with a contrast of 2 mag. The LFD photometry is capable of fitting the flux of sources even when significant portions of the PRF fall off the detector by fitting the wings of the PRF.

Further investigation is needed to fully understand the limits of LFD photometry in crowded fields as a function of both target separation and target magnitude in crowded regions. At some spatial separation and contrast, our model will not be able to adequately separate targets. We leave this investigation to future work. Such an investigation may help us develop methods and practices for vetting planets identified in LFD photometry light curves.
The Kepler and K2 superstamp data, which cover several clusters, may benefit considerably from analysis using this approach. The photometry of sources in the K2 microlensing superstamp, which has many hundreds of thousands of sources, could be extracted using this method. However, we rely on a complete catalog for our method, and so fields toward the galactic center could be difficult to analyze, since Gaia may be less complete.

NASA’s TESS mission has much larger pixels (27‴), a broader PRF, and a fainter magnitude limit than Kepler, resulting in much more significant crowding. We suggest that LFD photometry is highly applicable to TESS fields and will be useful for separating contaminants from exoplanet host stars to better estimate the radius of transiting objects in the TESS survey.

This paper includes data collected by the Kepler mission and obtained from the MAST data archive at the Space Telescope Science Institute (STScI). Funding for the Kepler mission is provided by the NASA Science Mission Directorate. The STScI is operated by the Association of Universities for Research in Astronomy, Inc., under NASA contract NAS 526555. This work has made use of data from the European Space Agency Gaia (https://www.cosmos.esa.int/gaia), processed by the Gaia Data Processing and Analysis Consortium (DPAC; https://www.cosmos.esa.int/web/gaia/dpac/consortium). Funding for the DPAC has been provided by national institutions, in particular the institutions participating in the Gaia Multilateral Agreement. This research made use of exoplanet (Foreman-Mackey et al. 2021) and its dependencies (Kipping 2013; Astropy Collaboration et al. 2013, 2018; Salvatier et al. 2016; Theano Development Team 2016; Kumar et al. 2019; Luger et al. 2019; Agol et al. 2020; Foreman-Mackey et al. 2021). Funding for this work for C.H. and J.M.P. is provided by grant No. 80NSSC20K0874 through NASA ROSES.

Facility: Kepler.
Software: lightkurve (Lightkurve Collaboration et al. 2018), exoplanet (Foreman-Mackey et al. 2021), astropy (Astropy Collaboration et al. 2013, 2018), scipy (Virtanen et al. 2020), PSFMachine (Hedges & Martínez-Palomera 2021).

Appendix A
Paper Naming Conventions
In this paper, we use lowercase italics to describe single-valued variables, lowercase bold to describe vectors, and uppercase bold to describe matrices.

| Parameter | Description |
|-----------|-------------|
| \( n \)   | Number of pixels |
| \( m \)   | Number of cadences |
| \( l \)   | Number of sources |
| \( i \)   | Indicates the \( i \)th frame in a stack of images |
| \( \sigma \) | Indicates the measurement errors on a variable (in this case, a dummy variable \( y \)) |
| \( K \)   | Indicates the covariance matrix of a vector (in this case, a dummy variable \( y \)) |
| \( \tilde{y} \) | “Hat” symbol denotes a mean estimate of a variable (in this case, a dummy variable \( y \)) |
| \( \bar{y} \) | “Bar” symbol denotes the mean of a variable in time (in this case, a dummy variable \( y \)) |
| \( x, y \) | Cartesian coordinates |
| \( r, \phi \) | Polar coordinates |
| \( \Delta x \) | Vector of separations from a source in Cartesian coordinate \( x \) with length \( l \times n \) |
| \( \Delta y \) | Vector of separations from a source in Cartesian coordinate \( y \) with length \( l \times n \) |
| \( r \) | Vector of separations from a source in polar coordinate \( r \) with length \( l \times n \) |
| \( \phi \) | Vector of separations from a source in polar coordinate \( \phi \) with length \( l \times n \) |
| \( f \) | Vector of pixel fluxes with length \( n \times m \) (the data) |
| \( \bar{f} \) | Vector of pixel fluxes averaged over time with length \( n \) |
| \( g \) | Vector of pixel flux estimates built from source flux estimates from Gaia EDR3 with length \( l \times n \) |
| \( X' \) | Design matrix for circular apertures with shape \((l \times n, 9)\) |
| \( \hat{w}_0 \) | Best-fitting weights for design matrix \( X' \) to model circular apertures, vector with length 9 |
| \( f_0 \) | Estimate of flux in each pixel from the initial circular model, vector with length \( n \) |
| \( \hat{f} \) | Vector of pixel flux estimates with length \( n \) |
| \( \hat{v} \) | Vector of source flux weights with length \( l \) |
| \( s \) | Vector of the normalized PRF shape model with length \( l \times n \) |
| \( S \) | Matrix of the normalized PRF shape with shape \((l, n)\), the matrix version of \( s \) and when dotted with \( \hat{v} \) will give the scene model |
| \( \Lambda \) | Matrix of “weights” for each pixel, a diagonal matrix consisting of ones for uncontaminated pixels and zeros for contaminated pixels, has shape \((n, n)\) |
| \( \Phi \) | Design matrix of basis spline components in the \( \phi \) dimension, has shape \((l \times n, j_\phi)\) |
| \( R \) | Design matrix of basis spline components in the \( r \) dimension, has shape \((l \times n, j_r)\) |
| \( j_\phi \) | Number of components in the design matrix \( \Phi \) |
| \( j_r \) | Number of components in the design matrix \( R \) |
| \( X \) | Design matrix for PRF model, has shape \((l \times n, j_\phi \times j_r)\) |
| \( B \) | Matrix used to create \( f_0 \) |
| \( b \) | Number of cadences to bin in time |
| \( f_0 \) | Flux array, having been binned in time, has shape \((l \times n, m/b)\) |
| \( \Delta X \) | Matrix consisting of basis spline components for motion model, each column vector is one spline component of the \( \Delta x \) vector, has shape \((l \times n, j_r)\) |
| \( \Delta Y \) | Matrix consisting of basis spline components for motion model, each column vector is one spline component of the \( \Delta y \) vector, has shape \((l \times n, j_r)\) |
Appendix B
Linear Least Squares

In this paper, we solve to find a best-fitting model using linear least squares. In general, if we have a vector \( y \) containing data, we assume that it is Gaussian distributed,

\[
y \sim \mathcal{N}(m, K_y),
\]

where \( K_y \) is the covariance of the data, and \( m \) is a vector with the same length as \( y \) and is the mean. We model \( m \) as

\[
m = X \cdot w,
\]

where \( X \) is a design matrix, and \( w \) is a vector of weights. We can find the best-fitting weights for the design matrix by solving the linear system

\[
K_w^{-1} = X^\top \cdot K_y^{-1} \cdot X + C^{-1},
\]

\[
\hat{w} = K_w^{-1} \cdot (X^\top \cdot K_y^{-1} \cdot y + C^{-1} \cdot \mu),
\]

where

\[
K_{y}^{-1} = (K_y \circ \Lambda^{-1})^{-1}
\]

and

\[
C^{-1} = I \left( \frac{1}{\sigma^2} \right),
\]

\( \hat{w} \) is a vector of the estimates of the best-fitting coefficients, \( K_y \) is the covariance matrix of the data, and \( \Lambda \) is an optional diagonal matrix consisting of weights for each element in \( y \). Setting \( \Lambda \) to the identity matrix sets all pixels to equal weight; setting \( \Lambda \) to a diagonal matrix of ones and zeros (or small values) allows some elements of \( y \) to be effectively excluded (masked) from the fit. Here \( \mu \) is the prior mean of each of the \( w \), \( \sigma \) is the prior standard deviation of \( w \), \( \hat{w} \) has a length equal to the number of components in the design matrix, and \( K_w \) is the covariance matrix of \( w \). We solve this system several times in this work. In practice, rather than using \( \Lambda \) to change the weight of each element in the fit, we mask out elements of \( y \) and \( K_y \) in our Python implementation using numpy arrays.

Appendix C
Constructing B-Splines

Basis splines, or B-splines, provide a way for us to build a simple linear model of piecewise polynomials, which are forced to vary smoothly across all pieces. In this work, we use B-splines to model several aspects of the data. Our use of B-splines converts a vector into a matrix, where each column of the matrix is a spline component.

The B-spline polynomial pieces are defined by “knots.” Each spline component is zero outside of a given pair of knots and valued between knots. Splines are also defined by their degree, which we will denote as \( k \).

For example, in Section 4.3, we build a matrix of spline components from a vector \( r \) (the radial distance from a source). We specify knots \( r_0, r_1, r_2, \ldots, r_n \), which are single values, and all lie between the lowest and highest values of \( r \). All of the knots must be in ascending order. Depending on the degree, the knots at the beginning and end of the sequence must be repeat values. We specify a total of \( n_k \) knots. There are \( n_r \) elements in the vector \( r \).

For a degree 1 b-spline, a single spline component (column vector) for the \( i \)th knot is given as

\[
b_{i,1} = \begin{cases} 
1 & \text{if } r_{i,1} \leq r < r_{i,1+1}, \\
0 & \text{otherwise}.
\end{cases}
\]

The matrix defining the b-spline model is then

\[
R_i = \begin{bmatrix} b_{0,1} & b_{1,1} & b_{2,1} & \cdots & b_{n,1} \end{bmatrix},
\]

where each \( b_i \) is a column vector of length \( n_r \), so \( R_i \) is a matrix with shape \((n_r, n_k)\). Each \( b_i \) is a step function, which is 1 between knots.

The B-splines of order \( k > 1 \) are defined by a recursive relation. Each vector of the second-order B-spline is given as

\[
b_{i,2} = \omega_{i,1} \circ b_{i,1} + (1 - \omega_{i+1,1}) \circ b_{i+1,1},
\]

where \( \circ \) indicates the elementwise product and

\[
\omega_{i,k} = \begin{cases} 
\frac{r - r_i}{r_{i+k} - r_i} & r_{i+k} \neq r_i, \\
0 & \text{otherwise}
\end{cases}
\]

This creates a set of piecewise first-order polynomials between each set of knots. The second-order basis spline matrix \( R_2 \) is given by the matrix of all column vectors for every \( i \).

Higher orders of the b-spline can be made recursively using

\[
b_{i,k+1} = \omega_{i,k} \circ b_{i,k} + (1 - \omega_{i+k,1}) \circ b_{i+1,k}.
\]

In this work, we find that a third-order B-spline adequately trades model complexity with efficient computing, and we employ \( k = 3 \) throughout this work. The above procedure is used in this work to create B-spline matrices \( \delta X, \delta Y, \Phi \), and \( R \) from vectors \( \delta x, \delta y, \phi \), and \( r \). We direct the reader to Appendix D for an example of a Python implementation of a B-spline that wraps at a value of \( \pi \), such that the value at \( -\pi \) is equal to the value at \( \pi \), which we use to create the matrix \( \Phi \). An
example of the matrix structure obtained from an input vector is shown in Figure 9.

Appendix D
Wrapped Spline Python Implementation

```python
import numpy as np
def wrapped_spline_matrix(input_vector, degree = 2, nknots = 10):
    """
    Creates a matrix of splines according to the input vector. This will wrap between -pi and pi.
    This is meant to be used to build the basis vectors for periodic data, like the angle in polar coordinates.
    Parameters
    ----------
    input_vector : numpy.ndarray
        Input data to create basis, angle values MUST BE BETWEEN -PI and PI.
    degree : int
        Degree of the spline basis
    nknots : int
        Number of knots for the splines
    Returns
    -------
    folded_basis : numpy.ndarray
        Array of folded-spline basis
    """
    # Higher order basis
    for order in np.arange(1, degree):
        basis_1 = []
        for idx in range(len(t) - 1):
            a = ((x1 - t[idx]) / (dt * order)) * basis[idx]
            if ((idx + order + 1)) < (nt-1):
                b = (- (x1 - t[(idx + order + 1)]) / (dt * order))
            else:
                b = np.zeros(len(x1))
            basis_1.append(a + b)
        basis = np.vstack(basis_1)
        folded_basis = np.copy(basis)[: nt // 2, : len(x)]
        for idx in np.arange(-(degree), 0):
            folded_basis[idx, :] += np.copy(basis)[nt // 2 + idx, len(x):]
        return folded_basis
```

References

Agol, E., Luger, R., & Foreman-Mackey, D. 2020, AJ, 159, 123
Aigrain, S., Hodgkin, S. T., Irwin, M. J., Lewis, J. R., & Roberts, S. J. 2015, MNRAS, 447, 2880
Alard, C. 2000, A&AS, 144, 363
Alard, C., & Lupton, R. H. 1998, ApJ, 503, 325
Anderson, J., & King, I. R. 2000, PASP, 112, 1360
Anderson, J., & King, I. R. 2006, Instrument Science Report ACS 2006-01

ORCID iDs

Christina Hedges @ https://orcid.org/0000-0002-3385-8391
Rodrigo Luger @ https://orcid.org/0000-0002-0296-3826
Jessie Dotson @ https://orcid.org/0000-0003-4206-5649
Geert Barentsen @ https://orcid.org/0000-0002-3306-3484

Figure 9. Example of the spline matrix Φ used in this work at different degrees. In this work, we use a third-degree spline. This spline has been created using the "wrapped" spline code given in Appendix D, so the value at π is the same as the value at -π. The B-splines are broken into piecewise polynomials. Increasing the degree of the spline increases the order of the polynomial in each piece. Fitting Φ to a vector using the equations in Appendix B will result in a smooth trend.
