PYCHEOPS Cookbook v1.0

Thomas G. Wilson; `tgw1@st-andrews.ac.uk`

November 2021

Note: For the construction of this cookbook PYCHEOPS was successfully installed and run using Ubuntu 18 and Python 3.6.

Abstract

The purpose of this document is to provide users of *CHEOPS* data with information on the PYCHEOPS Python module. This cookbook details PYCHEOPS dependencies and installation, how to download *CHEOPS* data, and several data analysis recipes. Therefore, this document can be used as a walk-through of obtaining and analysing *CHEOPS* data, or as a reference guide of PYCHEOPS.

Contents

1 Introduction

pycheops is a Python package that contains tools for analysis of light curves taken by the ESA *CHEOPS* spacecraft [2] (http://cheops.unibe.ch). This includes downloading, visualising, and decorrelating *CHEOPS* data, fitting transits and eclipses of exoplanets, and calculating light curve noise. This package has been primarily written by Pierre Maxted (`p.maxted@keele.ac.uk`) and is current under development. Therefore, the package is constantly being developed and this document will be updated accordingly. Further documentation and the source code can be found here: https://github.com/pmaxted/pycheops. In this document, the dependencies of pycheops are given in Section 2 and instructions on how to install pycheops provided in Section 3. Section 4 details how to retrieve *CHEOPS* data using DACE with several useful recipes for the analyses of *CHEOPS* light curves using PYCHEOPS presented in Section 5. Finally, in the Appendices to this document, a holistic walkthrough encapsulating the majority of PYCHEOPS functionality detailed in Sections 4 and 5 is given as a ready-to-use example code, with the PYCHEOPS functions given in the aforementioned recipes also described in detail.

2 PYCHEOPS Dependencies

There are multiple package dependencies that PYCHEOPS requires to run successfully. Several of these, and the required versions, are checked during installation of PYCHEOPS, however for completeness (and to allow for comparison against), a list of the dependencies is presented. Packages can be installed using `conda install` or `pip install`, with a few examples given below. Anaconda can be installed following the procedure outlined at https://docs.anaconda.com/anaconda/install/. Pip is installed as a part of Anaconda, however it can also be installed using `sudo apt install pythonX-pip`, where `X` represents the version of Python used and here is equal to 3.

PYCHEOPS has been successfully run using Python 3.6, however it is expected to be stable using later versions. Therefore, it is recommended that users of PYCHEOPS install Python 3.6 or later. It should be noted that if Python is installed via the typical Anaconda method package dependencies may be satisfied, as highlighted in the following table.

Package	**Version**	**Command**
Python	3.6.10	`conda create --name my_env python=3.6`
asteval	0.9.13	`conda install -c conda-forge asteval`
astropy	4.0.2	`conda install -c anaconda astropy`
astroquery	0.4.0	`conda install -c conda-forge astroquery`
cdspyreadme	1.3.4	`pip install cdspyreadme`
celerite2[1]	0.0.1	`pip install celerite2`
corner	2.0.1	`conda install -c conda-forge corner`
dace[2]	1.8.0	See Section 4.
emcee	3.0.2	`conda install -c conda-forge emcee`
lmfit	0.9.14	`conda install -c conda-forge lmfit`
matplotlib	3.3.4	`conda install -c anaconda matplotlib`
numba	0.51.2	`conda install -c anaconda numba`
numpy	1.19.2	`conda install -c anaconda numpy`
photutils	1.0.2	`conda install -c conda-forge photutils`
pybind11	2.7.0	`conda install -c conda-forge pybind11`
requests	2.25.1	`conda install -c anaconda requests`
scipy	1.5.2	`conda install -c anaconda scipy`
setuptools	52.0.0	Included in Python installation of conda.
tqdm	4.62.0	`conda install -c conda-forge tqdm`
uncertainties	3.1.2	`conda install -c conda-forge uncertainties`

Notes. 1) For the installation of the celerite2 package a recent C++ compiler is required, so users may need to update their compiler (for example Microsoft Visual C++).
2) Multiple modules of PYCHEOPS requires the DACE package if the user wants to use the DACE Python API to download datasets or retrieve stellar and planetary parameters. As utilising the DACE database to access *CHEOPS* data is an important first step in the analysis of *CHEOPS* data, the installation of the DACE package is the focus of a separate section below (see Section 4.2).

3 PYCHEOPS Installation

The current version of PYCHEOPS can be installed using the command:

```
pip install pycheops
```

As mentioned above, during the installation of PYCHEOPS the majority of the package dependencies listed above will be checked to determine if they are installed and satisfy any version requirement. If Pip does not detect a package it will subsequently be downloaded and installed, along with any further dependencies that package requires. Therefore, it is not necessary to install the dependencies listed above prior to the installation of PYCHEOPS.

An exception is the ellc package that is currently not a strict dependency, and therefore will not be checked for during PYCHEOPS installation, however it may be used in the ld module (for calculation of limb-darkening coefficients) and thus may need to be installed for users of this module.

Following installation the setup script must be run. This is done by running the following commands in a Python session:

```
from pycheops.core import setup_config
setup_config()
```

You will then be prompted to enter a data cache directory. This is the directory that PYCHEOPS will search for *CHEOPS* data when a module has a data input, for example, when creating a Dataset object. If the users press return to accept the default directory, it is set to the home directory. The directory is stored as the variable `data_cache_path` in the PYCHEOPS configuration file (pycheops.cfg) that can be found in the home directory. *Note: it is important that the data cache directory is set to a valid value otherwise errors will arise when trying to important data.*

PYCHEOPS is now ready to use! For subsequent versions PYCHEOPS can be updated with:

```
pip install pycheops --upgrade
```

For some updates, new features might be added that require the configuration file to be updated. If that is the case, users should run the `setup_config` code again.

4 Accessing *CHEOPS* Data

With PYCHEOPS installed it is possible to utilise its modules to analyse *CHEOPS* data. There are two methods you can use to access *CHEOPS* data; firstly, via the Université de Genève Data & Analysis Center for Exoplanets (DACE) website, and second, via the Python-based DACE API. For a quick inspection of the data, and use of existing visualisation tools, the web interface should suffice for most users. However, for bulk data access and more detailed analyses it is recommended that users utilise the Python API. This can either be done in PYCHEOPS or separately, as will be outlined below.

First time users of DACE must go to the website and request an account. This can be done by going to https://dace.unige.ch/dashboard/ and clicking on “Sign in/Create Account” in the top right of the page, and then “Request an account here” that will take you to an account sign up page. Following confirmation, users will be able to log in and start querying the data.

4.1 Using the DACE Web Interface

On the DACE homepage the "Cheops" hexagon should now be visible in the top left and users can go to the *CHEOPS* database by clicking on it. If it is not visible users may need to request access by clicking on their username in the top right of the page and selecting "Requests". The current instrument, program, and mission accesses are shown on the left with it possible to request access on the right of the page. If the "Cheops" hexagon is still not visible after gaining access, users may wish to submit a bug report by clicking on their username in the top right of the top and selecting "Bug report".

4.1.1 Viewing the Data

The main DACE *CHEOPS* page (https://dace.unige.ch/cheops/) provides access to both the observations database and a selection of analysis tools (such as the DACE radial velocity and transit photometry modules, and a specialised *CHEOPS* light curve analysis tool). By selecting the database a table showing all targets observed by *CHEOPS* and various properties about the object and the observations will be displayed.

In this view there are several options available to the user to customise the table. For example, the table can be sorted in an ascending or descending manner by selecting the double headed arrow alongside the name of the desired column. Furthermore, it is possible to filter the table based on criteria defined for one or more columns. These can be set either as a range of values for a specific parameter or as a string of characters that the parameter must or must not contain. Filters can be set by selecting the magnifying glass in the desired column. It should be noted that in several columns the units of the column can be changed by either clicking the units under the name of the column or in the menu that appears after clicking the magnifying glass.

Finally, it is possible to increase the maximum number of rows viewed by selecting the gear icon in the first column of the table (next to the number of rows). Moreover, new columns can be added to the table by selecting the gear icon in the last column, on the far right of the table.

This is especially important for users who wish to use the Dataset module of PYCHEOPS as it requires the file key of the observations in order to download the data from DACE and create a PYCHEOPS Dataset object. *These can be viewed by clicking on this gear icon and selecting "File Key" under "Other variables" in the drop down menu.*

4.1.2 Visualising the Data

There are a couple of visualisation tools currently available in DACE. By selecting the "Plot" tab at the top left of the *CHEOPS* database webpage, a range of stellar and observation parameters for the objects listed in the table. Note that, as the plotting tool displays the objects in the table, specific objects (or series of objects) can be plotted by filtering the table as detailed above. In the plotting tool the parameters to be plotted on the x- and y-axes can be set, with the option of setting colour and dot radius axes. Produced plots can be customised and downloaded using the right most black and white icon above the plot.

Individual light curves can be visualised by clicking the "Photometry" or "CHEOPS" icons at the end of the desired row. This loads the DACE photometry tool that shows the normalised light curve with it possible to view a section of the light curve by clicking and dragging over the desired section. By mousing over the data it is possible to see the time, flux, and flux error of individual data points. Using this tool it is possible to determine the period of the light curve using the Box Least Squares (BLS) method with the option to phase fold the data on this period. Finally, it is possible to compute a light curve model and then fit it to the data via two methods (Nelder-Mead and BFGS) with statistics about the fit provided to the left of the plot. Produced plots can be customised and downloaded using the right most black and white icon above the plot.

4.1.3 Downloading the Data

In order to download data from the DACE website users can select a row in the table, that turns the background colour of the row light green, and click one of the black and white icons in the top right of the webpage named: "Light curves", "Images", "Logs", and "All data products". Files for multiple objects can be download concurrently by selecting multiple rows. By choosing "All data products", all light curve, image, and log files outlined below will be downloaded along with several of the raw and calibrated files used and generated during the data reduction process (DRP). Should a download request fail, users should fill out a bug report.

4.2 Using the DACE Python-Based API

The DACE Python-based API can be used to query the database and to download data from the archive. This is done by using the DACE Python package that can be installed using:

```
pip install --extra-index-url https://dace.unige.ch/api python-dace-client
```

and can be updated by adding `--upgrade` to the command:

```
pip install --extra-index-url https://dace.unige.ch/api python-dace-client
--upgrade
```

However, before utilising the package an authentication key must be generated in order to access the *CHEOPS* data on DACE. Users can generate a new key by going to their DACE profile page at https://dace.unige.ch/user/?tab=profile and clicking on the black and white "Generate a new API key" icon in the DACE API section in the centre of the page.

Users should then create a .dacerc file in their home directory with the following lines:

```
[user]
key = apiKey:*Your API key here*
```

with the newly generated key following the colon in the second line. Following these steps users will be able to query and download from DACE. It should be noted that as the Dataset module of PYCHEOPS uses the DACE package to download *CHEOPS* data, if users wish to use this functionality of PYCHEOPS then they should also generate an API key and create a .dacerc file. It should be noted that PYCHEOPS works with both version 1 and 2 of the DACE API.

4.2.1 Querying the Database

Prior to downloading *CHEOPS* data using DACE users may which to query the database. This can be useful not only for finding datasets, but also for obtaining the file key of a dataset which is needed for downloading data from DACE using the Python API. Users can query the entire database with:

```
from dace.cheops import Cheops
data = Cheops.query_database()
```

This will return a dictionary with keys and values for each light curve in the database. Users can see the dictionary keys using:

```
data.keys()
```

These keys can therefore be used to return the values the users desire. For example, the file key of the first object in the query can be found by:

```
data["file_key"][0]
```

This is especially important to note for users who wish to download data from DACE using the PYCHEOPS Dataset module as is outlined below.

It is possible to sort the output of the query in an ascending (asc) or descending (desc) manner for the specified keyword. For example, to sort by the object ID in the catalogue:

```
data = Cheops.query_database(sort={"obj_id_catname":"asc"})
```

In addition to a simple query returning the entire catalogue, using the aforementioned keys users can create filters that are given as arguments to the `query_database` function. These filters are based on the data types of the values. For keys with int and double type values, users can set minimum and maximum bounds:

```
myfilter = {"obj_mag_cheops":{"min":6.0, "max":14.0}}
```

For keys with string type values, users can filter based on if the value contains or does not contain a specific string, or if the value is an empty string or not:

```
myfilter = {"obj_id_catname":{"contains":"WASP", "notContains":"TYC",
             "empty":False}}
```

Finally, for keys with a Boolean type value, users can choose to only return values that are True or are False:

```
myfilter = {"status_published":{"is":true}}
```

Of course these filters can be combined and then used in the `query_database` function:

```
myfilter = {"obj_mag_cheops":{"min":6.0, "max":14.0},
             "obj_id_catname":{"contains":"WASP",
             "notContains":"TYC", "empty":False},
             "status_published":{"is":true}}
data = Cheops.query_database(filters=myfilter)
```

A more detailed description for this function can be seen by running:

```
help(Cheops.query_database)
```

4.2.2 Downloading the Data

It is also possible to download *CHEOPS* data using the DACE Python API. Users should define the data file type ("all", "lightcurves", "images", or "logs"; see below for a description of the different file types), the directory and file name where the data will be downloaded to, and any filters the user wants to use (as per the examples given above). The following will download all images that meet the conditions of the previously defined filter into "/home/user/cheops-data.tgz":

```
from dace.cheops import Cheops
Cheops.download(file_type="images", filters=myfilter,
                output_full_file_path="/home/user/cheops-data.tgz")
```

If the Cheops.download command results in a HTTP error then your account access needs to be updated and verified. Users should submit a bug report on the DACE website stating that this has occurred. Furthermore, the DACE package includes the functionality to download the light curve of a target in the form of a Python dictionary using the `get_lightcurve` function:

```
target_lightcurve = Cheops.get_lightcurve(target=target, aperture=aperture)
```

Where the `aperture` argument can be set to "default", "optimal", "rinf", or "rsup". These correspond to different aperture radii used during the photometry conducted to produce the light curve, as detailed below. The returned dictionary contains the bjd date, flux, flux error, and x and y centroids of the observations. Note, that this is different to downloading the light curves using the `download` function above, as that method returns light curves in fits file tables.

A more detailed description for both these functions can be seen by running:

```
help(Cheops.download)
```

and

```
help(Cheops.get_lightcurve)
```

Users can also utilise the Dataset module of PYCHEOPS to download data from DACE using the following:

```
from pycheops import Dataset
D = Dataset(file_key)
```

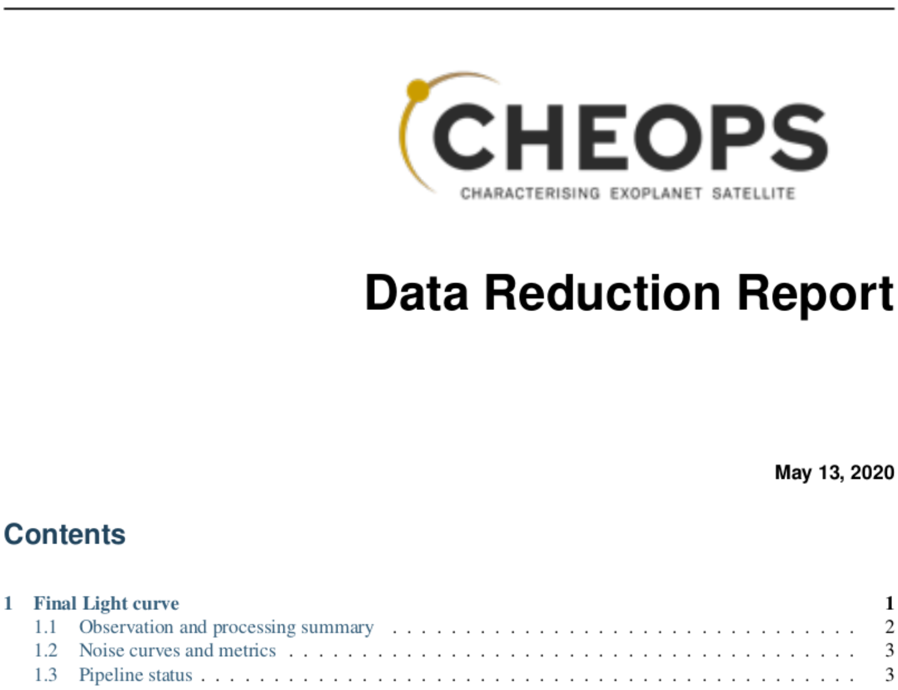
Where the "file_key" for a given light curve can be found either by viewing the data on the DACE web interface or by querying the database using the Python API as has been detailed above. This method will download all data types for the given file key, by default, into the directory provided as the `data_cache_path` when PYCHEOPS was installed. Note that this directory can be changed in the pycheops.cfg file usually found in the user's home directory. If the data have previously been download the above command will use the local files, instead of re-downloading them. Furthermore, as this method take a specific file key as an input only one set of observations are downloaded at a time and therefore, multiple Dataset objects need to be created if the user wants to download and use multiple sets of observations.

The main benefit of using PYCHEOPS and the example above to download data from DACE is that the data is returned a Dataset object that can then be manipulated using other PYCHEOPS functions, such as those in the "Useful Recipes" section below.

Additionally, if users download *CHEOPS* data using the PYCHEOPS dataset object then a PDF of the DRP report, described below, is automatically shown. *It is strongly advised that users read this document in order to assess the quality of their data and check for potential issues that need to be accounted for during decorrelation.* If users are updating their version of PYCHEOPS they might have to run the `setup_config` code again in order to specific which PDF viewer program they want to use:

```
from pycheops.core import setup_config
setup_config()
```

For example, when prompted, Ubuntu users may wish to input: `evince {} &`, where the {} is a placeholder for file names to be used by the PyCheops Dataset object. The beginning of a DRP report is shown for identification:



CHEOPS
CHARACTERISING EXOPLANET SATELLITE

Data Reduction Report

May 13, 2020

Contents

Figure 1: The top of the front page of an example DRP report.

4.3 Downloadable Data Products

Here the main data products users can download using the processes outlined above are discussed. As mentioned previously, by downloading all files for a target, raw and calibrated files used and generated during the DRP will also be downloaded in addition to the files outline below. For a detailed overview of all files, users should consult the *CHEOPS* Observers Manual [5], that can be found at: https://www.cosmos.esa.int/web/cheops-guest-observers-programme/ao-1.

4.3.1 Light Curves

By selecting to download the *CHEOPS* light curves DACE provides the users with four fits files for each series of observations selected (i.e. each row in the table). These light curve fits files are the products of the DRP, as reported in the corresponding log, with each file the result of aperture photometry conducted on the calibrated science images using different aperture radii. The DRP and aperture radii are detailed and reported in Hoyer et al. 2019 [6]. These radii are default ('DEFAULT', $r_{\rm ap} = 25$ pixels = 25 arcsec), inferior ('RINF', $r_{\rm ap} = 22.5$ pixels = 22.5 arcsec), superior ('RSUP', $r_{\rm ap}$ = 30 pixels = 30 arcsec), and optimal ('OPTIMAL', determined during each visit taking into account factors such as target brightness and nearby sources).

4.3.2 Images

If users want to download the observations of a target in order to check potential problems with the data or to conduct their own photometry they should select this option (or "All data products"). A data cube of all calibrated and corrected subarray frames (a section of the full frame array centred on the target with size 200×200 pixels) will be downloaded as a fits file. Additionally, an imagette (a 50×50 pixel region of the full frame array centred on the target) data cube of the raw observations is also downloaded. A representative frame of the observations in the raw, calibrated, and corrected states, along with light curves for the raw, calibrated, and corrected data, can be found in Section 2 ("Summary of the processing stages") in the corresponding observing and data reduction log.

4.3.3 Logs

Downloading the observing and data reduction log provides the user with PDF of the Data Reduction Report for that light curve observation. This report gives a summary of the observations conducted by *CHEOPS* and details the processes undertaken during the DRP, such as bias, dark, and flat field correction, bad pixel and background correction, and aperture optimisation and photometry. As the reports provide detailed information on the DRP users should consult them.

5 Useful Recipes

As described above, PYCHEOPS can be used to download *CHEOPS* data from DACE. Moreover, PYCHEOPS also has significant functionality for the analysis of *CHEOPS* light curves. Below several basic analysis recipes are given that users may find useful. It should be noted that there are currently multiple example Jupyter notebooks covering much of the functionality of PYCHEOPS in the `examples/Notebooks/` folder of your PYCHEOPS installation directory.

Prior to discussing the functionality of PYCHEOPS it is worth highlighting that the inline documentation of all functions can be viewed, for example for the `Dataset` class, by:

```
help(pycheops.Dataset)
```

To help guide users of this document, the recipes below will use the *CHEOPS* visit of KELT-11 b as an example. This dataset can be downloaded using:

```
from pycheops import Dataset

file_key = "CH_PR300024_TG000101_V0100"
```

```
D = Dataset(file_key)
```

5.1 Getting the Data and Plotting the Light Curves

There are two main methods users can utilise to plot their data; using the `get_lightcurve` function in the DACE package and using the Dataset class in PYCHEOPS. Provided here are code snippets that produce the light curve plot presented below for KELT-11 b using the OPTIMAL aperture. *It should be noted that in the following examples* `Cheops.get_lightcurve` *and* `D.get_lightcurve` *are different methods and users should be careful to use the desired function. The former comes from the DACE package, whereas the later comes from* PYCHEOPS.

5.1.1 DACE Method

```
from dace.cheops import Cheops

target = "KELT11"
aperture = "OPTIMAL"
target_lightcurve = Cheops.get_lightcurve(target=target, aperture=aperture)

time = np.array(target_lightcurve["obj_date_bjd_vect"])
flux = np.array(target_lightcurve["photom_flux_vect"])
flux_err = np.array(target_lightcurve["photom_flux_vect_err"])
plt.plot(time, flux, "k.")
plt.title(target + " - aperture = " + aperture)
plt.xlabel("BJD Date (d)")
plt.ylabel("Flux")
```

5.1.2 PYCHEOPS Method

```
from pycheops import Dataset

file_key = "CH_PR300024_TG000101_V0100"
D = Dataset(file_key)

aperture = "OPTIMAL"
time, flux, flux_err = D.get_lightcurve(aperture = aperture,
                                        decontaminate = True)
plt.plot(time, flux, "k.")
plt.title(D.target + " - aperture = " + aperture)
plt.xlabel("BJD Date (d)")
plt.ylabel("Normalised Flux")
```

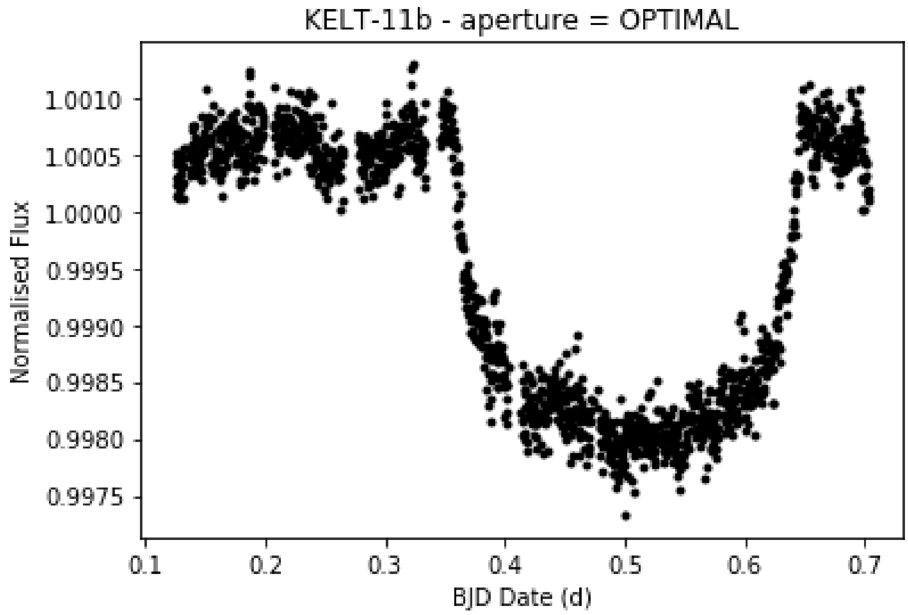


Figure 2: Plot of the light curve of KELT 11-b produced using the "OPTIMAL" aperture generated using both DACE and PYCHEOPS code snippets above.

The DACE `get_lightcurve` function provides a straightforward method to download and plot the bjd date, flux, flux error, and x and y centroids, and can be useful as a quick look. However, using the PYCHEOPS Dataset class may be useful for a more detailed analysis as it includes methods that include a basic sigma clipping of the light curve (using the `reject_highpoints` argument) and the removal of potentially inaccurate data flagged during the DRP (for example, observations taken when *CHEOPS* passes through the South Atlantic Anomaly). The PYCHEOPS `get_lightcurve` function can also perform a subtraction of the contamination from nearby sources that might have affected the photometry of the target star via the `decontaminate` argument, which needs to be set as either True or False by the user upon importing data. This decision is left to the user as for some datasets removing the contamination may degrade the light curve quality, and therefore, users are advised to assess their data both with `decontaminate` set equal to True and False. The PYCHEOPS `get_lightcurve` function also prints information about the retrieved dataset such as visit duration and efficiency, the median background level, and contamination, smearing, and ramp estimates. Furthermore, the Dataset class downloads all data for a given visit by default and subsequently builds a dictionary from the headers of the FITS files. This provides the user with more information on the observations, such as roll angle, x and y centroid offsets, and background and contamination values, that might be useful in further analysis. In the above example these arrays can be returned by running `D.lc["roll_angle"]` after the `D.get_lightcurve()` command, for example. Moreover, the metadata of the observations are stored as an `astropy` table within the `Dataset` object and can be viewed with `D.metadata`.

5.2 Visualising Subarrays and Imagettes

In addition to plotting light curves, users may wish to visualise the subarrays or imagettes of the observations, especially in cases when the light curve may be contaminated by a nearby source. The subarrays and imagettes are 200×200 and 50×50 pixel cutouts of the full frame array centred on the target, as described above in Section 4.3. Animations of both sets of images can be produced and displayed using the `animate_frames` PYCHEOPS function as shown below. By default every tenth frame is displayed, however users can set how often images are included in the animation with the `nframes` argument, as well as overplotting a grid and the aperture of the observations. One benefit of this function is that the produced animations are also saved in the current working directory.

5.2.1 Subarrays

```
from pycheops import Dataset

file_key = "CH_PR300024_TG000101_V0100"
D = Dataset(file_key)
aperture = "OPTIMAL"
time, flux, flux_err = D.get_lightcurve(aperture = aperture,
                                        decontaminate = True)
Nth_frame = 10
Min_value_scaling_factor = 1.0
Max_value_scaling_factor = 1.0
frames = D.animate_frames(nframes = Nth_frame,
                          vmin = Min_value_scaling_factor,
                          vmax = Max_value_scaling_factor,
                          subarray = True, grid = True,
                          aperture = aperture)
```

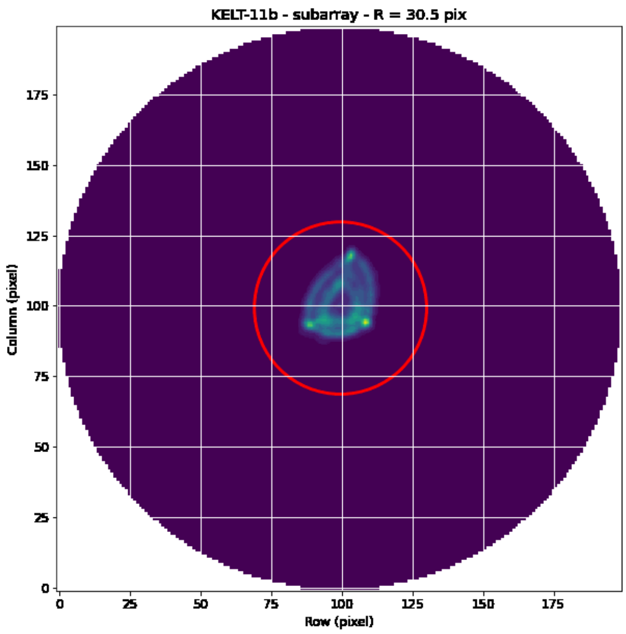


Figure 3: A frame of the animation of subarrays taken during observations of KELT 11-b produced using the code snippet above.

5.2.2 Imagettes

```
from pycheops import Dataset

file_key = "CH_PR300024_TG000101_V0100"
D = Dataset(file_key)
aperture = "OPTIMAL"
time, flux, flux_err = D.get_lightcurve(aperture = aperture,
                                        decontaminate = True)
Nth_frame = 10
Min_value_scaling_factor = 1.0
Max_value_scaling_factor = 1.0
frames = D.animate_frames(nframes = Nth_frame,
                          vmin = Min_value_scaling_factor,
                          vmax = Max_value_scaling_factor,
                          imagette = True, grid = True,
                          aperture = aperture)
```

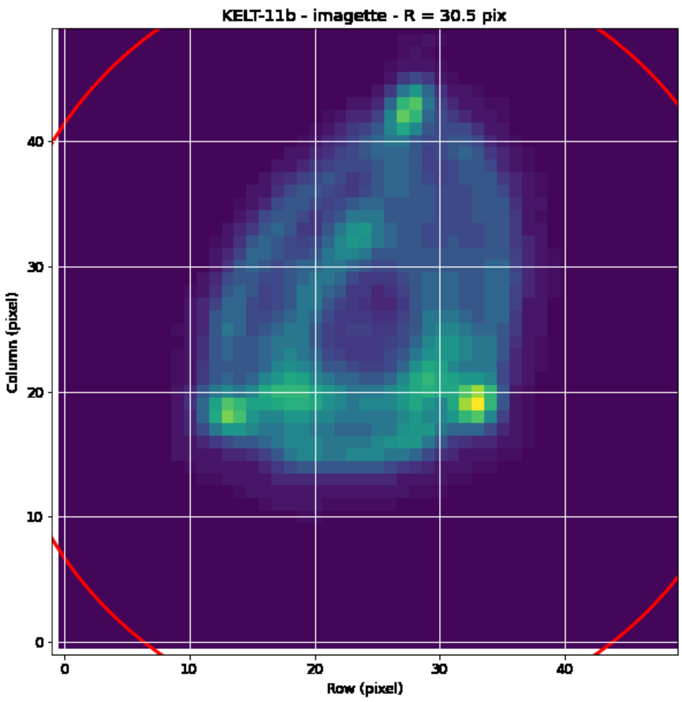


Figure 4: A frame of the animation of imagettes taken during observations of KELT 11-b produced using the code snippet above.

It is worth noting that in both of the above example only every 10th frame is included in the animated data cube in order to avoid memory issues.

5.3 Preparing your Data

After visualising your data users may want to prepare the dataset for fitting using various methods such as clipping outliers, decorrelating and detrending the light curve, or flattening sections of the dataset. This can be done by utilising some of the functionality of PYCHEOPS outlined below. In addition to viewing the DRP report when the data are downloaded it is also possible to show the report using a stand-alone function:

```
from pycheops import Dataset

file_key = "CH_PR300024_TG000101_V0100"
D = Dataset(file_key)
D.view_report(pdf_cmd = "evince {} &")
```

Where the `pdf_cmd` argument specifies the PDF viewer to use, for the example above the Evince program is used. *It is recommended that users consult the DRP report before proceeding to decorrelate and fit their data as there might be issues identified in the report that could assist in data analysis.*

5.3.1 Clipping Outliers in the Dataset

Should users notice outliers in the dataset that they want to remove the stand alone outlier clipping routine in PYCHEOPS can be utilised. The `clip_outliers` function removes outliers from a dataset by calculating the mean absolute deviation (MAD) from the light curve following median smoothing, and rejects data greater than the smoothed dataset plus the MAD multiplied by a clipping factor, by default equal to five.

The smoothing of the light curve is done in sections with the window width set to be smaller enough (11 data points by default) that this clipping should be able to remove outliers in transit or eclipses. The `clip_outliers` function returns the clipped time, flux, and flux error arrays with an example shown below:

```
from pycheops import Dataset

file_key = "CH_PR300024_TG000101_V0100"
D = Dataset(file_key)
aperture = "OPTIMAL"
time, flux, flux_err = D.get_lightcurve(aperture = aperture,
                                        decontaminate = True)
clipping_factor_value = 5
time, flux, flux_err = D.clip_outliers(clip = clipping_factor_value)
```

5.3.2 How to Decorrelate your Dataset - Diagnosing the Issue

For decorrelating a dataset, a useful first step is to run the `diagnostic_plot` function that produces a series of ten plots, such as flux versus time, flux versus *CHEOPS* roll angle, flux versus x and y centroids, flux versus background, flux versus contamination, and flux versus smear, that may be useful in determining the cause of the unwanted trend. It should be noted that this function can be run before or after decorrelation, therefore allowing users to view the effects of the decorrelation on their data. The following plot can be produced using:

```
from pycheops import Dataset

file_key = "CH_PR300024_TG000101_V0100"
D = Dataset(file_key)
aperture = "OPTIMAL"
time, flux, flux_err = D.get_lightcurve(aperture = aperture,
                                         decontaminate = True)
D.diagnostic_plot()
```

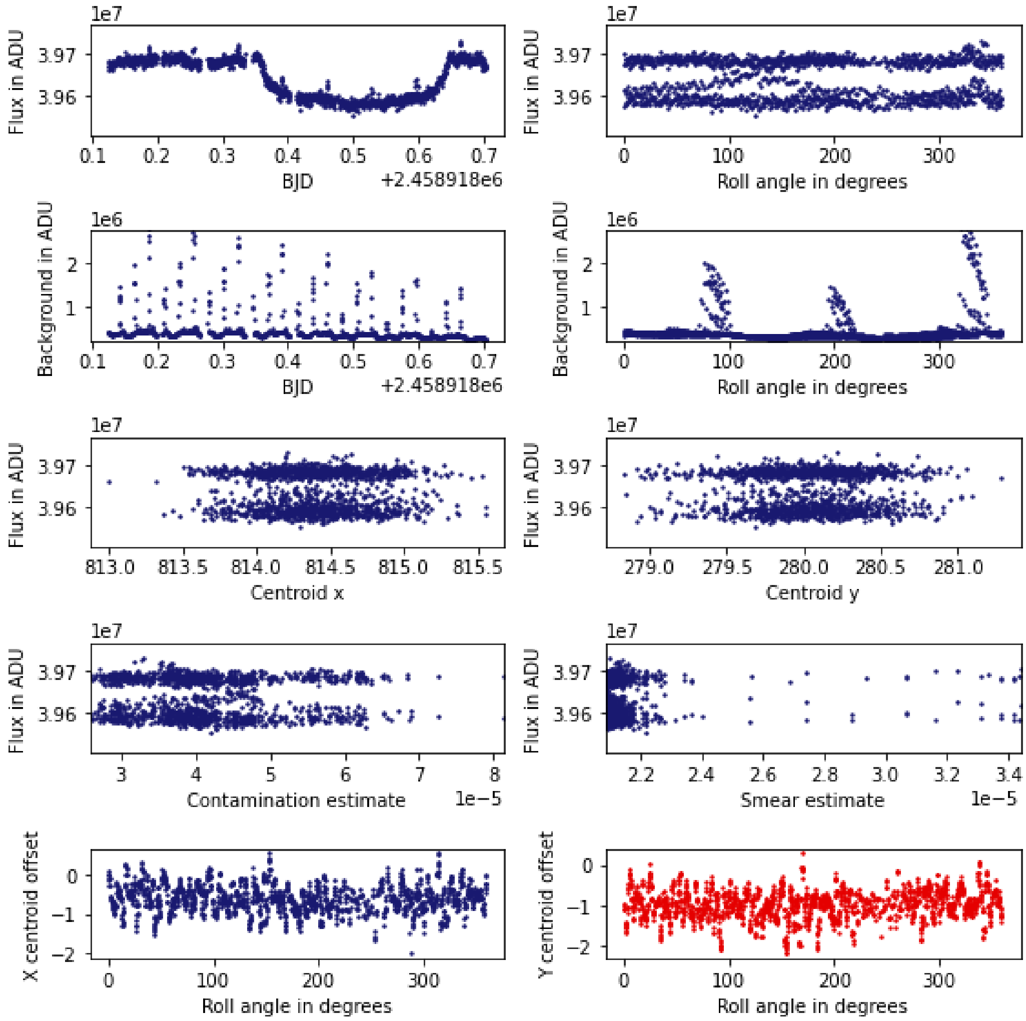


Figure 5: A diagnostic plot of the *CHEOPS* observations of KELT-11. A description of these subplots is given in the text.

The example series of plots shown above is typical of a light curve containing a transit, as can be seen in the first row, left plot. The flux versus roll angle, and flux versus x and y centroids plots in the first and fourth rows show two bands of fluxes from the out of- and in-transit data, as does the flux versus contamination estimate plot in the left plot of the third row. The background flux plots in the second row are also somewhat typical with the peaks in background flux indicative of the target passing close to the limb of the Earth.

In general, users should look at the first and fourth rows of the plots in order to assess if and how the flux is changing against roll angle or x and y centroids. However, the background flux plots in the second row, and the flux versus contamination estimate and the flux versus smear estimate in the last row may also be useful to check to evaluate if there are any irregularities caused by, for example, contamination from a nearby source. These can be decorrelated against using the keywords presented below.

If instead, the flux versus roll angle, and flux versus x and y centroids plots have a sinusoidal or linear trend, as can be seen in the example plots, then decorrelation is needed.

Another good check to perform if users notice any periodic flux trend in the light curve is a calculation of the separation between the target and various Solar System bodies to assess if stray light from these objects could affect the observations. This can be done in PYCHEOPS using the `planet_check` function that prints out the coordinates and separation to the Moon, Mars, Jupiter, Saturn, Uranus, and Neptune:

```
from pycheops import Dataset

file_key = "CH_PR300024_TG000101_V0100"
D = Dataset(file_key)
D.planet_check()
```

```
BJD = 2458918.124880498
Body     R.A.         Declination  Sep(deg)
-------------------------------------------
Moon     11:21:44.78  +09:19:29.6     16.0
Mars     19:05:24.57  -23:08:38.0     81.6
Jupiter  19:29:58.33  -21:50:58.1    113.6
Saturn   20:03:17.21  -20:27:01.5    125.8
Uranus   02:06:47.07  +12:20:30.2    129.0
Neptune  23:18:02.79  -05:36:58.0    163.1
```

Figure 6: Example output of the `planet_check` function showing the BJD of the observations, and the coordinates of various Solar System bodies, and the separation between the target and them.

5.3.3 How to Decorrelate your Dataset - Performing the Decorrelation

The main PYCHEOPS decorrelation method is the `decorr` function in the Dataset module that fit trend models to the *CHEOPS* light curve using routines in the `lmfit` package. Using this function it is possible to model first, second or third order trends in the flux over time, x or y centroid, roll angle, background, contamination, or smear by setting various keyword arguments to True. Below is a list of the trend to be decorrelated and the corresponding keyword arguments:

- flux versus time - `dfdt`, `d2fdt2`
- flux versus x centroid - `dfdx`, `d2fdx2`

- flux versus y centroid - `dfdy`, `d2fdy2`
- flux versus roll angle - `dfdsinphi`, `dfdcosphi`, `dfdsin2phi`, `dfdcos2phi`, `dfdsin3phi`, `dfdcos3phi`
- flux versus background - `dfdbg`
- flux versus contamination - `dfdcontam`
- flux versus smear - `dfdsmear`

Therefore, it is possible to do a linear decorrelation of a first order trend in the flux against roll angle using:

```
from pycheops import Dataset

file_key = "CH_PR300024_TG000101_V0100"
D = Dataset(file_key)
aperture = "OPTIMAL"
time, flux, flux_err = D.get_lightcurve(aperture = aperture,
                                        decontaminate = True)
time, flux, flux_err = D.decorr(dfdsinphi = True, dfdcosphi = True)
D.diagnostic_plot()
```

By running the `diagnostic_plot` function after the decorrelation, the decorrelated flux should be plotted allowing users to determine if further decorrelation is needed. It should be noted that as `sinphi` and `cosphi` are simply the sine and cosine of the roll angle, if users wish to decorrelate against the roll angle then both the `sinphi` and `cosphi` keyword arguments should be set to True.

5.3.3.1 Removing Glint

As mentioned previously it has been found that periodic flux trends have been observed that correspond to ranges of *CHEOPS* roll angles specific to that observation. An origin of these flux trends could be stray light or "glint" from nearby source such as the Moon or a bright neighbour. If such a trend is found using the `rollangle_plot` function or if the separation to a Solar System object is relatively small as seen via the `planet_check` function then users can attempt to model this artefact using the `add_glint` function.

This method creates a spline function, with the number of splines used to be input by the user, to be used to fit flux artefacts as a function of roll angle. By default, this is done by fitting the residuals of an eclipse or transit fit with the spline model in order to not remove the eclipse or transit. To do this the `lmfit_eclipse` or `lmfit_transit` functions should be run first, as is seen in the code snippet below. However, if a user parses an array to the `mask` argument, for example covering an eclipse or transit, then `add_glint` can be used to model the unmasked out of eclipse/transit fluxes instead of the fit residuals. Both approaches seem to perform equally well in removing glints. Lastly, if it was found that the target to Moon separation is small it could be useful to model the fluxes or residuals as a function of roll angle relative to the Moon. This can be done by setting the `moon` argument equal to True.

After the glint model as been built using `add_glint` it can be used during an eclipse or transit fit to model and remove the flux artefact, as can be seen below, with `glint_scale` used as a scaling factor to fit the artefact:

```
from pycheops import Dataset, StarProperties

file_key = "CH_PR300024_TG000101_V0100"
D = Dataset(file_key)
aperture = "OPTIMAL"
time, flux, flux_err = D.get_lightcurve(aperture = aperture,
                                                decontaminate = True)
Period_value = 4.736529
Period_error = 0.000068

host_star_properties = StarProperties(D.target)
Log_stellar_density = host_star_properties.logrho

result = D.lmfit_transit(P = ufloat(Period_value, Period_error),
                            logrhoprior = Log_stellar_density)
N_spline = 30
glint = D.add_glint(nspline = N_spline, moon = True)
result = D.lmfit_transit(P = ufloat(Period_value, Period_error),
                            logrhoprior = Log_stellar_density,
                            glint_scale = (0.,2))
```

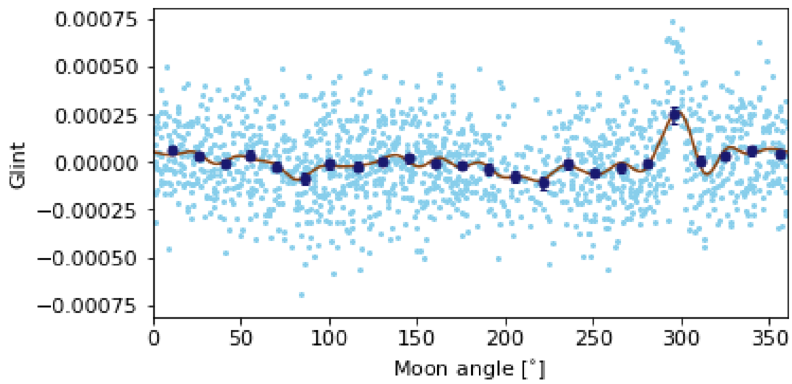


Figure 7: Example roll angle plot produced by the `add_glint` function showing the spline model fit (brown solid line) to the glint flux artefact seen in the data (blue).

5.3.3.2 Removing Ramp

In multiple datasets observed to date, a ramp at beginning of the visits has been seen that has a characteristic increase or decrease in fluxes with a decay timescale of several *CHEOPS* orbits. It has been found that this flux variation is due to PSF shape changes on the order of subpixels, that is caused by changes in the telescope focus because of a shift in the telescope tube temperature due to a change in the thermal load on *CHEOPS*. Thus, this flux ramp can be corrected using the temperature metadata. PYCHEOPS users can utilise the `correct_ramp` function to correct the measured fluxes based on the aperture radius used and the temperature via:

```
from pycheops import Dataset

file_key = "CH_PR300024_TG000101_V0100"
D = Dataset(file_key)
aperture = "OPTIMAL"
time, flux, flux_err = D.get_lightcurve(aperture = aperture,
                                        decontaminate = True)
time, flux, flux_err = D.correct_ramp(plot = True)
```

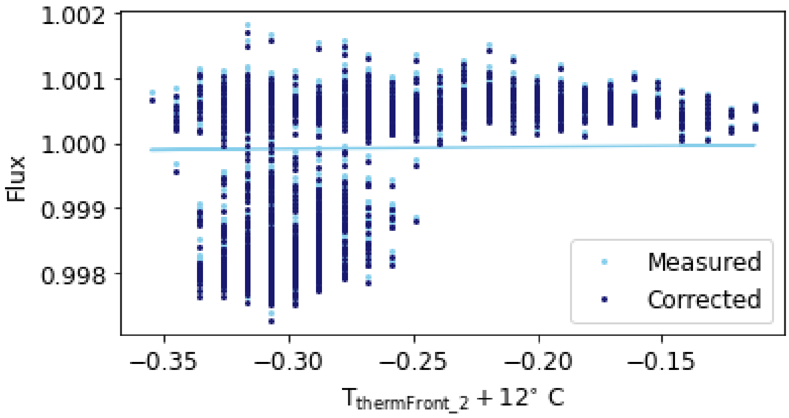


Figure 8: Plot showing the measured fluxes (light blue) and corrected data (dark blue) against the telescope temperature at time of observation for the KELT-11 dataset.

It should be noted that both the returned fluxes and the values stored within the `Dataset` object are corrected.

5.3.4 How to Decorrelate your Dataset - To Decorrelate or not to Decorrelate

It might not always be intuitive to determine if the decorrelation of a light curve is necessary. If this is the case, then users may wish to use the `should_I_decorr` method. This function runs the `decorr` function for all basis vectors listed above and calculates the Bayesian Information Criteria (BIC) of each decorrelation combination. This can be run using:

```
from pycheops import Dataset

file_key = "CH_PR300024_TG000101_V0100"
D = Dataset(file_key)
aperture = "OPTIMAL"
time, flux, flux_err = D.get_lightcurve(aperture = aperture,
                                        decontaminate = True)
mask_centre_value = 0.5
mask_width_value = 0.2
D.should_I_decorr(mask_centre_value, mask_width_value)
```

The function will report whether or not decorrelation of the light curve is necessary and what trend or trends should be decorrelated. This is done under the assumption that the combination of decorrelation basis vectors that produces the lowest BIC value describes the modelling of any observed trends the best and therefore should be decorrelated against to remove these trends. Users should then use the `decorr` function or the decorrelation functionality of the `lmfit_eclipse` or `lmfit_transit` functions if fitting of an observed feature is desired.

In an identical manner to the `flatten` function described below, it is also possible to mask sections of the light curve that should not be included in the decorrelation combination tests. As seen in the code snippet above, this is done by providing a mask centre time and width with all data inside this window excluded. This feature is potentially useful if there are eclipses or transits in the dataset that might affect trend fitting done, for example by resulting in an apparent large trend in flux against time.

5.3.5 Flattening or Masking Sections of the Light Curve

If, upon inspection of the diagnostic plot, users notice that sections of the light curve need to be re-normalised the `flatten` function can be utilised. This routine fits the dataset with a polynomial which is then used in the normalisation. Importantly, it is possible to mask specified regions of the light curve from re-normalisation, for example an eclipse or transit, by providing mask centre and width values as shown below:

```
from pycheops import Dataset

file_key = "CH_PR300024_TG000101_V0100"
D = Dataset(file_key)
aperture = "OPTIMAL"
time, flux, flux_err = D.get_lightcurve(aperture = aperture,
                                        decontaminate = True)
mask_centre_value = 0.5
mask_width_value = 0.2
time, flux, flux_err = D.flatten(mask_centre_value, mask_width_value)
```

However, if is it found that sections of the light curve contain noisy data that either cannot be corrected for via decorrelation or should not be flatten for fitting, then it is possible to mask these data out in order to avoid fitting it using the functions below. This can be done in PYCHEOPS using the `mask_data` function by parsing an input Boolean array of True or False values indicating which data should be masked. In the example, data with `time < 0.3` are masked out.

```
from pycheops import Dataset

file_key = "CH_PR300024_TG000101_V0100"
D = Dataset(file_key)
aperture = "OPTIMAL"
time, flux, flux_err = D.get_lightcurve(aperture = aperture,
                                        decontaminate = True)
mask_Boolean_array = time < 0.3
time, flux, flux_err = D.mask_data(mask_Boolean_array)
```

5.4 Fitting your Data - A Single Visit

Once *CHEOPS* data has been downloaded, visualised, and detrended (if needed) PYCHEOPS can be utilised to fit transits or eclipses of observed exoplanets in order to determine physical and orbital parameters of these objects. In addition to a model used to detrend light curves, the Models module of PYCHEOPS also includes transit and eclipse models that can be fit to the data using the `lmfit` or `emcee` packages, as will be detailed below.

In addition to deriving the values of physical and orbital parameters via transit or eclipse fitting, it is possible to set priors when building the model to be fit. Parameters initial values can be input as a simple Python float or a ufloat float with an uncertainty, a tuple that includes upper and lower bounds to a range of values, or a `lmfit` Parameter object. If a tuple is parsed and a third, mid point is provided this is taken as the initial value in the fit. Examples of each of these input types are shown below in the `lmfit` examples. Finally, as well as conducting decorrelation of a light curve separately, as outlined above in Section 5.3, it is also possible to perform the decorrelation method using the same keyword arguments listed above prior to fitting the transit by setting a range of values to search for trends over in `lmfit_transit` and the detrend keyword equal to True in `plot_lmfit`, as seen below.

It has been seen that, due to the nature of the orbit of *CHEOPS*, it is possible that periodic flux artefacts may be apparent in the data corresponding to values or ranges of *CHEOPS* roll angles. To assist in identification of these periodic trends PYCHEOPS has functionality to plot the roll angle against flux residuals from a prior fit, for example using the `lmfit_transit` function outlined below, using the `rollangle_plot` function. If any decorrelation against roll angle using the glint module has been done, the fit of this model to the data will also be shown. By setting the `binwidth` argument the dataset can be binned in widths with units of degrees. It should be noted that the `rollangle_plot` function detects the gaps in the light curve and shifts the data, such that the data appears as a continuous plot, as can be seen below:

```
from pycheops import Dataset

file_key = "CH_PR300024_TG000101_V0100"
D = Dataset(file_key)
aperture = "OPTIMAL"
time, flux, flux_err = D.get_lightcurve(aperture = aperture,
                                         decontaminate = True)
bin_width_value = 15.0
D.rollangle_plot(binwidth = bin_width_value)
```

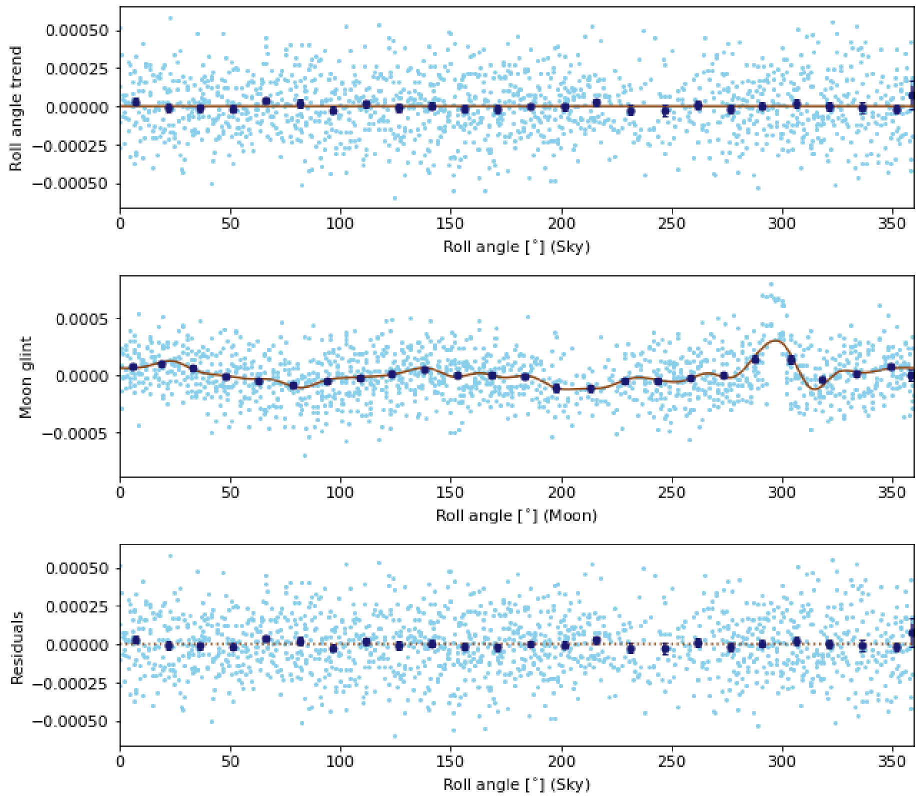


Figure 9: Example of a roll angle plot showing the roll angle against trend (top), moon angle against glint (middle), and roll angle against the residuals of a transit fit (bottom). The light blue data points are the complete dataset with the dark blue points showing the data binned every 15 degrees.

Importantly, the `rollangle_plot` function currently only works after one of the fitting routines, such as `lmfit_transit` or `lmfit_eclipse`, has been run as this function plots the residuals to the fit against the roll angle.

5.4.1 Obtaining Stellar Parameters of the Host Star

Before fitting your data with either a transit, eclipse, or phase curve model it can be useful to obtain various parameters about the target host star, such as stellar density or limb-darkening coefficients. Currently, PYCHEOPS includes two options that users may want to utilise to acquire such information.

Firstly, if the power-2 limb-darkening coefficients, h_1 and h_2, of a transit are not known prior to fitting they can be estimated from stellar parameters ($T_{\rm eff}$, log(g), Fe/H) of the host star and the Stagger grid [8]. This functionality is included in the PYCHEOPS ld module and is shown below:

```
from pycheops.ld import stagger_power2_interpolator

Teff_value = 5414
logg_value = 3.73
FeH_value = 0.25
```

```
power2_ld_coefficients = stagger_power2_interpolator()
c, alpha, h_1, h_2 = power2_ld_coefficients(Teff_value, logg_value, FeH_value)
```

However, if the stellar parameters are also not known users can query the SWEET-Cat database [13] to potentially retrieve them by creating a `StarProperties` object. Should the target of choice be in the SWEET-Cat database this functionality is very useful as it returns the stellar parameters ($T_{\rm eff}$, log(g), Fe/H), an estimate of the stellar density from log(g) [10], and the power-2 limb-darkening coefficients, h_1 and h_2, calculated using the Stagger grid interpolation shown above. For stars with stellar parameters outside of this grid [8], values are interpolated from results of limb-darkening analyses of ATLAS and PHOENIX models [4]. By inputting the target name, the properties of the host star can be obtained by:

```
from pycheops import Dataset, StarProperties

file_key = "CH_PR300024_TG000101_V0100"
D = Dataset(file_key)
aperture = "OPTIMAL"
time, flux, flux_err = D.get_lightcurve(aperture = aperture,
                                        decontaminate = True)
host_star_properties = StarProperties(D.target)
Log_stellar_density = host_star_properties.logrho
h_1_value = host_star_properties.h_1.n
h_2_value = host_star_properties.h_2.n
```

```
Identifier : KELT-11
Coordinates: 10:46:49.74 -09:23:56.5
T_eff :  5370 +/- 50 K   [SWEET-Cat]
log g :  3.73 +/- 0.04    [SWEET-Cat]
[M/H] : +0.18 +/- 0.07    [SWEET-Cat]
log rho : -1.17 +/- 0.08  (solar units)
h_1 : 0.715 +/- 0.011     [Stagger]
h_2 : 0.442 +/- 0.050     [Stagger]
```

Figure 10: Example output of the `StarProperties` class showing the target name and coordinates, along with the stellar properties retrieved from SWEET-Cat and the derived stellar density and limb-darkening coefficients.

In the above example, the value and uncertainty of the logarithm of the stellar density are returned in the `Log_stellar_density` variable, along with the nominal values of the limb-darkening coefficients, h_1 and h_2, without any previously reported uncertainties. As this function queries both SIMBAD and SWEET-Cat it is recommended that users check the reported coordinates to make sure that the properties for the correct target are returned. Furthermore, if the target has high proper motion, users are advised to increase the `match_arcsec` argument of `StarProperties` in order to increase the search radius in SIMBAD and SWEET-Cat, and thus find the target.

In addition, users can also query the stellar parameters table hosted at DACE by setting the `dace` keyword of `StarProperties` equal to True.

If users already know the stellar parameters of their target, but want to use the `StarProperties` functionality to determine the stellar density and limb-darkening coefficients they can set the `match_arcsec` argument equal to `None` to avoid querying SWEET Cat, and then provide their own values to `teff`, logg, and metal arguments of StarProperties in the form of a ufloat object or a length=2 tuple containing the property value and uncertainty. These provided values will over-write any properties obtained from SWEET-Cat or DACE.

5.4.2 Obtaining Available Planetary Parameters of the Target

In order to aid in eclipse, transit, or phase-curve fitting it can be useful to set informative priors on various planetary properties. This can be done within PYCHEOPS being retrieving known values and uncertainties for a target from either TEPCat [15] or DACE using the PlanetProperties functionality in a similar manner as for StarProperties. To obtain the available planet property values users should provide the PlanetProperites class with a string of the planet identifier, and if the target is in TEPCat or DACE an object will be returned that includes the known values and uncertainties the transit centre time (T0), period (P), transit depth (depth) and width (width), and ecosw and esinw, the catalogue source of those values, and the derived eccentricity (ecc), argument of periastron (omega), and their components (f_c and f_s). The code snippet below shows how to query values for the planet KELT-11b from TEPCat and how to set the retrieved values for transit centre time (T0), period (P), transit depth (depth) and width (width) to variables:

```
from pycheops import PlanetProperties

planet_properties = PlanetProperties("KELT-11b", query_tepcat = True,
                                     query_dace = False)
Transit_centre_time = planet_properties.T0
Period_value = planet_properties.P.n
Depth_value = planet_properties.depth.n / 1.e6
Width_value = planet_properties.width.n / Period_value
```

```
TEPCat data downloaded from
 https://www.astro.keele.ac.uk/jkt/tepcat/observables.csv
Identifier : KELT-11b
T0 : 2457483.4305 +/- 0.0008 BJD      [TEPCat]
P :     4.7362083 +/- 0.0000041 days  [TEPCat]
D :  2200.0000 +/- 220.0000 ppm        [TEPCat]
W :  0.2974 +/- 0.0100 days           [TEPCat]
```

Figure 11: Output of the PlanetProperties class for KELT-11b showing the target name, along with the planet properties retrieved from TEPCat. For this target no eccentricity and argument of periastron information is found.

The transit depth and width values and uncertainties reported in PlanetProperties are in parts-per-million and days, respectively, and so in the code snippet above are converted into units used by the lmfit_eclipse and lmfit_transit functions. However, the retrieved transit centre time is the literature reported BJD time and so needs to be corrected to the epoch of the visit using the planet orbital period and bjd_ref within the Dataset object. As with StarProperties, users can provide their own values to the T0, P, depth, width, ecosw, and esinw arguments of PlanetProperties either in the form of a ufloat object or a length=2 tuple that contains the property value and uncertainty. These user values will fill in any empty values or over-write any properties obtained from TEPCat or DACE.

It should be noted that, at the moment, planet properties values can only be obtained from DACE if users are members of the *CHEOPS* Science Team. Therefore, for non-member users should set the query_tepcat and query_dace arguments to True and False, respectively, as shown in the code snippet above in order to retrieve planet property information from TEPCat.

5.4.3 Fitting a Transit

Prior to outlining the different fitting methods it is worth briefing discussing the transit model used in PYCHEOPS. Transit models are constructed using the limb-darkening described by the power-2 law [8, 9] and the following transit parameters; the orbital period (P), the transit centre time (T_0), the transit depth (D), the logarithm of the stellar density (logrhoprior), the transit width (W), the impact parameter (b), a flux scaling factor (c), the limb-darkening coefficients (h_1 and h_2), and the orbital eccentricity and longitude of periastron components (f_c and f_s), and allows for the fitting of high impact parameter, grazing transits.

It should be noted here that the transit width (W) is in units of phase. If the width is not known from a previous transit it can be calculated using the transit_width function that takes R_*/a, $k = R_p/R_*$, b, and P as inputs, where R_* and R_p are the stellar and planet radii, and a is the planetary orbital semi-major axis:

```
from pycheops.funcs import transit_width

Ra_value = 0.203
k_value = 0.052
Impact_value = 0.36
Period_value = 4.736529
Width_value = transit_width(Ra_value, k_value, Impact_value, Period_value)
```

Where the outputted width is in the same units as the inputted period.

5.4.3.1 Using lmfit

The lmfit package contains useful functionality for the fitting of models to data. This can be done by constructing a model using input priors (either a set value or a range) and allowing these values to vary when fitting the model to data via a least-square method. In PYCHEOPS this is wrapped up in the lmfit_transit function with a readable output of the parameter values produced by lmfit_report. This is shown in the following example, along with performing a decorrelation in background, and plotting the fit using plot_lmfit. In this example, parameter values are either defined below or in previous code snippets.

```
from pycheops import Dataset
from uncertainties import ufloat

file_key = "CH_PR300024_TG000101_V0100"
D = Dataset(file_key)
aperture = "OPTIMAL"
time, flux, flux_err = D.get_lightcurve(aperture = aperture,
                                                decontaminate = True)

Centre_time_value, Centre_time_error = 0.5, 0.1
Depth_value, Depth_error = 0.0027, 0.0003
```

```
Impact_min, Impact_max = 0.16, 0.49
f_c_value, f_s_value = 0., 0.
dfdbg_lower, dfdbg_upper = -1., 1.
result = D.lmfit_transit(P = ufloat(Period_value, Period_error),
                    T_0 = ufloat(Centre_time_value, Centre_time_error),
                    D = ufloat(Depth_value, Depth_error),
                    logrhoprior = Log_stellar_density,
                    W = lmfit.Parameter(value = Width_value, vary = True),
                    b = lmfit.Parameter(value = Impact_value,
                                        min = Impact_min, max = Impact_max,
                                        vary = True),
                    f_c = f_c_value, f_s = f_s_value,
                    h_1 = h_1_value, h_2 = h_2_value,
                    dfdbg = (dfdbg_lower, dfdbg_upper))
bin_width_value = 0.01
figure = D.plot_lmfit(binwidth = bin_width_value, detrend = True)
print(D.lmfit_report())
```

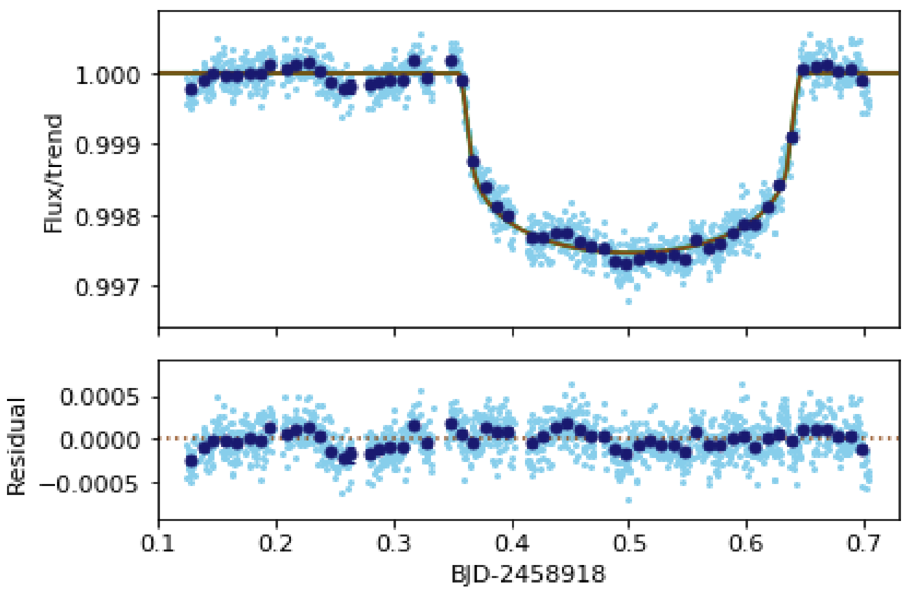


Figure 12: The *CHEOPS* light curve of a transit of KELT-11b. *Top.* The observed flux in blue with the `lmfit` fit produced by the code snippet above in orange against time. *Bottom.* The residuals of the fit.

```
[[Fit Statistics]]
    # fitting method   = leastsq
    # function evals   = 104
    # data points      = 1496
    # variables        = 7
    chi-square         = 2764.50084
    reduced chi-square = 1.85661574
    Akaike info crit   = 932.641557
    Bayesian info crit = 969.815408
    RMS residual       = 218.3 ppm
[[Variables]]
    T_0:     0.50106580 +/- 0.06584143 (13.14%) (init = 0.5004546)
    P:       4.71125684 +/- 0.13605501 (2.89%) (init = 4.736525)
    D:       0.00221491 +/- 1.3956e-05 (0.63%) (init = 0.00267)
    W:       0.06292318 +/- 0.00177522 (2.82%) (init = 0.06424541)
    b:       0.53347753 +/- 0.01089463 (2.04%) (init = 0.5)
    f_c:    -0.02262488 (fixed)
    f_s:     0.1753428 (fixed)
    h_1:     0.713 (fixed)
    h_2:     0.442 (fixed)
    c:       1.00054854 +/- 9.0341e-06 (0.00%) (init = 1)
    dfdbg:   6.7099e-04 +/- 4.3885e-05 (6.54%) (init = 0)
    k:       0.04706287 +/- 1.4827e-04 (0.32%) == 'sqrt(D)'
    aR:      4.55772954 +/- 0.12779591 (2.80%) == 'sqrt((1+k)**2-b**2)/W/pi'
    sini:    0.99312614 +/- 2.8173e-04 (0.03%) == 'sqrt(1 - (b/aR)**2)'
    logrho: -1.24234122 +/- 0.01392670 (1.12%) == 'log10(4.3275e-4*((1+k)**2-b**2)**1.5/W**3/P**2)'
    e:       0.03125700 +/- 0.00000000 (0.00%) == 'f_c**2 + f_s**2'
    q_1:     0.31136400 +/- 0.00000000 (0.00%) == '(1-h_2)**2'
    q_2:     0.48566308 +/- 0.00000000 (0.00%) == '(h_1-h_2)/(1-h_2)'
[[Correlations]] (unreported correlations are < 0.500)
    C(P, W)   = -0.998
    C(T_0, b) =  0.996
    C(T_0, D) =  0.612
    C(D, b)   =  0.604
    C(D, c)   =  0.515
[[Priors]]
    P:      4.73652500 +/- 0.10000000
    dfdbg:  0.00000000 +/- 1.00000000
    logrho:-1.16766000 +/- 0.15557188
[[Bayes Factors]]  (values >~1 => free parameter probably not useful)
    dfdbg:          0.000
[[Software versions]]
    CHEOPS DRP : cn01t-20200615T224519
    pycheops   : 0.9.2
    lmfit      : 0.9.14
```

Figure 13: The `lmfit` report for the transit of KELT-11b showing the fit statistics, the fitted parameter values and uncertainties, correlations and priors.

A useful feature of the `plot_lmfit` function is the binning of the light curve using the `binwidth` argument with the binned data being over-plotted on the full dataset in bins of width provided by the user in units of hours. Users can find a range of properties of the transit fit in the `lmfit` report, such as statistics (log likelihood, RMS, BIC, AIC, etc.), variable values and their uncertainties, any correlations between the parameters, the provided priors, and software versions used. If users notice that the fit is particularly bad they can view the starting position of the fit by printing `dataset.lmfit`. If these values are substantially erroneous compared to any priors provided to the `lmfit_transit` function then this may be the cause of the bad fit. Users can also use the Bayes Factors in the `lmfit` report for comparison between decorrelation models, as the values reported for each basis vector (i.e. time, background, glint function, etc.) are the Savage-Dickey Density Ratios [16] of fitting the model with and without the specific decorrelation basis vector. Thus, if a Bayes Factor larger than 1 is reported, the corresponding basis vector may not be useful to decorrelate against. Users can also include the logarithm of a fixed additional Gaussian white noise component to be added in quadrature with the flux errors when fitting a transit using the `lmfit_transit` function by setting the `log_sigma` argument to the desired value.

5.4.3.2 Using emcee

The physical and orbital properties of exoplanets can also be derived from transits in *CHEOPS* light curves using a Markov chain Monte Carlo (MCMC) methodology. PYCHEOPS does this by utilising the affine invariant sampler Python package `emcee` to sample the posterior probability distribution of fitting the constructed transit model to the data. This approach has several benefits when used in isolation, however a strength of this method is using it in combination with the `lmfit` fitting detailed above. This is because, if a `lmfit` least-squares fit is performed on a dataset, the best fit values from that analysis will be used as priors for the `emcee_sampler` function. Therefore, it is recommended that users run the previous code snippet before the example below.

For the `emcee_sampler` users can set the number of burn-in and sampling steps, and the number of walkers in the MCMC. Here the number of walkers defines the number of chains in the MCMC with the number of sampling steps representing how many steps around the posterior probability distribution each walker takes. The burn-in refers to the number of steps taken prior to the main sampling. This is done in an attempt to find the global minimum that is then sampled by the main MCMC. Therefore, it is recommended to make the burn-in a substantial fraction of the number of sampling steps.

Additional useful functions are `emcee_report` and `plot_emcee` that produce a readable report on the determined physical and orbital properties and a plot of the N_samples number of fitted transit models over-plotted on the data in a similar manner to the corresponding `lmfit` functions. The produced report is identical in nature to the report generated by the `lmfit_report` function described above with fit statistics, values, uncertainties, correlations, and priors provided to the user. The fitted transit plot is binned by giving a bin width value in hours to the `plot_emcee` function as shown in the code snippet below.

In PYCHEOPS, as well as transit or eclipse fitting, the `emcee_sampler` routine has built-in functionality to fit and remove correlated stellar noise using a Gaussian process regression method from the `celerite2` Python package. The regression is done by using a SHOTerm plus JitterTerm kernel that is constructed using `log_sigma`, `log_Q`, `log_omega0`, and `log_S0` parameters with bounds on the values of these parameters to be inputted by the user. This functionality can be included in the transit fit by setting the `add_shoterm` argument equal to True, as shown in the code snippet below. This regression constructs a kernel that includes stochastically driven, damped harmonic oscillator and white noise terms, that has been found to well model stellar granulation [11]. These parameters are related to the undamped period of the oscillator, $\rho = 2\pi/\omega_0$, the damping timescale of the process, $\tau = 2Q/\omega_0$, and the standard deviation of the process, $\sigma = \sqrt{S_0\omega_0 Q}$ that are used as priors on the SHOTerm kernel and reported in the fitting output.

In addition, a corner plot of the results of MCMC can be produced using the `corner_plot` function, with it possible to plot all fitted parameters by parsing "all" to the function. Users may also only plot selected parameters by providing an array of parameter names. It should be noted that plot tick values are not shown by default, but can be viewed by setting `show_ticklabels` equal to True.

Finally, as shown below, trail plots of "all" or an array of user selected parameters used in the MCMC can be produced using the `trail_plot` function. These plots show the step number of the MCMC against the parameter values and can be useful in indicating how well the parameters converged, with a good convergence reach if the chains show no clear trend.

```
from pycheops import Dataset

file_key = "CH_PR300024_TG000101_V0100"
D = Dataset(file_key)
aperture = "OPTIMAL"
time, flux, flux_err = D.get_lightcurve(aperture = aperture,
```

```
                                                    decontaminate = True)

N_steps = 256
N_walkers = 32
N_burn = 128
log_sigma_lower, log_sigma_upper = -10.5, -7.5
log_omega0_lower, log_omega0_upper = 3.5, 8.5
log_S0_lower, log_S0_upper = -30, -20
result = D.emcee_sampler(steps = N_steps, nwalkers = N_walkers,
                            burn = N_burn, add_shoterm = True,
                            log_sigma = (log_sigma_lower, log_sigma_upper),
                            log_omega0 = (log_omega0_lower, log_omega0_upper),
                            log_S0 = (log_S0_lower, log_S0_upper))
print(D.emcee_report())
corner_figure = D.corner_plot(["P", "T_0", "D", "W", "b"],
                                  show_ticklabels = True)
trail_figure = D.trail_plot("all")
bin_width_value = 0.01
N_samples = 32
figure = D.plot_emcee(binwidth = bin_width_value, detrend = True,
                         nsamples = N_samples)
```

It is also possible to save the chains of a MCMC fitting of *CHEOPS* data within PYCHEOPS using the backend functionality of the `emcee` package. This allows users to load a background and continue a previous analysis, for example if it did not reach convergence. If users import a backend HDF5 file into a variable using the `backends.HDFBackend` functionality of the `emcee` package, it is possible to load this into the PYCHEOPS `emcee_sampler` using the `backend` argument and subsequently continue running the analysis.

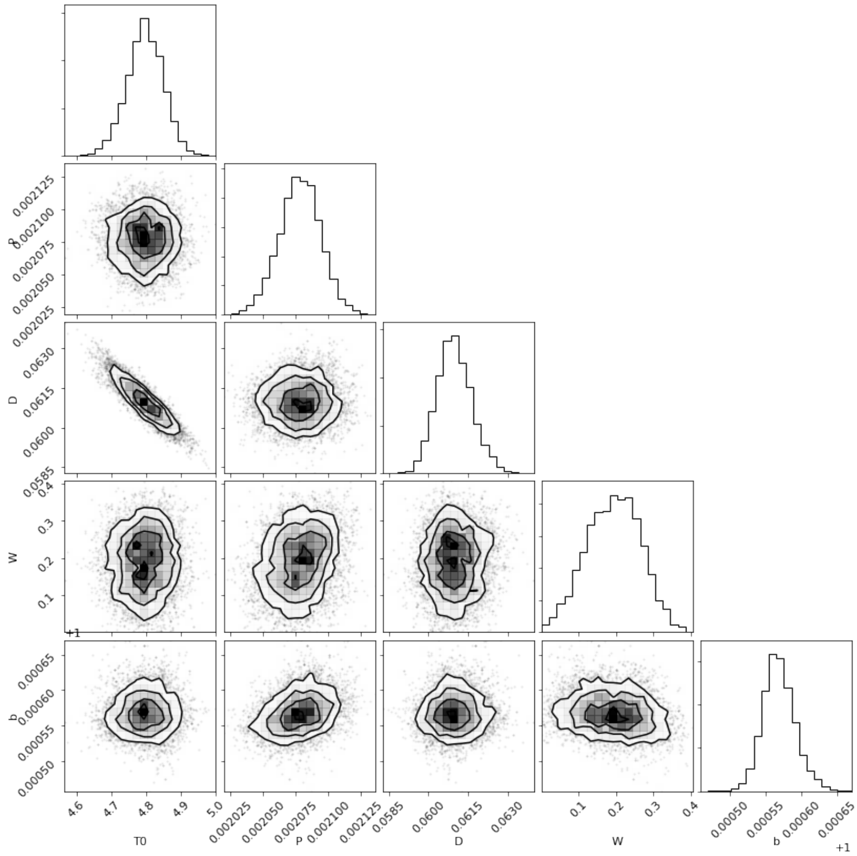


Figure 14: A corner plot of the MCMC fit of the transit of KELT-11b showing the distributions of and correlations between; the Period (P), the Transit centre time (T_0), the Depth (D), the Width (W), and the Impact parameter (b).

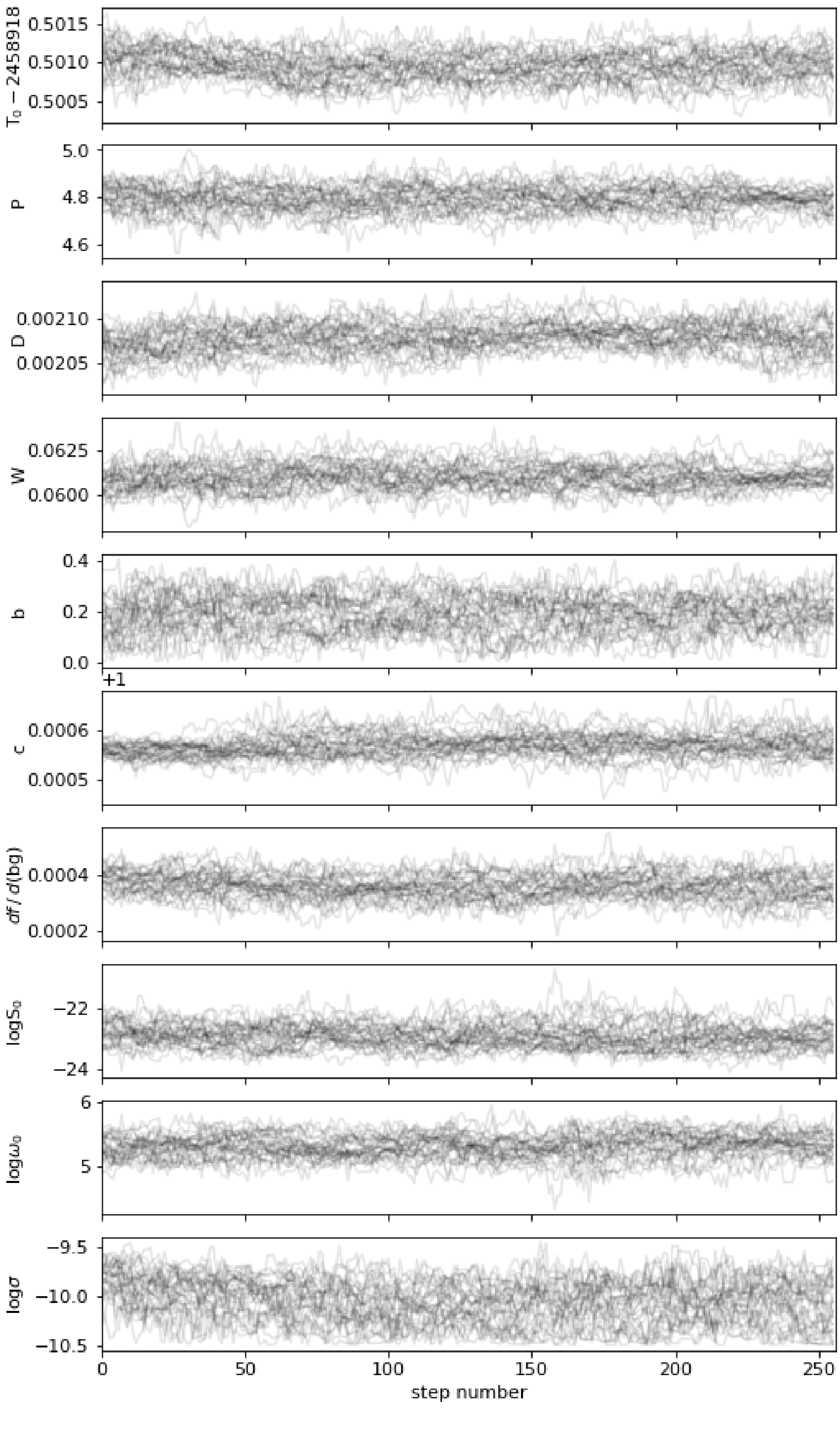


Figure 15: Example plots produced by parsing "all" to the `trail_plot` function that shows the parameters values against the step number of the MCMC.

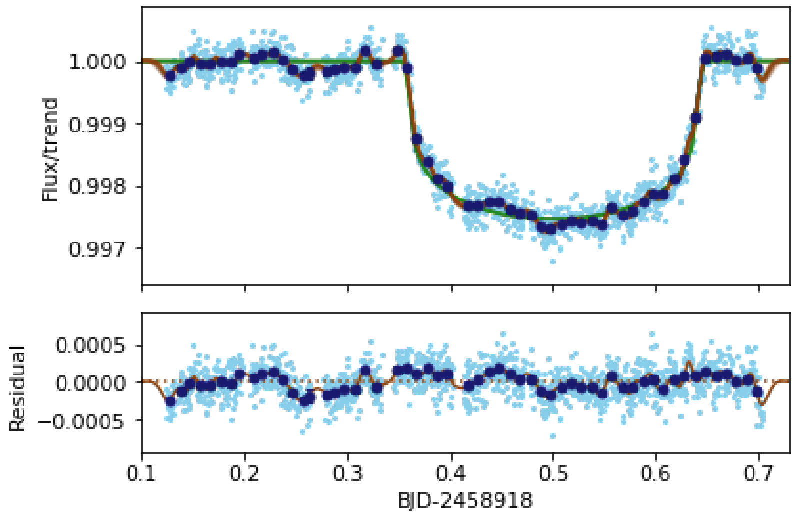


Figure 16: The *CHEOPS* light curve of a transit of KELT-11b. *Top.* The observed flux in blue with the `emcee` fit produced by the code snippet above in orange against time. *Bottom.* The residuals of the fit.

5.4.4 Fitting an Eclipse

PYCHEOPS also has specialised functions for fitting the eclipse of an exoplanet by its host star or a star in a binary. Using the same approach as for the transit fitting, users can build a model and then determine the following parameters; the orbital period (P), the eclipse depth (L), the transit centre time (T_0), the transit depth (D), the transit width (W), the light travel time (a_c), the impact parameter (b), a flux scaling factor (c), and the stellar orbit eccentricity and longitude of periastron components (f_c and f_s). It should be noted here that the transit width (W) is in units of phase, and that the transit centre time (T_0) is the time of the mid-point of the transit, and so this is eclipse centre time minus half of the period.

5.4.4.1 Using lmfit

A least-squares fitting of an eclipse model can be performed using the `lmfit` package. Similarly to fitting a transit, the PYCHEOPS `lmfit_eclipse` function, that utilising `lmfit`, can be used to construct a model, conduct any decorrelation, and fit the model to the data. Users can additionally run the `lmfit_report` and `plot_lmfit` functions to output a report and plots showing the fit as detailed in the code snippet below with the produced plot showing an example of eclipse fitting of WASP-189b taken from [7]:

```
from pycheops import Dataset
from uncertainties import ufloat

D = Dataset(file_key)
aperture = "OPTIMAL"
time, flux, flux_err = D.get_lightcurve(aperture = aperture,
                                        decontaminate = True)
D.lmfit_eclipse(P = ufloat(Period_value, Period_error),
```

```
                    L = ufloat(Eclipse_depth_value, Eclipse_depth_error),
                    T_0 = ufloat(Centre_time_value, Centre_time_error),
                    D = ufloat(Depth_value, Depth_error),
                    W = lmfit.Parameter(value = Width_value, vary = True),
                    a_c = lmfit.Parameter(value = Travel_time_value,
                                          vary = True),
                    b = lmfit.Parameter(value = Impact_value,
                                          min = Impact_min, max = Impact_max,
                                          vary = True),
                    c = (c_lower, c_upper),
                    f_c = f_c_value, f_s = f_s_value,
                    dfdcontam = (dfdcontam_lower, dfdcontam_upper),
                    dfdbg = (dfdbg_lower, dfdbg_upper))
print(D.lmfit_report())
figure = D.plot_lmfit(binwidth = bin_width_value, detrend=True)
```

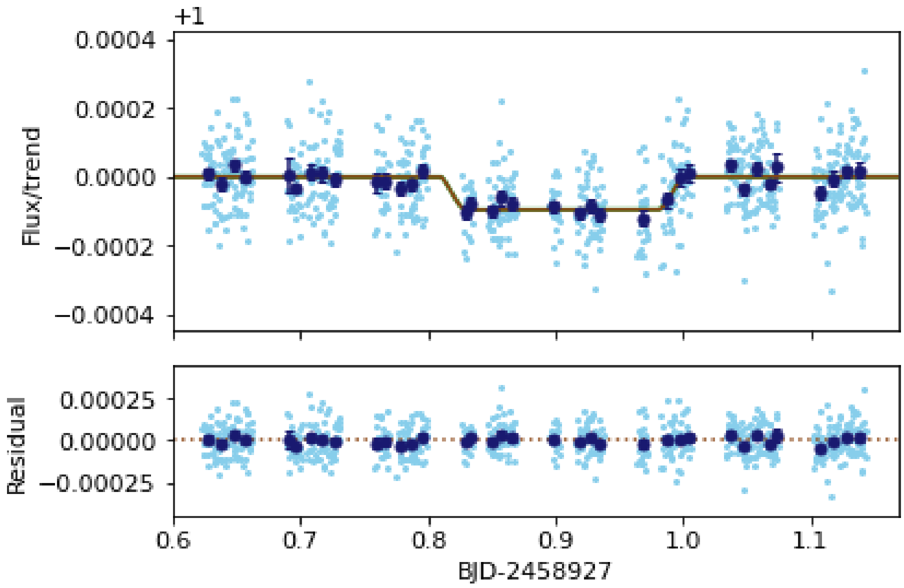


Figure 17: The *CHEOPS* light curve of an eclipse of WASP-189b. *Top.* The observed flux in blue with the `lmfit` fit in orange against time. *Bottom.* The residuals of the fit.

The `lmfit_report` produced by the code snippet above will be almost identical to the report from the transiting fitting of KELT-11b in Section 5.4.3.

5.4.4.2 Using emcee

The PYCHEOPS MCMC functions built from the `emcee` package described above in Section 5.4.3.2 for fitting transits can also be used to fit eclipses. The sole difference being the allowed strings for the array given to the `corner_plot` and `trail_plot` functions. As recommended above, if users wish to use the values of the parameters derived using the `lmfit` functions as priors for the MCMC, then the previous `lmfit` code snippet should be run before the following example:

```
from pycheops import Dataset

D = Dataset(file_key)
aperture = "OPTIMAL"
time, flux, flux_err = D.get_lightcurve(aperture=aperture,
                                        decontaminate=True)
N_steps = 256
N_walkers = 32
N_burn = 128
log_sigma_lower, log_sigma_upper = -10.5, -9.5
log_omega0_lower, log_omega0_upper = 3.5, 8.5
log_S0_lower, log_S0_upper = -30, -20
result = D.emcee_sampler(steps = N_steps, nwalkers = N_walkers,
                         burn = N_burn, add_shoterm = True,
                         log_sigma = (log_sigma_lower, log_sigma_upper),
                         log_omega0 = (log_omega0_lower, log_omega0_upper),
                         log_S0 = (log_S0_lower, log_S0_upper))
print(D.emcee_report())
corner_figure = D.corner_plot(["P", "L", "T_0", "D", "W"],
                              show_ticklabels = True)
trail_figure = D.trail_plot("all")
bin_width_value = 0.01
N_samples = 32
figure = D.plot_emcee(binwidth = bin_width_value, detrend = True,
                      nsamples = N_samples)
```

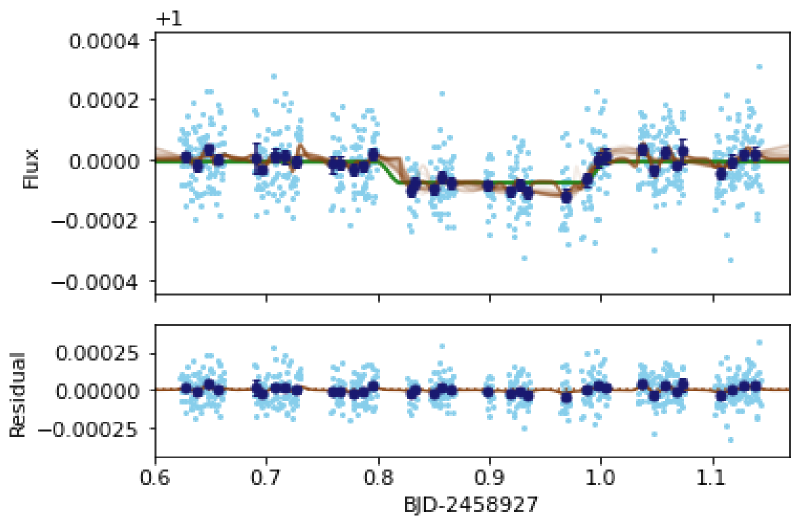


Figure 18: The *CHEOPS* light curve of an eclipse of WASP-189b. *Top.* The observed flux in blue with the `emcee` fit in orange against time. *Bottom.* The residuals of the fit.

The `corner_plot` and `trail_plot` functions will produce similar plots as to those seen for KELT-11b in Section 5.4.3.

5.4.5 Fitting a Transit and an Eclipse in the same Dataset

Whilst there are specific functions built into PYCHEOPS to do transit and eclipse fitting separately, it is also possible to construct a transit and eclipse combined model to fit datasets that contain both observed features. As shown above there are multiple physical and orbital parameters in common when building a transit and an eclipse model. Therefore, if both features are observed in a light curve users may wish to fit them simultaneously as this may lead to a better fit.

After obtaining the times and fluxes of a light curve, either using the `get_lightcurve` function or by other means, users must build a combined transit and eclipse model using `TransitModel()` and `EclipseModel()` objects. Note that if the `FactorModel()` object is also included, as it is in the example below, users can also detrend the light curve following the examples above. Following the creation of a Model, the parameter object of that model can be created using `make_params()`, and populated with the values and uncertainties (represented by the lower and upper bounds) for various physical and orbital parameters using the `add()` function. It should be noted that for the transit fitting parameters to be linked to the eclipse fitting parameters where possible, additional parameters must be added to the model. In the following example this is done using a `for` loop and the `add` function, and by giving the transit parameters the prefix "T_". This is in effect the same as using the `EBLMModel()` model in PYCHEOPS that can be used to model the transits and eclipses of eclipsing binaries with a low-mass companion. Therefore, in the following code snippet, when building the model the `TransitModel(prefix = "T_") * EclipseModel()` code could simply be replaced by `EBLMModel()`.

Using the `lmfit` package, the Model can fitted to the light curve using the Levenberg-Marquardt least squares method by setting up a chi-squared minimise function that uses the `eval` function to compare the observed flux with the model. The model flux can be returned using `eval`, and the residuals of the fit can be obtained using the `residual` function on the fit result. Finally, a readable report of the fit and determined physical and orbital parameters can be viewed with `fit_report`. The following is a code example of the fitting process described in the preceding paragraphs along with the code to plot these graphs.

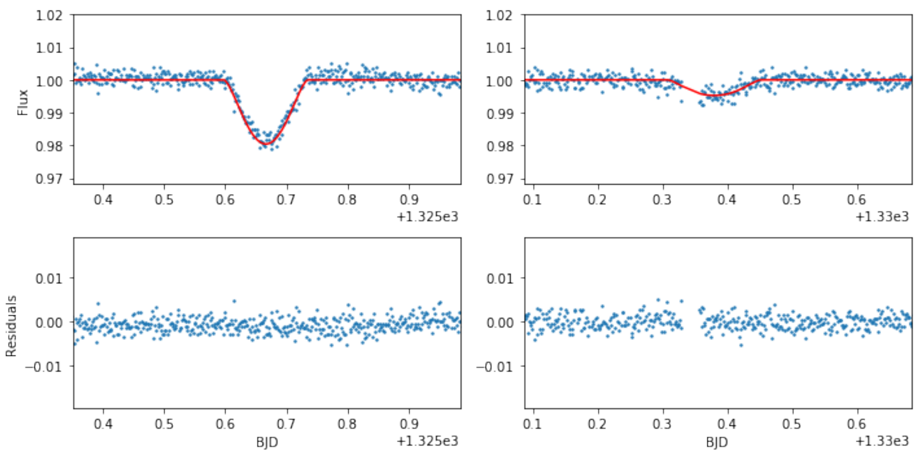


Figure 19: A `lmfit` fit of the transit and eclipse of an eclipsing binary system generated using the code snippet above. *Top left.* The transit/primary eclipse of the system with the flux in blue and the fitted model in red, against time. *Top right.* The fit of the secondary eclipse in red to the flux in blue, against time. *Bottom.* The residuals of the transit fit (*left*) and the eclipse fit (*right*).

Similarly to the previous examples it is possible to use the `emcee` Python package to fit both the transit and eclipse observed in a light curve simultaneously using a MCMC method. However, this requires the building of log-posterior and likelihood functions prior to running the `emcee` `EnsembleSampler` sampler and any plotting tools like the `corner` function in the corner package. It is recommended that users either utilise the `lmfit` fitting of both detailed above, or fit the transits and eclipse separately using either `lmfit` or `emcee` method.

```
from pycheops import Dataset
from pycheops.models import TransitModel, FactorModel, EclipseModel
import astropy.units as u
from lmfit import minimize
import numpy as np
from uncertainties import UFloat

D = Dataset(file_key)
aperture = "OPTIMAL"
time, flux, flux_err = D.get_lightcurve(aperture = aperture,
                                         decontaminate = True)

def _chisq_prior(pars, *args):
      r = (flux - model.eval(pars, t = time))/flux_err
      for p in pars:
            u = pars[p].user_data
            if isinstance(u, UFloat):
                  r = np.append(r, (u.n - pars[p].value)/u.s)
      return r

Model = FactorModel() * TransitModel(prefix = "T_") * EclipseModel()
pars = Model.make_params()
pars.add("P", value = Period_value, min = Period_lower, max = Period_upper)
pars.add("T_0", value = Centre_time_value, min = Centre_time_lower,
          max = Centre_time_upper)
pars.add("D", value = Depth_value, min = Depth_lower, max = Depth_upper)
pars.add("logrhoprior", value = Density_value, min = Density_lower,
          max = Density_upper)
pars.add("W", value = Width_value, min = Width_lower, max = Width_upper)
pars.add("b", value = Impact_value, min = Impact_lower, max = Impact_upper)
pars.add("f_c", value = f_c_value, min = f_c_lower, max = f_c_upper)
pars.add("f_s", value = f_s_value, min = f_s_lower, max = f_s_upper)
pars.add("T_h_1", value = h_1_value, vary=True)
pars.add("T_h_2", value = h_2_value, vary=True)
pars.add("L", value = Eclipse_depth_value, min = Eclipse_depth_lower,
          max = Eclipse_depth_upper)
```

```
pars.add("a_c", value = Travel_time_value, min = Travel_time_lower,
           max = Travel_time_upper)
for parameters in ["P", "T_0", "D", "W", "b", "f_c", "f_s", "c"]:
    pars.add("T_".format(parameters), expr = parameters)

result = minimize(_chisq_prior, pars, nan_policy = 'propagate',
                    args = (model, time, flux, flux_err))
flux_model = Model.eval(result.params, t = time)
residuals = result.residual

N_plots = 2
figure, ax = subplots(ncols = N_plots, nrows = N_plots)
for i in range(N_plots):
    ax[0][i].scatter(time, flux)
    ax[0][i].plot(time, flux_model, "r")
for j in range(N_plots):
    ax[1][j].scatter(time, residuals)
    ax[1][j].set_xlabel("BJD")
for k in range(N_plots):
    ax[k][0].set_xlim(Centre_time_value - 2 * Period_value * Width_value,
                      Centre_time_value + 2 * Period_value * Width_value)
for l in range(N_plots):
    ax[l][1].set_xlim(Centre_time_value + Period_value * (Eclipse_phase_value
                      - 2 * Eclipse_width_value),
                      Centre_time_value + Period_value * (Eclipse_phase_value
                      + 2 * Eclipse_width_value))
ax[0][0].set_ylabel("Flux")
ax[1][0].set_ylabel("Residuals")
print(result.fit_report())
```

```
[[Model]]
    ((Model(factor) * Model(_transit_func, prefix='T_')) * Model(_eclipse_func))
[[Fit Statistics]]
    # fitting method   = leastsq
    # function evals   = 316
    # data points      = 18094
    # variables        = 12
    chi-square         = 0.06108148
    reduced chi-square = 3.3780e-06
    Akaike info crit   = -227940.176
    Bayesian info crit = -227846.536
[[Variables]]
    d2fdt2:       0 (fixed)
    dfdt:         0 (fixed)
    dfdcosphi:    0 (fixed)
    dfdsinphi:    0 (fixed)
    dfdcos2phi:   0 (fixed)
    dfdsin2phi:   0 (fixed)
    d2fdy2:       0 (fixed)
    d2fdxdy:      0 (fixed)
    d2fdx2:       0 (fixed)
    dfdy:         0 (fixed)
    dfdx:         0 (fixed)
    c:            1.00004030 +/- 1.3875e-05 (0.00%) (init = 0)
    T_T_0:        1325.66664 +/- 0.00294021 (0.01%) == 'T_0'
    T_P:          8.79923603 +/- 2.6418e-04 (0.00%) == 'P'
    T_D:          0.02903141 +/- 0.01171632 (40.36%) == 'D'
    T_W:          0.01573351 +/- 0.00143301 (9.11%) == 'W'
    T_b:          0.90475491 +/- 0.17578035 (19.43%) == 'b'
    T_f_c:        0.23265316 +/- 1.57692572 (677.80%) == 'f_c'
    T_f_s:       -0.01390761 +/- 0.34208427 (2459.69%) == 'f_s'
    T_h_1:        0.68103339 +/- 0.80385140 (118.03%) (init = 0.8186516)
    T_h_2:        0.44373206 +/- 3.61005021 (813.57%) (init = 0.4161293)
    T_0:          1325.66664 +/- 0.00294021 (0.01%) (init = 1325.668)
    P:            8.79923603 +/- 2.6418e-04 (0.00%) (init = 8.79936)
    D:            0.02903141 +/- 0.01171632 (40.36%) (init = 0.02)
    W:            0.01573351 +/- 0.00143301 (9.11%) (init = 0.02)
    b:            0.90475491 +/- 0.17578035 (19.43%) (init = 0.9)
    L:            0.00571775 +/- 0.00402708 (70.43%) (init = 0.005)
    f_c:          0.23265316 +/- 1.57692572 (677.80%) (init = 0.234)
    f_s:         -0.01390761 +/- 0.34208427 (2459.69%) (init = 0)
    a_c:          0.00937926 +/- 4.09483086 (43658.35%) (init = 0)
    T_k:          0.17038607 +/- 0.00000000 (0.00%) == 'sqrt(T_D)'
    T_aR:         15.0206188 +/- 0.00000000 (0.00%) == 'sqrt((1+T_k)**2-T_b**2)/T_W/pi'
    T_rho:        0.58730180 +/- 0.00000000 (0.00%) == '0.013418*T_aR**3/T_P**2'
    k:            0.17038607 +/- 0.03438167 (20.18%) == 'sqrt(D)'
    J:            5.07742025 +/- 3.46807541 (68.30%) == 'D/L'
    aR:           15.0206188 +/- 4.43343408 (29.52%) == 'sqrt((1+k)**2-b**2)/W/pi'
    rho:          0.58730180 +/- 3.5266e-05 (0.01%) == '0.013418*aR**3/P**2'
    T_c:          1.00004030 +/- 1.3875e-05 (0.00%) == 'c'
[[Correlations]] (unreported correlations are < 0.100)
    C(f_c, a_c)      = -1.000
    C(T_0, f_c)      = -0.949
    C(T_0, a_c)      =  0.947
    C(T_h_1, T_h_2)  =  0.922
    C(b, L)          =  0.914
    C(T_h_1, L)      =  0.885
    C(T_h_2, L)      =  0.866
    C(T_h_1, b)      =  0.851
    C(T_h_2, b)      =  0.789
    C(T_0, f_s)      = -0.787
    C(W, a_c)        =  0.764
    C(W, f_c)        = -0.762
    C(T_0, W)        =  0.644
    C(b, f_c)        =  0.571
    C(b, a_c)        = -0.571
    C(f_c, f_s)      =  0.563
    C(f_s, a_c)      = -0.559
    C(D, f_s)        = -0.543
    C(T_0, b)        = -0.539
    C(T_h_2, f_s)    =  0.494
    C(L, f_s)        =  0.483
    C(b, f_s)        =  0.429
    C(T_h_2, W)      =  0.418
    C(D, b)          =  0.416
    C(T_h_1, W)      =  0.387
    C(T_h_1, D)      =  0.370
    C(D, L)          =  0.338
    C(T_0, L)        = -0.331
    C(T_h_2, T_0)    = -0.324
    C(T_0, D)        =  0.307
    C(L, f_c)        =  0.273
    C(L, a_c)        = -0.271
    C(W, L)          =  0.215
    C(W, f_s)        = -0.215
    C(T_h_2, f_c)    =  0.210
    C(T_h_2, a_c)    = -0.208
    C(T_h_1, f_s)    =  0.206
    C(T_h_1, T_0)    = -0.152
    C(T_h_1, f_c)    =  0.141
    C(T_h_1, a_c)    = -0.141
```

Figure 20: The `lmfit` report of the transit and eclipse fitting produced by the code snippet above, showing the fit statistics, determined parameter values and uncertainties, and parameter correlations.

5.4.6 Fitting a Thermal Phase Curve

Aside from fitting eclipses and transits, it is also possible to fit thermal phase curves of a tidally locked planet with PYCHEOPS. This is done by creating a `ThermalPhaseModel()` object and adding model parameters to it by first building a parameters object using `make_params()` and populating it using the `add` function. There are five parameters that can be used to construct the thermal phase curve model; the period (P), the transit centre time (T_0), and three properties of the curve: the coefficients of the cosine and sine terms (a_th and b_th) and a constant term that corresponds to the minimum flux (c_th). As can be seen in the code snippet below, the parameter values, minimum and maximum bounds, and a Boolean on whether or not to vary the parameter during the fitting can be set in the `add` function.

The data can then be fitted using a chi-squared minimisation function defined in the code snippet below and the model constructed function `eval` from the `lmfit` Python package to produce the result of the Levenberg-Marquardt least squares fitting. The `eval` function can be used again to calculate the flux of the best fitting model with the residuals obtainable using the `residual` function on the `result` object created. Finally, a report detailing the statistics, variables used, and correlations found during the fit can be returned using the `fit_report` function.

In addition to the fitted parameters, multiple derived values are reported that users may find useful. As can be seen in the thermal phase curve fitting report below, these are; the phase curve peak-to-trough amplitude (A), the maximum and minimum flux (Fmax and Fmin), and the phase to maximum flux (ph_max).

As with the simultaneous transit and eclipse fitting shown above, it is also possible to build more complex models, for example including detrending or eclipse fitting, by multiplying the `ThermalPhaseModel()` by other model classes such as `FactorModel()` or `EclipseModel()` and adding the desired parameters and their values. However, in PYCHEOPS there is a combined `PlanetModel()` that can be used to model transits, eclipses, and thermal phase curves, as is described below.

```
from pycheops import Dataset
from pycheops.models import ThermalPhaseModel
import astropy.units as u
from lmfit import minimize
import numpy as np
from uncertainties import UFloat

D = Dataset(file_key)
aperture = "OPTIMAL"
time, flux, flux_err = D.get_lightcurve(aperture = aperture,
                                         decontaminate = True)

def _chisq_prior(pars, *args):
      r = (flux - model.eval(pars, t = time))/flux_err
      for p in pars:
            u = pars[p].user_data
            if isinstance(u, UFloat):
                  r = np.append(r, (u.n - pars[p].value)/u.s)
      return r

Model = ThermalPhaseModel()
```

```
pars = Model.make_params()
pars.add("P", value = Period_value, min = Period_lower, max = Period_upper)
pars.add("T_0", value = Centre_time_value, min = Centre_time_lower,
            max = Centre_time_upper)
pars.add("a_th", value = a_th_value, vary = True)
pars.add("b_th", value = b_th_value, vary = True)
pars.add("c_th", value = c_th_value, vary = True)

result = minimize(_chisq_prior, pars, nan_policy = "propagate",
                    args = (model, time, flux, flux_err))
flux_model = Model.eval(result.params, t = time)
residuals = result.residual

figure, ax = subplots(2, 1, sharex = True)
ax[0].scatter(time, flux)
ax[0].plot(time, flux_model, "r")
ax[1].scatter(time, residuals)
ax[0].set_ylabel("Flux")
ax[1].set_ylabel("Residuals")
ax[1].set_xlabel("Time")
print(result.fit_report())
```

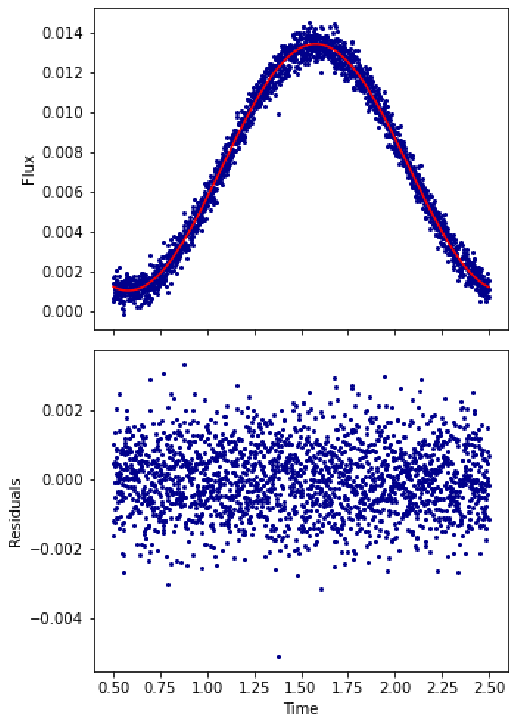


Figure 21: A `lmfit` fit of the thermal phase curve generated using the code snippet above showing the flux (blue) and fitted model (red) in the top panel, and the the residuals to the fit in the bottom panel.

```
[[Fit Statistics]]
    # fitting method   = leastsq
    # function evals   = 24
    # data points      = 2000
    # variables        = 5
    chi-square         = 2030.38515
    reduced chi-square = 1.01773692
    Akaike info crit   = 40.1566454
    Bayesian info crit = 68.1611577
[[Variables]]
    T_0:     0.49736123 +/- 15104.5341 (3036934.35%) (init = 0.5)
    P:       1.99626792 +/- 0.00553526 (0.28%) (init = 2)
    a_th:    0.01193730 +/- 149.476842 (1252183.28%) (init = 0.012)
    b_th:   -0.00314417 +/- 567.511120 (18049653.31%) (init = -0.003)
    c_th:    0.00282188 +/- 358.493978 (12704090.90%) (init = 0.0027)
    A:       0.01234443 +/- 3.5356e-05 (0.29%) == 'hypot(a_th,b_th)'
    Fmax:    0.01339066 +/- 1.9085e-05 (0.14%) == 'c_th+(a_th+b_th+A)/2'
    Fmin:    0.00104623 +/- 0.00000000 (0.00%) == 'Fmax - A'
    ph_max: -0.45901109 +/- 7566.38618 (1648410.31%) == 'arctan2(b_th,-a_th)/(2*pi)'
[[Correlations]] (unreported correlations are < 0.100)
    C(b_th, c_th) = -1.000
    C(a_th, c_th) = -1.000
    C(T_0, b_th)  =  1.000
    C(a_th, b_th) =  1.000
    C(T_0, c_th)  = -1.000
    C(T_0, a_th)  =  1.000
    C(T_0, P)     = -0.172
    C(P, b_th)    = -0.172
    C(P, c_th)    =  0.172
    C(P, a_th)    = -0.172
```

Figure 22: The `lmfit` report of the thermal phase curve fitting produced by the code snippet above, showing the fit statistics, determined parameter values and uncertainties, and parameter correlations

5.4.7 Fitting a Transit, Eclipse, and Thermal Phase Curve in the same Dataset

It has been shown in the previous subsections that it is possible within PYCHEOPS to use the built in models to construct more complex models to fit *CHEOPS* data. One such model is the `PlanetModel()` that provides a framework to fit a transit, eclipse, and thermal phase curve in a dataset. If this is combined with a `FactorModel()` users can also conduct decorrelation in addition to feature fitting. This can be done by building the desired model, adding parameters and their corresponding values and limits, fitting the model to the data using a least squares method, and plotting the result, as shown in the code snippet below.

As the procedure and parameters used are very similar to those provided in previous subsections, a detailed explanation is superfluous and users are referred to those subsections in order to obtain an understanding of the code snippet below. However, there are a few subtle differences between the `PlanetModel()` and the component models described before that should be mentioned. Unlike in the `TransitModel()` the stellar density is not a parameter, but rather is calculated explicitly within the model class. Similarly, the eclipse depth is also not a parameter that can be defined. Finally, to build the thermal phase curve model the parameters required are the thermal phase minimum and maximum flux (F_min and F_max) and the maximum flux phase offset (ph_off). These are calculated within the separate `ThermalPhaseModel()` and are different to the thermal phase curve coefficients used in that class.

Should users want to include these parameters in a model it is recommended that they replace the `PlanetModel()` with `TransitModel(prefix = "T_") * EclipseModel() * ThermalPhaseModel()` and link the common model parameters together as seen in the transit and eclipse fitting subsection above. For example, the period and transit centre time between the transit, eclipse, and thermal phase models, and the transit depth, width, impact parameter, and orbital eccentricity and longitude of periastron components between the transit and eclipse models.

```
from pycheops import Dataset
from pycheops.models import PlanetModel, FactorModel
import astropy.units as u
from lmfit import minimize
import numpy as np
from uncertainties import UFloat

D = Dataset(file_key)
aperture = "OPTIMAL"
time, flux, flux_err = D.get_lightcurve(aperture = aperture,
                                        decontaminate = True)

def _chisq_prior(pars, *args):
      r = (flux - model.eval(pars, t = time))/flux_err
      for p in pars:
           u = pars[p].user_data
           if isinstance(u, UFloat):
                r = np.append(r, (u.n - pars[p].value)/u.s)
      return r

Model = PlanetModel() * FactorModel()
pars = Model.make_params()
pars.add("P", value = Period_value, min = Period_lower, max = Period_upper)
pars.add("T_0", value = Centre_time_value, min = Centre_time_lower,
           max = Centre_time_upper)
pars.add("D", value = Depth_value, min = Depth_lower, max = Depth_upper)
pars.add("W", value = Width_value, min = Width_lower, max = Width_upper)
pars.add("b", value = Impact_value, min = Impact_lower, max = Impact_upper)
pars.add("F_min", value = Flux_minimum_value, min = Flux_minimum_lower,
           max = Flux_minimum_upper)
pars.add("F_max", value = Flux_maximum_value, min = Flux_maximum_lower,
           max = Flux_maximum_upper)
pars.add("ph_off", value = Phase_offset_value, min = Phase_offset_lower,
           max = Phase_offset_upper)
pars.add("f_c", value = f_c_value, min = f_c_lower, max = f_c_upper)
pars.add("f_s", value = f_s_value, min = f_s_lower, max = f_s_upper)
pars.add("h_1", value = h_1_value, vary=True)
pars.add("h_2", value = h_2_value, vary=True)
pars.add("a_c", value = Travel_time_value, min = Travel_time_lower,
pars.add("dfdbg", value = 0, vary=True)
pars.add("dfdcontam", value = 0, vary=True)
           max = Travel_time_upper)

result = minimize(_chisq_prior, pars, nan_policy = "propagate",
```

```
                    args = (model, time, flux, flux_err))
flux_model = Model.eval(result.params, t = time)
residuals = result.residual

N_plots = 2
figure, ax = subplots(ncols = N_plots, nrows = N_plots)
for i in range(N_plots):
    ax[0][i].scatter(time, flux)
    ax[0][i].plot(time, flux_model, "r")
for j in range(N_plots):
    ax[1][j].scatter(time, residuals)
    ax[1][j].set_xlabel("BJD")
for k in range(N_plots):
    ax[k][0].set_xlim(Centre_time_value - 2 * Period_value * Width_value,
                      Centre_time_value + 2 * Period_value * Width_value)
for l in range(N_plots):
    ax[l][1].set_xlim(Centre_time_value + Period_value * (Eclipse_phase_value
                      - 2 * Eclipse_width_value),
                      Centre_time_value + Period_value * (Eclipse_phase_value
                      + 2 * Eclipse_width_value))
ax[0][0].set_ylabel("Flux")
ax[1][0].set_ylabel("Residuals")
print(result.fit_report())
```

5.4.8 Saving your Datasets

Following the previous steps of obtaining and preparing your data, decorrelating and subsequent fitting of an eclipse or transit in the dataset, users may want to save the `Dataset` object they have been working on. This can be done by running the following code snippet after the steps the decorrelating and fitting steps outlined before:

```
D.save()
```

This saves all decorrelation and fitting done on a Dataset, including the addition of a glint function and modelling stellar variability using Gaussian processes, to a ".dataset" file in the current working directory. *Importantly, this must be done after a* `lmfit` *or* `emcee` *fit of an eclipse or transit is conducted.*

Conversely, saved datasets can be loaded using the corresponding `load` function:

```
D.load(filename)
```

Which be useful in order to inspect previous fits or when analysing multiple visits of the same target in PYCHEOPS.

Furthermore, users can construct CDS readable files of the dataset to be uploadable to the CDS website (http://cdsarc.u-strasbg.fr) using the `cds_data_export` function:

```
D.cds_data_export()
```

Values included in the file are the raw and detrended light curves, if decorrelation was done, and various instrument parameters from the visit (i.e. x and y centroid, roll angle, background, contamination, smear, and telescope temperature). Users may also define the name of the output file via the `lcfile` argument, and the title (`title`), first author (`author`), author list (`authors`), abstract (`abstract`), key words (`keywords`), bibliography code (`bibcode`), and acknowledgments (`acknowledgments`) relating to an upcoming publication via the corresponding argument keyword.

5.5 Fitting your Data - Multiple Visits

For targets observed multiple times with *CHEOPS* it is possible to use PYCHEOPS to analyse the visits simultaneously by fitting any observed eclipses or transits across the datasets. Furthermore, users can utilise functionality within PYCHEOPS to calculate any transit timing variations (TTVs) and eclipse depth variations (EDVs) between the fits of the individual visits. The multiple dataset fitting is done in a similar manner to the individual dataset fitting and is described below.

5.5.1 Loading your Datasets

In order to analyse multiple visits within PYCHEOPS a `MultiVisit` object needs to be created. This is done by passing the target name to the `MultiVisit` class as shown in the code snippet below. *Importantly, in order to build this object the current working directory is searched for ".dataset" files with the target name provided. Therefore, before attempting to create a* `Multivisit` *object users must create and save* `Dataset` *objects of the individual visits to be analysed. Crucially, this also requires the fitting of any eclipse or transit using the* `lmfit` *or* `emcee` *functions outlined in Section 5.4 before saving and incorporation into a* `MultiVisit` *object.*

```
from pycheops import MultiVisit

M = MultiVisit(target_name, id_kws = {"dace": False,
                                       "teff": (Teff_value, Teff_error),
                                       "match_arcsec": 10})
```

The `MultiVisit` object also queries the `StarProperties` function and prints the retrieved stellar property values (T_{eff}, log(g), Fe/H) and calculate stellar density and limb-darkening coefficients for potential use in subsequent fitting. Users can pass arguments from `MultiVisit` to `StarProperties` by listing keys and corresponding values in the `id_kws` dictionary argument. For example, users can specify not to query the DACE stellar table, set their own T_{eff} value, or change the search radius in SIMBAD and SWEET-Cat as shown in the code snippet above. A description of the complete set of `StarProperties` arguments is given in Section 5.4.1.

In addition to showing the stellar properties of the host star of the target, building a `MultiVisit` object also prints to the screen the retrieved saved datasets and lists the file key, aperture, and pipeline version of the data, the function last used to fit the eclipse or transit (i.e. `lmfit` or `emcee`), and if a Gaussian process or glint function was included.

5.5.2 Fitting Multiple Datasets - Transits or Eclipses

After loading the multiple datasets and prior to a global fit it can be beneficial to determine a transit centre time near the centre of the *CHEOPS* observing window in order to aid the eclipse or transit fitting, or the determination of TTVs if desired. This is be done by providing the `tzero` function with a known transit centre time and period, for example from a fit of an individual dataset. It should be noted that the known transit centre time should be provided in BJD. The function propagates the ephemerides and returns the transit centre time closest to the mid-point of the multiple visits.

```
from pycheops import MultiVisit

M = MultiVisit(target_name)
New_centre_time_value = M.tzero(Old_centre_time_value, Period_value)
```

In the `MultiVisit` class there are three functions that can be used to fit observed features and returns the result in the form of a `lmfit` object that includes the derived parameter values, fit statistics, and metadata; `fit_transit`, `fit_eclipse`, and `fit_eblm`, where the last routine fits both transits and eclipses in the saved datasets. These functions utilise the `emcee` Python package to conduct the fitting in a similar manner to the `emcee_sampler` function of the `Dataset` class. Therefore, there are many arguments in common, for example setting the number of walkers, and burn-in and main steps in the MCMC, and the option to model the stellar noise using a Gaussian process regression utilising the `celerite2` Python package. These arguments are covered in more detail in Section 5.4.3 and references therein, however briefly, the stellar variability and granulation has been found to be well modelled by a stochastically driven, damped harmonic oscillator and white noise that can be represented using SHOTerm and JitterTerm kernels that are built using the `log_sigma_w`, `log_omega0`, and `log_S0` terms and arguments.

It has been found that *CHEOPS* data can be affected by systematic trends that occur over a range of roll angle frequencies. Within the `Dataset` fitting routines it was possible to detrend against these effects using the sines and cosines of the first, second, and third order of roll angle frequencies, however when analysing multiple datasets this number of parameters scales quickly and becomes computationally intractable. Therefore, common to all three fitting functions is a new feature available in the `MultiVisit` class that conducts roll angle decorrelation on each dataset separately, automatically without the needed for any additional definition from the user. Instead of explicitly calculating the scaling factor free parameters for the sines and cosines of each roll angle frequency, these factors are implicitly marginalised over in the MCMC using a method presented here[12], and therefore this lack of calculation can significantly reduce the time needed to fit multiple datasets. By default, this feature is set to run up to the third order frequencies. Should users want to disable or alter this method it is possible by setting the `unroll` argument to False or changing the value of `nroll`. *However, it is strongly recommended utilise this feature as roll angle decorrelation when not needed is unlikely to degrade the light curve, but trends in the data are seen this is a straight-forward method to potentially remove them.* If the instrumental noise that is correlated with roll angle is large conducting decorrelation via this method may add additional noise to the dataset. Therefore, users can set the `unwrap` argument to True in order to divide the individual datasets by the corresponding roll angle trends found in previous `Dataset` fitting before conducting the simultaneous roll angle decorrelation detailed above.

The last feature common to all fitting routines is the `extra_priors` dictionary argument. This allows users to pass additional constraints, such as stellar density, or decorrelation basis vectors, for example flux versus time, in the same fashion as for the eclipse and transit fitting in the `Dataset` class, as can be seen in the examples below.

5.5.2.1 Fitting Transits

In addition to the features mentioned above, there are some arguments that are unique to the `fit_transit` function. As well as the parameters that define the transit fit (the orbital period (P), the transit centre time (T_0), the transit depth (D), the transit width (W), the impact parameter (b), the limb-darkening coefficients (h_1 and h_2), and the orbital eccentricity and longitude of periastron components (f_c and f_s)), users can also fit for TTVs by setting the `ttv` argument to True and defining the range in seconds of the TTVs to be searched over in `ttv_prior`, as can be seen in the code snippet below:

```
from pycheops import MultiVisit

M = MultiVisit(target_name)
result = M.fit_transit(T_0 = New_centre_time_value, P = Period_value,
                        ttv = True, ttv_prior = TTV_prior_value,
                        D = ufloat(Depth_value, Depth_error),
                        W = ufloat(Width_value, Width_error),
                        h_1 = h_1_value, h_2 = h_2_value,
                        steps = N_steps, nwalkers = N_walkers, burn = N_burn,
                        log_sigma_w = (log_sigma_w_lower, log_sigma_w_upper),
                        log_omega0 = (log_omega0_lower, log_omega0_upper),
                        log_S0 = (log_S0_lower, log_S0_upper),
                        extra_priors = {"logrho":  Log_stellar_density,
                                        "dfdt_1":  ufloat(0, 1),
                                        "dfdt_2":  ufloat(0, 1),
                                        "dfdt_3":  ufloat(0, 1)})
```

In this example, as TTVs are being fitted the transit centre time and period are fixed.

5.5.2.2 Fitting Eclipses

For fitting multiple eclipses, users can utilise the `fit_eclipse` function that contains the common features outlined above, and some functionality specific to eclipse fitting. This include the parameters used in the fitting (the orbital period (P), the eclipse depth (L), the transit centre time (T_0), the transit depth (D), the transit width (W), the light travel time (a_c), the impact parameter (b), and the stellar orbit eccentricity and longitude of periastron components (f_c and f_s)), and the option to fit EDVs. Similarly to fitting TTVs, this can be done by setting `edv` to True and providing a range of EDVs to fit over in `edv_prior` as can be seen in the following:

```
from pycheops import MultiVisit

M = MultiVisit(target_name)
result = M.fit_eclipse(L = Eclipse_depth_value,
                        edv = True, edv_prior = EDV_prior_value,
                        T_0 = ufloat(New_centre_time_value,
                                     New_centre_time_error),
```

```
                        P = ufloat(Period_value, Period_error),
                        D = ufloat(Depth_value, Depth_error),
                        W = ufloat(Width_value, Width_error),
                        steps = N_steps, nwalkers = N_walkers, burn = N_burn,
                        log_sigma_w = (log_sigma_w_lower, log_sigma_w_upper),
                        log_omega0 = (log_omega0_lower, log_omega0_upper),
                        log_S0 = (log_S0_lower, log_S0_upper),
                        extra_priors = {"logrho": Log_stellar_density})
```

As this example shows the fitting of EDVs the absolute eclipse depth is fixed with the error taken to be in the range of the EDV prior given. *Importantly, it should be noted that the T_0 value here is the transit centre time and not the eclipse centre time. Therefore, if users only have observations of eclipses they should subtract half the phase from the observed eclipse centre time for the correct T_0.*

5.5.2.3 Fitting Transits and Eclipses

As well as fitting transits and eclipses seen in multiple visits separately, it is also possible to fit multiple visits that contain both an eclipse and a transit simultaneously. This can be done in using the `fit_eblm` function that uses all the features outlined above including fitting the transits for TTVs and the eclipses for EDVs. It should be noted that whilst both sets of features are fitted, the function does not include the fitting of any thermal or phase effects.

```
from pycheops import MultiVisit

M = MultiVisit(target_name)
result = M.fit_eblm(T_0 = New_centre_time_value, P = Period_value,
                        L = Eclipse_depth_value,
                        ttv = True, ttv_prior = TTV_prior_value,
                        edv = True, edv_prior = EDV_prior_value,
                        D = ufloat(Depth_value, Depth_error),
                        W = ufloat(Width_value, Width_error),
                        h_1 = h_1_value, h_2 = h_2_value,
                        steps = N_steps, nwalkers = N_walkers, burn = N_burn,
                        log_sigma_w = (log_sigma_w_lower, log_sigma_w_upper),
                        log_omega0 = (log_omega0_lower, log_omega0_upper),
                        log_S0 = (log_S0_lower, log_S0_upper),
                        extra_priors = {"logrho": Log_stellar_density,
                                        "dfdt_1": ufloat(0, 1),
                                        "dfdt_2": ufloat(0, 1),
                                        "dfdt_3": ufloat(0, 1)})
```

5.5.3 Plotting and Assessing the Multiple Visit Fits

There are several plotting and assessment functions available in the `MultiVisit` class that users can run to view the fitting of multiple transits, eclipses, or both conducted with the functions mentioned above. These include plotting the fitted feature, the trail and corner plots of the fit, printing the fit report, and in the case of fitting TTVs, an observed minus calculated (O-C) plot. It should be noted that the first four functions described above can be run after the fitting of multiple transits, eclipses, or both.

To highlight the functionality of `MultiVisit`, the code snippet below shows a simplified `MultiVisit` fit including EDVs showing the plotting and assessment functions of the *CHEOPS* observations of WASP-189b that have been recently published [7]:

```
from pycheops import MultiVisit

M = MultiVisit(target_name)
result = M.fit_eclipse(L = Eclipse_depth_value,
                        edv = True, edv_prior = EDV_prior_value,
                        T_0 = ufloat(New_centre_time_value,
                                         New_centre_time_error),
                        P = ufloat(Period_value, Period_error),
                        h_1 = h_1_value, h_2 = h_2_value,
                        steps = N_steps, nwalkers = N_walkers, burn = N_burn,
                        extra_priors = {"logrho":  Log_stellar_density})

figure = M.plot_fit(detrend = True, add_gaps = True, gap_tol = 0.001,
                      data_offset = 0.0005, res_offset = 0.0005)
trail_figure = M.trail_plot(plotkeys = "all", plot_kws = {"color":  "k"})
corner_figure = M.corner_plot(plotkeys = "all", show_priors = True)
print(M.fit_report())

fig, ax = plt.subplots()
for j in range(len(M.datasets)):
    t = M.datasets[j].lc["time"].mean() - 1900
    edv = M.result.params[f"L_j+1:02d"].value - M.result.params["L"].value
    edv_err = M.result.params[f"L_j+1:02d"].stderr
    ax.errorbar(t, edv, yerr = edv_err, color = "b")
    plt.axhline(0, c = "darkcyan", ls = ":")
    ax.set_xlabel("BJD - 2458900")
    ax.set_ylabel(r"ΔL (ppm)")
```

The `plot_fit` function is similar to the `Dataset` `plot_lmfit` and `plot_emcee` functions described in Section 5.4.3, albeit with multiple unique features to aid in viewing multiple visits. For example, in addition to plotting the decorrelated light curves by setting the `detrend` argument to True, users can specify the offset between the fluxes and residuals from each visit using the `data_offset` and `res_offset` arguments, and set the y-axes limits for the data and residual subplots by giving a length=2 tuple to the `data_ylim` and `res_ylim` arguments. When plotting after using `fit_eblm`, fluxes and residuals can be offset for transits and eclipses separately using a length=2 tuple. As can be seen below it is also possible to avoid plotting over gaps in the data by setting the `add_gaps` argument to True. The fitted model will then not be plotted over gaps larger than specified by the `gap_tol` argument.

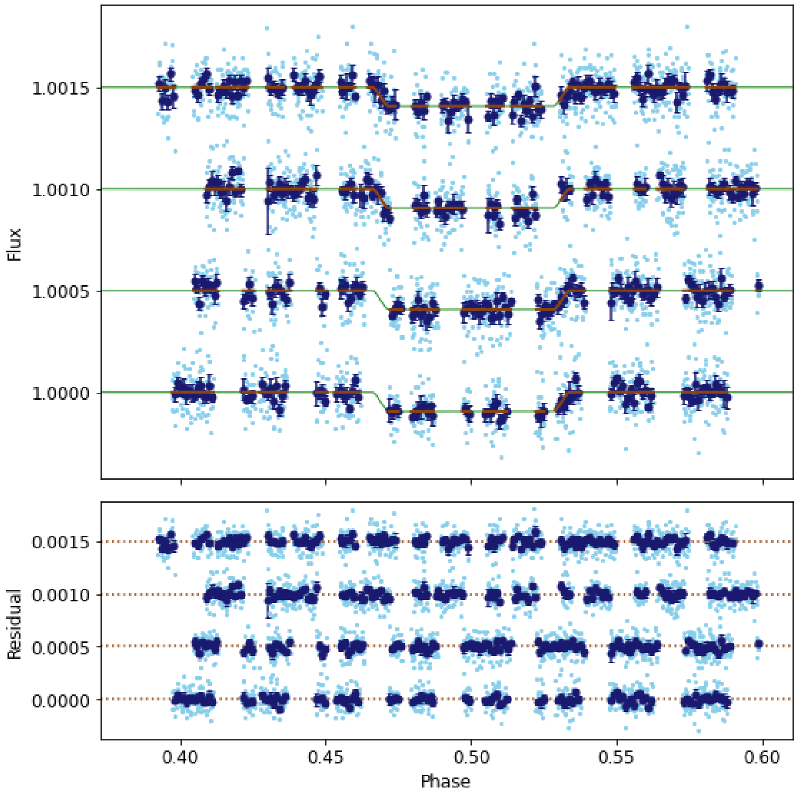


Figure 23: The multiple visit eclipse fit of WASP-189b showing the raw photometry in light blue, binned values in dark blue, transit model in green, and the combined transit, Gaussian process, and decorrelation model in brown, with gaps in the combined model apparent.

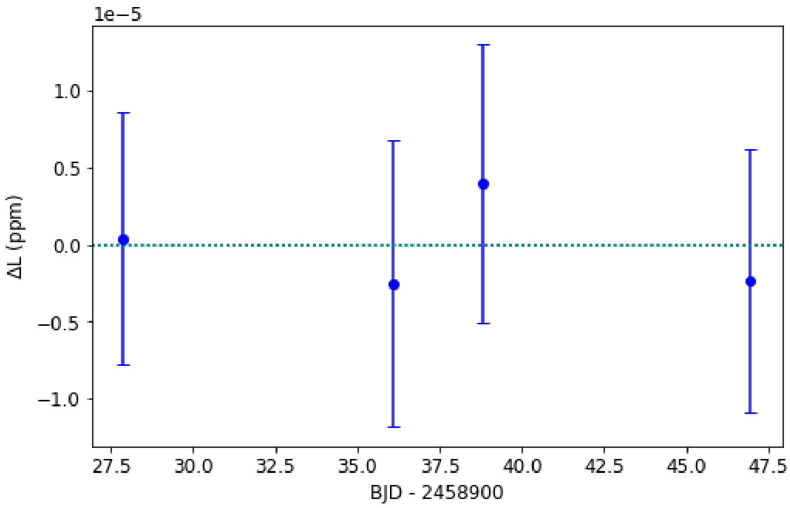


Figure 24: The EDV plot of the WASP-189b *CHEOPS* observations produced using the code snippet above showing the transit centre time shift from the calculated value against the observed epoch.

The trail_plot, corner_plot, and fit_report MultiVisit functions are effectively identical to the corresponding Dataset functions detailed in Section 5.4.3. For example, the plotkeys argument allows users to select which fitted parameters should be plotted by including parameter names in a list or by giving the string: "all". Another useful feature to highlight is the plot_kws dictionary argument in the trail_plot function. This allows the user to tweak aesthetic aspects of the trail_plot, such as plotting all the chains in black.

A unique feature of the MultiVisit class is the ttv_plot function that produces an O-C plot of the determined TTV values from the MultiVisit fit and plots them against the fitted transit centre time. Similarly to the trail_plot, the dictionary argument plot_kws can be used to change aesthetic aspects, for example, the marker colour and shape, or errorbar cap size.

5.5.4 Producing CDS Readable Files

Following a complete analysis of the *CHEOPS* data of an object using the MultiVisit functionality, users can create CDS readable data files for upload to the VizieR library of catalogues in relation to a paper that uses the data. This can be done by providing text strings of the paper title, lead and co-authors, abstract, keywords, bibcode, and acknowledgements to the cds_data_export function:

```
with open("abstract.txt") as fh:  abstract = fh.read()
with open("authors.txt") as fh:  authors = fh.read()
with open("keywords.txt") as fh:  keywords = fh.read()

M.cds_data_export(title="PYCHEOPS Cookbook v1.0",
                  author="Wilson T. G.",
                  authors=authors,
                  abstract=abstract,
                  keywords=keywords,
                  bibcode="XXXXYYYYY...ZZZZ..AAA",
                  acknowledgements="Thomas G. Wilson,
                                   tgw1(at)st-andrews.ac.uk")
```

5.6 Further Analysis of the Data

5.6.1 Estimating Light Curve Noise

In addition to fitting features of the light curves as detailed above, PYCHEOPS also provides the ability to estimate the noise in the data. This is done by determining the depth of a transit that would be detected with a signal-to-noise (S/N) of 1, which is similar to the Combined Differential Photometric Precision (CDPP) method used to calculated noise in *Kepler* light curves. To calculate this value a nominal transit model (with an impact parameter, b = 0, and a width, in hours, over which the noise is calculated) is inserted into the light curve and scaled until the S/N is met.

To determine the transit depth that satisfies this requirement the model is fit to the data with the true standard errors of the data calculated via two methods; the scaled errors method and the minimum errors method. In the scaled errors method it is assumed that the true flux uncertainties are underestimated by a factor compared to the reported errors, whereas in the minimum errors method the reported errors are taken to be the lower-bound of the true flux uncertainty distribution. In general, the minimum errors method usually assumes greater true standard errors of the data and, thus, produces a larger S/N = 1 transit depth. A detailed break-down of the calculation of the errors via these methods is beyond the scope of this document and therefore, users are referred here [14].

PYCHEOPS calculates these values for a light curve using the `transit_noise_plot` function in the Dataset module. This function assumes a flat light curve with no transit that is normalised to 1. Therefore, prior to estimating the noise of the dataset, any transit apparent in the light curve must be fit and removed. This can be done using either the `lmfit` or `emcee` methods detailed above to obtain the transit parameters which are then input in the `lmfit_transit`. The example below shows how to use `lmfit_transit`, `plot_transit`, and `transit_noise_plot` functions to fit and remove a transit from the light curve and then calculate the transit noise. In this example, parameter values are previously defined in Section 5.4.

```
from pycheops import Dataset
from uncertainties import ufloat

file_key = "CH_PR300024_TG000101_V0100"
D = Dataset(file_key)
aperture = "OPTIMAL"
time, flux, flux_err = D.get_lightcurve(aperture = aperture,
                                        decontaminate = True)

D.lmfit_transit(P = ufloat(Period_value, Period_error),
                T_0 = ufloat(Centre_time_value, Centre_time_error),
                D = ufloat(Depth_value, Depth_error),
                logrhoprior = Log_stellar_density,
                W = ufloat(Width_value, Width_error),
                b = ufloat(Impact_value, Impact_error),
                h_1 = h_1_value, h_2 = h_2_value)
figure = D.plot_lmfit()
residuals = (D.lc["flux"] - D.model.eval(D.lmfit.params, t = D.lc["time"]))
D.lc["flux"] = residuals + 1
Window_width_value = 3
noise_dictionary = D.transit_noise_plot(width = Window_width_value)
```

In order to provide a more representative noise estimate value the `transit_noise_plot` function splits the light curve into 500 segments, by default, and calculates the noise on each of those segments separately. The minimum, maximum, and mean of the array of noises is then reported, as can be seen above, and return in the form of a dictionary if the `return_values` argument is set to True. In the output noise plot the green line indicates the minimum error noise estimates and the blue line indicates the scaled error noise values. It should be noted that as no transit fit is perfect additional noise may be added into the light curve when the model is subtracted from the data to obtain the residuals. Therefore, the produced noise estimates are more conservative than the true noise of the light curve.

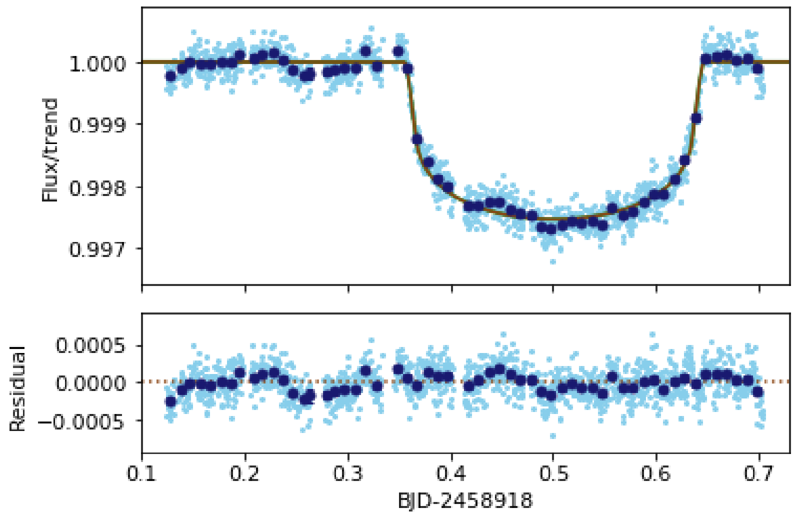


Figure 25: A `lmfit` fit to the transit of KELT-11b identical to Figure. 5. In order to calculate the noise of the dataset the residuals of the fit (seen in the bottom plot) must be used.

```
Scaled noise method
Mean noise = 8.9 ppm
Min. noise = 8.4 ppm
Max. noise = 9.4 ppm
Mean noise scaling factor = 1.243
Min. noise scaling factor = 1.224
Max. noise scaling factor = 1.247

Minimum error noise method
Mean noise = 12.2 ppm
Min. noise = 11.2 ppm
Max. noise = 13.0 ppm
```

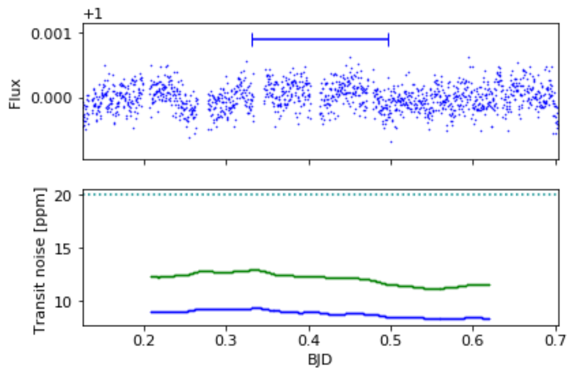


Figure 26: The calculated transit noise statistics for both scaled noise and minimum error noise methods. The plots show the flux of the light curve against time (*top*). Here it is the residuals of the transit fit. The blue horizontal bar indicates the input transit width. *Bottom.* The transit noise calculated via the scaled noise method, in blue, and via the minimum error noise method, in green. The dashed line at the top of the plot shows the "required noise value" that the user wishes to compare the true noise against. This can be set using the `requirement` argument in the `transit_noise_plot` function.

5.6.2 Calculating and Plotting the Planet Properties against Internal Structure Models

Following the fitting of a transit, PYCHEOPS users may want to plot the *CHEOPS* derived planetary radius against a known value of the planet mass and theoretical internal structure models in order to infer the internal properties of the target. This can be done using the `massradius` function after fitting the transit using either the `lmfit` and/or `emcee` methods detailed above. In order to calculate the planetary properties users should provide the host star's mass and radius in Solar units in the `m_star` and `r_star` arguments, respectively. If only one is provided, then the other stellar parameter is derived using the stellar density determined from the transit fit. The stellar density is calculated assuming that the planet to star mass ratio tends to 0. If this is invalid for the target system, users can provide a mass ratio in the argument `q`.

To determine the planetary mass, users should provide a known value of the semi-amplitude of the planet's orbit in m/s using the argument `K` that can be returned for planets with a semi-amplitude value in TEPCat or DACE using `PlanetProperties`. Finally, users can choose to view the outputted information relative to Jupiter, or Earth values by setting `jovian` equal to True or False, respectively. If the following code snippet is run then above the mass versus radius plot, a range of stellar and planetary properties will be reported, for example: the stellar and planetary mass, radius, and density, mass ratio, planet semi-major axis, and planetary surface gravity as can be seen below.

As mentioned above, this function can be used to over-plot a range of theoretical internal structure models on the mass versus radius figure for the planet. Currently, models from Zeng et al. (2016) [17] and Baraffe et al (2008) [1] can be selected by setting the `"zeng_models"` or `"baraffe_models"` key in the `plots_kws` argument to be equal to "all" or an array of specific models. By default, a range of well-studied planets will be plotted with values taken from TEPCat [15].

```
from pycheops import Dataset

file_key = "CH_PR300024_TG000101_V0100"
D = Dataset(file_key)
stellar_mass_value, stellar_mass_error = 1.44, 0.06
stellar_radius_value, stellar_radius_error = 2.72, 0.21
semi_amplitude_value, semi_amplitude_error = 18.5, 1.7
mass_ratio_value, mass_ratio_error = 0.00013, 0.00001
result, figure = D.massradius(m_star = (stellar_mass_value,
                                         stellar_mass_error),
                                r_star = (stellar_radius_value,
                                         stellar_radius_error),
                                K = (semi_amplitude_value,
                                         semi_amplitude_error),
                                q = (mass_ratio_value, mass_ratio_error),
                                jovian = True,
                                plot_kws = {"baraffe_models":  "all"})
```

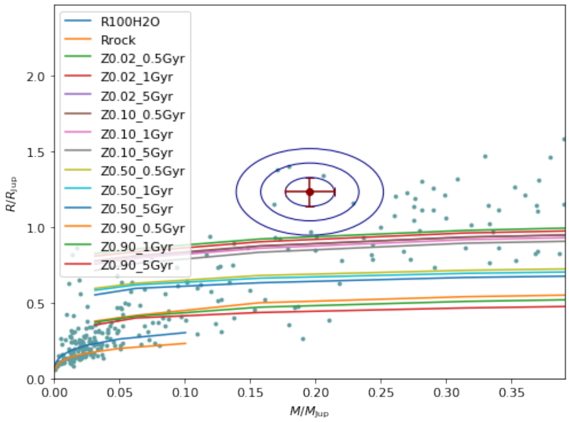


Figure 27: Example mass versus radius plot with the planet shown in maroon with dark blue error contours. Theoretical internal structure models from Baraffe et al. (2008) are shown with well-studied planets in cyan.

It should be noted that as well as producing mass versus radius plots for individual datasets, it is also possible to construct these plots using radii determined via fitting multiple visits using the same models and arguments as detailed above. This can be done after a `MultiVisit` fit by running:

```
result, figure = M.massradius(m_star = (stellar_mass_value,
                                        stellar_mass_error),
                              r_star = (stellar_radius_value,
                                        stellar_radius_error),
                              K = (semi_amplitude_value,
                                        semi_amplitude_error))
```

The inputted and calculated values are returned in the form of a dictionary, with the samples used in the determination of parameter statistics and uncertainties also provided if the `return_samples` argument is set equal to True.

5.6.3 Ploting the Fourier Transform of the Dataset

Upon fitting the light curve, if users notice a periodic variability in the dataset then they can utilise the `plot_fft` function to construct and plot a Lomb-Scargle power spectrum of the residuals to the eclipse or transit fit in order to assess if the periodic trend is due to stellar variability. In addition, a fast Fourier transform of the smoothed residuals is shown, along with an indicator of the *CHEOPS* orbital frequency and the first two harmonics. If the stellar T_{eff} and $log(\mathrm{g})$ are known, for example by using the StarProperties functionality, and $5000 < T_{eff} < 7000\,\mathrm{K}$, then the maximum stellar variability frequencies can be calculated [3], and over-plotted on the figure in green. Lastly the expected white-noise level is plotted as a dashed grey line, as shown below:

```
from pycheops import Dataset, StarProperties

file_key = "CH_PR300024_TG000101_V0100"
D = Dataset(file_key)
host_star_properties = StarProperties(D.target)
```

```
figure = D.plot_fft(host_star_properties)
```

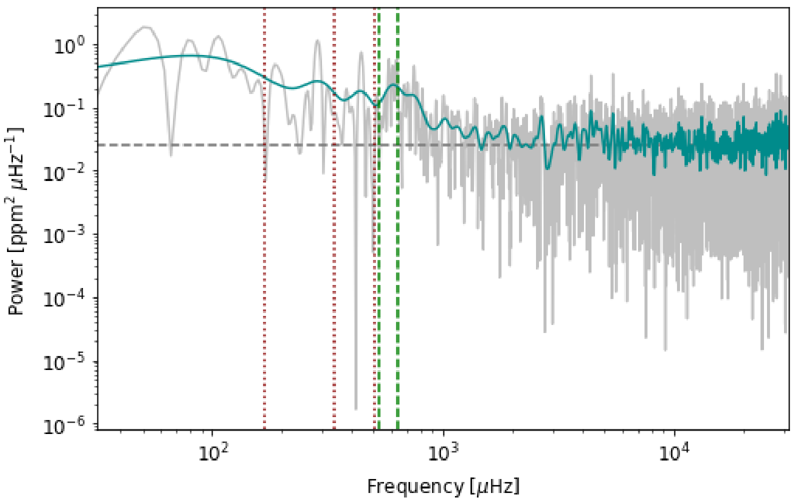


Figure 28: Example Lomb-Scargle power spectrum, in grey, and smoothed in dark green. *CHEOPS* orbital frequency harmonics are in red, and maximum stellar variability frequency estimates are in light green.

Appendix

A A Code Compilation for Downloading, Viewing, Decorrelating, and Fitting your Data

This section provides a code example compiling many of the code snippets presented in Sections 4 and 5 to download, view, decorrelate, and fit *CHEOPS* data of an observed transit with the aim to provide PYCHEOPS users with a holistic code capable of basic light curve analysis. The code below has been written in a generalised manner and therefore, can be used as a template to fit eclipses, if the `lmfit_transit` function is replaced with the `lmfit_eclipse` function, or for further analysis.

```
# Import the relevant modules.
from pycheops import Dataset, PlanetProperties, StarProperties
from uncertainties import ufloat

# Download the data from DACE and view the DRP report.
file_key = "CH_PR300024_TG000101_V0100"
D = Dataset(file_key)

# View the light curve by selecting which aperture size to use.
# For most visits "OPTIMAL" is recommended.
aperture = "OPTIMAL"
time, flux, flux_err = D.get_lightcurve(aperture = aperture,
```

```
                                                decontaminate=True)
plt.plot(time, flux, "k.")
plt.title(D.target + " - aperture = " + aperture)
plt.xlabel("BJD Date (d)")
plt.ylabel("Normalised Flux")

# Animate every 10th subarray and imagette frame of the light curve
# to assess if any nearby objects have contaminated the photometry.
Nth_frames = 10
Min_values_scaling_factor = 1.0
Max_values_scaling_factor = 1.0
frames = D.animate_frames(nframes = Nth_frames,
                            vmin = Min_values_scaling_factor,
                            vmax = Max_values_scaling_factor,
                            subarray = True, imagette = True)

# Create a diagnostic plot of the dataset to view key properties
# of the observations.
D.diagnostic_plot()

# Run the decorrelation tool to assess if there are any trends in the data
# that should be removed.
D.should_I_decorr()

# Check the separation between the target and bright Solar System objects
# to assess if any glint flux artefacts need to be modelled and removed.
D.planet_check()

# Obtain properties of the host star and planet to be used in the fitting.
host_star_properties = StarProperties(D.target)
Log_star_density = host_star_properties.logrho
h_1_value = host_star_properties.h_1.n
h_2_value = host_star_properties.h_2.n

planet_properties = PlanetProperties("KELT-11b", query_tepcat = True,
                                     query_dace = False)
Transit_reference_epoch = planet_properties.T0
Period_value, Period_error = planet_properties.P.n, planet_properties.P.s
cycle = round((D.bjd_ref-Transit_reference_epoch.n)/Period_value)
Transit_centre_time = Transit_reference_epoch - D.bjd_ref + cycle *
                      ufloat(Period_value, Period_error)
Depth_value = planet_properties.depth.n / 1.e6
Width_value = planet_properties.width.n / Period_value

# If glint flux artefacts are seen, do a preliminary fit of the transit
# to produce flux residuals that are used to construct the glint model,
```

```
# and plot flux residuals against roll angle.
Centre_time_value, Centre_time_error = Transit_centre_time.n,
                                       Transit_centre_time.s
D.lmfit_transit(P = ufloat(Period_value, Period_error),
                T_0 = ufloat(Centre_time_value, Centre_time_error),
                logrhoprior = Log_stellar_density)
N_spline = 30
glint = D.add_glint(nspline=N_spline)
D.rollangle_plot()

# Conduct a fit of the transit using the lmfit package,
# and produce plots of the fit and a report of the derived parameters.
# Remember that light curve decorrelation can be done simultaneously
# with transit fitting, for example against x and y centroid position,
# and glint if needed.
D.lmfit_transit(P = ufloat(Period_value, Period_error),
                T_0 = ufloat(Centre_time_value, Centre_time_error),
                logrhoprior = Log_stellar_density,
                h_1 = h_1_value, h_2 = h_2_value,
                dfdx = ufloat(0., 1.), dfdy = ufloat(0., 1.),
                glint_scale = (0., 2.))
figure = D.plot_lmfit(detrend = True)
print(D.lmfit_report())

# Conduct any clipping of outliers not cleaned by the decorrelation.
clipping_factor_value = 5
time, flux, flux_err = D.clip_outliers(clip = clipping_factor_value)

# Run a MCMC transit fit that automatically uses the least-squares
# fit transit properties derived from the lmfit fit as priors, and
# produce a report of the derived properties, corner and trail plots of the
# posterior probability distributions, and a plot of the fit to the data.
N_steps = 256
N_walkers = 32
N_burn = 128
N_samples = 32
result = D.emcee_sampler(steps = N_steps, nwalkers = N_walkers,
                         burn = N_burn)
print(D.emcee_report())
corner_figure = D.corner_plot("all")
trail_figure = D.trail_plot("all")
figure = D.plot_emcee(nsamples = N_samples)

# Estimate the noise of the light curve from the residuals to the fit.
residuals = (D.lc["flux"] - D.model.eval(D.lmfit.params, t = D.lc["time"]))
D.lc["flux"] = residuals + 1
Window_width_value = 3
```

```
noise_dictionary = D.transit_noise_plot(width = Window_width_value)

# If a periodic flux trend is still noticeable in the flux residuals of
# the fit, conduct a Fourier transform to assess if this is due to
# stellar variability.
fft_figure = D.plot_fft(host_star_properties)

# Lastly, if the mass of the target planet is known then plot a mass
# versus radius diagram with theoretical internal structure models shown.
stellar_mass_value, stellar_mass_error = 1.44, 0.06
semi_amplitude_value, semi_amplitude_error = 18.5, 1.7
result, figure = D.massradius(m_star=(stellar_mass_value, stellar_mass_error),
                              K=ufloat(semi_amplitude_value,
                                       semi_amplitude_error),
                              jovian=True, plot_kws={"baraffe_models":"all"})
```

B Description of the PYCHEOPS Functions in this Cookbook

In this section detailed outline of the PYCHEOPS functions used in this cookbook are given with a brief description of the specific function followed by an instance of the function with the keyword arguments showing the default values, and a listing and description of all function argument parameters and the expected returned variables.

pycheops.Dataset

A class that creates a Dataset object that downloads or extracts *CHEOPS* data from the DACE archive or a local directory based on the inputted `file_key`, and produces object specific tables and arrays. The Dataset object is a fundamental tool in PYCHEOPS analysis of *CHEOPS* light curves, and it should be noted that the following dataset based functions can only be run after a Dataset object is created.

`pycheops.Dataset`(*self*, *file_key*, *force_download=False*, *download_all=True*, *configFile=None*, *target=None*, *verbose=True*, *metadata=True*, *view_report_on_download=True*):

Parameters	**file_key**:	String of the file key of the dataset to be analysed
	force_download:	Boolean on if data should be downloaded regardless of the presence of a local file with the same file key
	download_all:	Boolean on downloading all data (light curves, images, and logs) or only light curves
	configFile:	String of the directory of the PYCHEOPS configuration file, if set to "None" the default directory is used
	target:	String of the target name
	verbose:	Boolean on the printing of information to the screen
	metadata:	Boolean on the loading of the metadata
	view_report_on_download:	Boolean on showing the DRP report

pycheops.Dataset.add_glint

A function that creates a smooth spline function that can be used to model the glint or flux artefacts seen periodically with roll angle. This spline function can be fit to the residuals of an eclipse or transit fit, or data outside of a mask provided by the user. By default the glint is fitted as a function of roll angle, but the moon angle may also be used. The number of splines can also be inputted. The fitted spline function of the glint is returned.

`pycheops.Dataset.add_glint`(*self*, *nspline=8*, *mask=None*, *fit_flux=False*, *moon=False*, *angle0=None*, *gapmax=30*, *shot_plot=True*, *binwidth=15*, *figsize=(6,3)*, *fontsize=11*):

Parameters	**nspline**:	Integer number of spline used in the created model
	mask:	Array of Booleans indicating the data to fit
	fit_flux:	Boolean on fitting the flux or residuals
	moon:	Boolean on fitting the data to the moon or roll angle
	angle0:	Integer or float of the roll angle after the gap in units of degrees
	gapmax:	Integer or float of the maximum gap in the data in units of degrees
	show_plot:	Boolean on showing a plot of the fit
	binwidth:	Integer or float of width of each bin for the binned data in units of degrees
	figsize:	Length = 2 list of produced figure size
	fontsize:	Integer or float of figure axes font size
Returns	**glint function**:	A spline function of glint versus roll or moon angle

pycheops.Dataset.animate_frames

A function that produces, saves, and displays animations of the subarrays and imagettes of a dataset by including every tenth frame of the visit, by default. This frequency can be changed by setting the `nframes` argument to the desired value. The minimum and maximum flux levels of the animation can be changed by inputting new scaling factors, `vmin` and `vmax`, and a grid over-plotted. The resulting animations for the subarrays and/or imagettes are save in the current working directory, with the frame cubes used in the animation returned to the user.

`pycheops.Dataset.animate_frames`(*self*, *nframes=10*, *vmin=1.*, *vmax=1.*, *subarray=True*, *imagette=False*, *grid=False*, *aperture=None*, *writer='pillow'*, *figsize=(10,10)*, *fontsize=12*, *linewidth=3*):

Parameters	**nframes**:	Integer value of every n-th frame to be animated
	vmin:	Integer or float of the scaling factor of the minimum flux level
	vmax:	Integer or float of the scaling factor of the maximum flux level
	subarray:	Boolean on animating the subarrays of a dataset
	imagette:	Boolean on animating the subarrays of a dataset
	grid:	Boolean on over-plotting a grid
	aperture:	String selecting the aperture size of the photometry conducted on the observations: "OPTIMAL", "RSUP", "RINF", or "DEFAULT"
	writer:	String of the name of the animation writer to be used
	figsize:	Length = 2 list of produced figure size
	fontsize:	Integer or float of figure axes font size
	linewidth:	Integer or float of aperture line width
Returns	**subarray animation cube**:	The subarray cube used in the animation
	imagette animation cube:	The imagette cube used in the animation

pycheops.Dataset.cds_data_export

A function that produces and saves a CDS uploadable file that includes the raw and detrended light curve of the dataset, if decorrelation was done, along with the instrumental parameters. It allows for the naming of the file, and setting the title, first author, author list, abstract, key words, bibliography code, and acknowledgments relating to an upcoming publication.

`pycheops.Dataset.cds_data_export`(*self*, *lcfile="lc.dat"*, *title=None*, *author=None*, *authors=None*, *abstract=None*, *keywords=None*, *bibcode=None*, *acknowledgments=None*):

Parameters	**lcfile**:	String of the name of the output file
	author:	String of the first author of a related publication
	authors:	String of the author list of a related publication
	abstract:	String of the abstract of a related publication
	keywords:	String of the key words of a related publication
	bibcode:	String of the bibliography code of a related publication
	acknowledgments:	String of the acknowledgments of a related publication

pycheops.Dataset.clip_outliers

A function that calculates the mean absolute deviation (MAD) of the median smoothed light curve and removes outliers from the dataset that are exterior to this value multiplied by an inputted scaling factor with the clipped time, flux, and flux error arrays returned.

pycheops.Dataset.clip_outliers(*self*, *clip=5*, *width=11*, *verbose=True*):

Parameters	**clip**:	Integer or float of MAD scaling factor used for clipping
	width:	Integer or float of the window width for the median-smoothing filter in units of data points
	verbose:	Boolean on the printing of information to the screen
Returns	**time**:	The MAD-clipped time array of the dataset
	flux:	The MAD-clipped flux array of the dataset
	flux error:	The MAD-clipped flux error array of the dataset

pycheops.Dataset.corner_plot

A function that produces a corner plot of the posterior distributions of selected eclipse or transit fitted properties with the option to over-plot prior values and to plot the tick labels with the figure returned.

pycheops.Dataset.corner_plot(*self*, *plotkeys=['T_0', 'D', 'W', 'b']*, *show_priors=True*, *show_ticklabels=False*, *kwargs=None*):

Parameters	**plotkeys**:	Array of eclipse or transit properties to plot or "all"
	show_priors:	Boolean on the plotting of the prior values
	show_ticklabels:	Boolean on the plotting of the tick labels
	kwargs:	Key word arguments to parse to the `corner.corner` function
Returns	**figure**:	A figure of the corner plot

pycheops.Dataset.correct_ramp

A function that corrects the flux of a dataset based on the telescope temperature and a ramp correct factor, `beta`, that depends on the aperture radius used in the photometry. The measured and corrected fluxes can be plotted using the `plot` argument with the correction able to be applied multiple times using the `force` argument. Corrected fluxes are returned to the user, along with the time and flux error arrays, and saved in the `Dataset` object.

pycheops.Dataset.correct_ramp(*self*, *beta=None*, *plot=False*, *force=False*, *figsize=(6,3)*, *fontsize=12*):

Parameters	**beta**:	Float of ramp correct factor
	plot:	Boolean on the plotting of the measured and corrected fluxes
	force:	Boolean on forcing the correction of the ramp
	figsize:	Length = 2 list of produced figure size
	fontsize:	Integer or float of figure axes font size
Returns	**time**:	The ramp corrected time array of the dataset
	flux:	The ramp corrected flux array of the dataset
	flux error:	The ramp corrected flux error array of the dataset

pycheops.Dataset.decorr

A function that decorrelates the flux of a dataset against time, x and y centroid positions, roll angle, contamination, and/or smear using the `lmfit` package and a constructed trend model, plots the fit and decorrelated light curve, and returns the decorrelated flux and flux errors.

`pycheops.Dataset.decorr`(*self*, *dfdt=False*, *df2dt2=False*, *dfdx=False*, *d2fdx2=False*, *dfdy=False*, *d2fdy2=False*, *d2fdxdy=False*, *dfdsinphi=False*, *dfdcosphi=False*, *dfdsin2phi=False*, *dfdcos2phi=False*, *dfdsin3phi=False*, *dfdcos3phi=False*, *dfdbg=False*, *dfdcontam=False*, *dfdsmear=False*):

Parameters	**dfdt**:	Boolean on linearly decorrelating flux against time
	df2dt2:	Boolean on quadratically decorrelating flux against time
	dfdx:	Boolean on linearly decorrelating flux against x centroid position
	d2fdx2:	Boolean on quadratically decorrelating flux against x centroid position
	dfdy:	Boolean on linearly decorrelating flux against y centroid position
	d2fdy2:	Boolean on quadratically decorrelating flux against y centroid position
	d2fdxdy:	Boolean on quadratically decorrelating flux against x and y centroid positions
	dfdsinphi:	Boolean on linearly decorrelating flux against the sine of the roll angle
	dfdcosphi:	Boolean on linearly decorrelating flux against the cosine of the roll angle
	dfdsin2phi:	Boolean on quadratically decorrelating flux against the sine of the roll angle
	dfdcos2phi:	Boolean on quadratically decorrelating flux against the cosine of the roll angle
	dfdsin3phi:	Boolean on cubically decorrelating flux against the sine of the roll angle
	dfdcos3phi:	Boolean on cubically decorrelating flux against the cosine of the roll angle
	dfdbg:	Boolean on linearly decorrelating flux against the background
	dfdcontam:	Boolean on linearly decorrelating flux against the contamination
	dfdsmear:	Boolean on linearly decorrelating flux against the smear
Returns	**flux**:	An array of the decorrelated flux
	flux error:	An array of the flux errors divided by the trend model

pycheops.Dataset.diagnostic_plot

A function that creates a set of eight plots comparing properties of a dataset: time versus flux; roll angle versus flux; time versus background flux; roll angle versus background flux; x centroid position versus flux; y centroid position versus flux; contamination estimate versus flux; smear estimate versus flux; and roll angle versus x and y centroid offsets.

`pycheops.Dataset.diagnostic_plot`(*self*, *fname=None*, *figsize=(8,8)*, *fontsize=10*, *flagged=None*):

Parameters	**fname**:	String of file name used to save figure
	figsize:	Length = 2 list of produced figure size
	fontsize:	Integer or float of figure axes font size
	flagged:	Boolean on comparing data against DRP flagged data

pycheops.Dataset.emcee_report

A function that produces a report of the eclipse or transit fit produced by the PYCHEOPS `emcee_sampler` function that includes the model, fit statistics, model variables, and correlations between the variables.

`pycheops.Dataset.emcee_report`(*self*, ***kwargs*):

Parameters	****kwargs**:	Key word arguments to parse to the `lmfit.fit_report` function
Returns	**report**:	A report of the eclipse or transit fit

pycheops.Dataset.emcee_sampler

A function that samples the posterior probability distributions of eclipse or transit fitting parameters using the Python `emcee` package with the number of sampling and burn-in steps, walkers, and samples to be thin inputted. The shot noise of the observations can be modelled by constructing shot and jitter term kernel using the Python `celerite2` package. The posterior probability distributions are returned to the user.

`pycheops.Dataset.emcee_sampler`(*self*, *params=None*, *steps=128*, *nwalkers=64*, *burn=256*, *thin=1*, *log_sigma=None*, *add_shoterm=False*, *log_omega0=None*, *log_S0=None*, *log_Q=None*, *init_scale=1e-2*, *progress=True*):

Parameters	**params**:	Dictionary of fit parameter priors, if set = None and a `lmfit` function was run previously then this dictionary will be taken from the previous fit
	steps:	Integer number of sampling steps for the MCMC to perform
	nwalkers:	Integer number of walkers in the MCMC
	burn:	Integer number of burn-in steps for the MCMC to perform
	thin:	Integer number of n-th sampled value to be thinned
	log_sigma:	Logarithm of sigma of the jitter term of the kernel
	add_shoterm:	Boolean on whether to included modelled shot noise to the sampler
	log_omega0:	Logarithm of omega0 of the shot-term kernel
	log_S0:	Logarithm of S0 of the shot-term kernel
	log_Q:	Logarithm of Q of the shot-term kernel
	init_scale:	Float of the initial scale of steps to be taken
	progress:	Boolean on printing the progress of the sampler
Returns	**result**:	The result of the MCMC fit to the data

pycheops.Dataset.flatten

A function that normalises a dataset using a polynomial fit of order to be inputted with the option to include a mask centre and width that can be used to avoid normalisation of a section of the light curve. The time, flux, and flux error arrays of the dataset are returned.

`pycheops.Dataset.flatten`(*self*, *mask_centre*, *mask_width*, *npoly=2*):

Parameters	**mask_centre**:	Integer or float of the time at the mask centre
	mask_width:	Integer or float of the mask width in the same units as the time array
	npoly:	Integer of the polynomial order used to normalise the data
Returns	**time**:	The normalised time array of the dataset
	flux:	The normalised flux array of the dataset
	flux error:	The normalised flux error array of the dataset

pycheops.Dataset.get_lightcurve

A function that extracts the light curve data of the dataset from a pre-downloaded .tgz file, decontaminates the data of any nearby sources, and returns it in the form of an astropy table or three arrays.

`pycheops.Dataset.get_lightcurve`(*self*, *aperture=None*, *decontaminate=None*, *returnTable=False*, *reject_highpoints=True*, *verbose=True*):

Parameters	**aperture**:	String selecting the aperture size of the photometry conducted on the observations: "OPTIMAL", "RSUP", "RINF", or "DEFAULT"
	decontaminate	Boolean on the subtraction of contaminating flux from background sources
	returnTable:	Boolean on returning a table of the light curve data or the time, flux, and flux error of the observations
	reject_highpoints:	Boolean on cutting high flux points from the light curve
	verbose:	Boolean on the printing of information to the screen
Returns	**table**:	An astropy table of the light curve data returned if returnTable=True
	time:	An array of the time of the light curve returned if returnTable=False
	flux:	An array of the flux of the light curve returned if returnTable=False
	flux_err:	An array of the flux error of the light curve returned if returnTable=False

pycheops.Dataset.lmfit_eclipse

A function that fits an eclipse in a dataset using a constructed eclipse model and parameters described below, via a least-squares fitting method with the options to decorrelate the flux of a dataset against time, x and y centroid positions, roll angle, background, contamination, and/or smear. The eclipse parameters and decorrelation trends can be input in an integer or float value, length = 2 list of upper and lower limit values, ufloat value and uncertainty object, or `lmfit.parameter` value, minimum, and maximum object.

`pycheops.Dataset.lmfit_eclipse`(*self*, *T_0=None*, *P=None*, *D=None*, *W=None*, *b=None*, *L=None*, *f_c=None*, *f_s=None*, *a_c=None*, *dfdbg=None*, *dfdcontam=None*, *dfdsmear=None*, *ramp=None*, *c=None*, *dfdx=None*, *dfdy=None*, *d2fdx2=None*, *d2fdy2=None*, *dfdsinphi=None*, *dfdcosphi=None*, *dfdsin2phi=None*, *dfdcos2phi=None*, *dfdsin3phi=None*, *dfdcos3phi=None*, *dfdt=None*, *d2fdt2=None*, *glint_scale=None*, *log_sigma=None*):

Parameters	**T_0**:	Transit centre time (days)
	P:	Orbital period (days)
	D:	Transit depth (0.0-1.0))
	W:	Transit width (phase)
	b:	Impact parameter
	L:	Eclipse depth (0.0-1.0)
	f_c:	Orbital eccentricity and longitude of periastron component
	f_s:	Orbital eccentricity and longitude of periastron component
	a_c:	Light travel time (days)
	dfdbg:	Linear flux against the background
	dfdcontam:	Linear flux against the contamination
	dfdsmear:	Linear flux against the smear
	ramp:	Linear flux against the telescope temperature
	c:	Flux scaling factor (set = 1 by default)
	dfdx:	Linear flux against x centroid position trend
	dfdy:	Linear flux against y centroid position trend
	d2fdx2:	Quadratic flux against x centroid position trend
	d2fdy2:	Quadratic flux against y centroid position trend
	dfdsinphi:	Linear flux against the sine of the roll angle trend
	dfdcosphi:	Linear flux against the cosine of the roll angle trend
	dfdsin2phi:	Quadratic flux against the sine of the roll angle trend
	dfdcos2phi:	Quadratic flux against the cosine of the roll angle trend
	dfdsin3phi:	Cubic flux against the sine of the roll angle trend
	dfdcos3phi:	Cubic flux against the cosine of the roll angle trend
	dfdt:	Linear flux against time trend
	df2dt2:	Quadratic flux against time trend
	glint_scale:	Glint model scaling factor
	log_sigma:	Logarithm of a Gaussian white noise term
Returns	**result**:	The result of the least-squares eclipse fit to the data

pycheops.Dataset.lmfit_report

A function that produces a report of the eclipse or transit fit produced by the PYCHEOPS `lmfit_eclipse` or `lmfit_transit` functions that includes the model, fit statistics, model variables, and correlations between the variables.

`pycheops.Dataset.lmfit_report`(*self*, ***kwargs*):

Parameters	****kwargs**:	Key word arguments to parse to the `lmfit.fit_report` function
Returns	**report**:	A report of the eclipse or transit fit

pycheops.Dataset.lmfit_transit

A function that fits a transit in a dataset using a transit model constructed with the power-2 limb-darkening law and transit parameters described below, via a least-squares fitting method with the options to decorrelate the flux of a dataset against time, x and y centroid positions, roll angle, background, and/or contamination. The transit parameters and decorrelation trends can be input in an integer or float value, length = 2 list of upper and lower limit values, ufloat value and uncertainty object, or `lmfit.parameter` value, minimum, and maximum object.

`pycheops.Dataset.lmfit_transit`(*self*, *T_0=None*, *P=None*, *D=None*, *W=None*, *b=None*, *f_c=None*, *f_s=None*, *h_1=None*, *h_2=None*, *c=None*, *dfdbg=None*, *dfdcontam=None*, *dfdsmear=None*, *ramp=None*, *dfdx=None*, *dfdy=None*, *d2fdx2=None*, *d2fdy2=None*, *dfdsinphi=None*, *dfdcosphi=None*, *dfdsin2phi=None*, *dfdcos2phi=None*, *dfdsin3phi=None*, *dfdcos3phi=None*, *dfdt=None*, *d2fdt2=None*, *glint_scale=None*, *logrhoprior=None*, *log_sigma=None*):

Parameters	**T_0**:	Transit centre time (days)
	P:	Orbital period (days)
	D:	Transit depth (0.0-1.0))
	W:	Transit width (phase)
	b:	Impact parameter
	f_c:	Orbital eccentricity and longitude of periastron component
	f_s:	Orbital eccentricity and longitude of periastron component
	h_1:	First limb-darkening coefficient
	h_2:	Second limb-darkening coefficient
	c:	Flux scaling factor (set = 1 by default)
	dfdbg:	Linear flux against the background
	dfdcontam:	Linear flux against the contamination
	dfdsmear:	Linear flux against the smear
	ramp:	Linear flux against the telescope temperature
	dfdx:	Linear flux against x centroid position trend
	dfdy:	Linear flux against y centroid position trend
	d2fdx2:	Quadratic flux against x centroid position trend
	d2fdy2:	Quadratic flux against y centroid position trend
	dfdsinphi:	Linear flux against the sine of the roll angle trend
	dfdcosphi:	Linear flux against the cosine of the roll angle trend
	dfdsin2phi:	Quadratic flux against the sine of the roll angle trend
	dfdcos2phi:	Quadratic flux against the cosine of the roll angle trend
	dfdsin3phi:	Cubic flux against the sine of the roll angle trend
	dfdcos3phi:	Cubic flux against the cosine of the roll angle trend
	dfdt:	Linear flux against time trend
	df2dt2:	Quadratic flux against time trend
	glint_scale:	Glint model scaling factor
	logrhoprior:	Logarithm of stellar density (solar units)
	log_sigma:	Logarithm of a Gaussian white noise term
Returns	**result**:	The result of the least-squares transit fit to the data

pycheops.Dataset.load

A function that loads a pickle file of a previously save `Dataset` object.

`pycheops.Dataset.load`(*self*, *filename*):

Parameters	**filename**:	A string of the filename of the saved pickle file
Returns	**dataset**:	The saved `Dataset` object

pycheops.Dataset.mask_data

A function that removes sections of the dataset that are indicated by an array of Booleans to be inputted by the user where True means data should be masked.

`pycheops.Dataset.mask_data`(*self*, *mask*, *verbose=True*):

Parameters	**mask**:	Boolean array indicating the data to be masked
	verbose:	Boolean on the printing of information to the screen
Returns	**time**:	The masked time array of the dataset
	flux:	The masked flux array of the dataset
	flux error:	The masked flux error array of the dataset

pycheops.Dataset.massradius

A function that calculates the mass, radius, and density of the host star and target planet, alongside mass ratio, semi-major axis, and planetary surface gravity. The function takes user inputs for stellar mass and radius as well as the semi-amplitude of the planetary orbit and the mass ratio. These values can be input in an integer or float value, length = 2 list of upper and lower limit values, ufloat value and uncertainty object, or `lmfit.parameter` value, minimum, and maximum object.

A plot of the planetary mass versus radius can be produced with theoretical internal structure models over-plotted and is returned with a dictionary of the calculated stellar, planetary, and orbital properties.

`pycheops.Dataset.massradius`(*self*, *m_star=None*, *r_star=None*, *K=None*, *q=0*, *jovian=True*, *plot_kws=None*, *return_samples=False*, *verbose=True*):

Parameters	**m_star**:	Host star mass in solar units
	r_star:	Host star radius in solar units
	K:	Planet orbit semi-amplitude in units of m/s
	q:	Planet to star mass ratio
	jovian:	Boolean on printing the values relative to Jupiter or Earth
	plot_kws:	Dictionary detailing properties of the produced plot, such as over-plotted models and plot title
	return_samples:	Boolean on returning the calculated posterior samples in the result dictionary
	verbose:	Boolean on the printing of information to the screen
Returns	**result**	Dictionary of the determined stellar and planetary properties
	fig:	Figure of the mass versus radius plot

pycheops.Dataset.planet_check

A function that computes the separation of the target to the Moon, Mars, Jupiter, Saturn, Uranus, and Neptune, and prints the values in degrees.

`pycheops.Dataset.planet_check`(*self*):

pycheops.Dataset.plot_emcee

A function that creates plots of the light curve of the dataset over-laid with the eclipse or transit fit produced by the `emcee_sampler` function, and the residuals to the fit.

`pycheops.Dataset.plot_emcee`(*self*, *title=None*, *nsamples=32*, *detrend=False*, *binwidth=0.01*, *show_model=True*, *figsize=(6,4)*, *fontsize=11*):

Parameters	**title**:	String of the plot title
	nsamples:	Integer number of parameter sets from the MCMC produced posterior distribution to be plotted
	detrend:	Boolean on conducting a separate detrending of the dataset
	binwidth:	Integer or float of width of each bin for the binned data in units of days
	show_model:	Boolean on plotting the fitted model
	figsize:	Length = 2 list of produced figure size
	fontsize:	Integer or float of figure axes font size
Returns	**fig**:	The figure of the eclipse or transit fit and residual plots

pycheops.Dataset.plot_fft

A function that conducts a fast Fourier transform of the raw and Gaussian smoothed residuals to an eclipse or transit fit and returns a figure of the Lomb-Scargle power spectrum that includes estimates of the maximum stellar variability frequency based on stellar properties.

`pycheops.Dataset.plot_fft`(*self*, *star=None*, *gsmooth=5*, *logxlim=(1.5,4.5)*, *title=None*, *fontsize=12*, *figsize=(8,5)*):

Parameters	**star**:	A StarProperties object of the target
	gmsooth:	Integer value that determines the width of the Gaussian kernel used for smoothing the data in units of datapoints
	logxlim:	Length = 2 list of the x-axis limits
	title:	String of the plot title
	fontsize:	Integer or float of figure axes font size
	figsize:	Length = 2 list of produced figure size
Returns	**fig**:	The figure of the Lomb-Scargle power spectrum plot

pycheops.Dataset.plot_lmfit

A function that creates plots of the light curve of the dataset over-laid with the eclipse or transit fit produced by the `lmfit_eclipse` or `lmfit_transit` functions, and the residuals to the fit.

`pycheops.Dataset.plot_lmfit`(*self*, *figsize=(6,4)*, *fontsize=11*, *title=None*, *show_model=True*, *binwidth=0.01*, *detrend=False*):

Parameters	**figsize**:	Length = 2 list of produced figure size
	fontsize:	Integer or float of figure axes font size
	title:	String of the plot title
	show_model:	Boolean on plotting the fitted model
	binwidth:	Integer or float of width of each bin for the binned data in units of days
	detrend:	Boolean on conducting a separate detrending of the dataset
Returns	**fig**:	The figure of the eclipse or transit fit and residual plots

pycheops.Dataset.rollangle_plot

A function that plots the residuals of a prior eclipse or transit fit against roll angle with the fitted glint model over-plotted if previously applied. If a decorrelation against Moon angle has been done this is also shown. A figure of the plots is returned.

`pycheops.Dataset.rollangle_plot`(*self*, *binwidth=15*, *figsize=None*, *fontsize=11*, *title=None*):

Parameters	**binwidth**:	Integer or float of width of each bin for the binned data in units of degrees
	figsize:	Length = 2 list of produced figure size
	fontsize:	Integer or float of figure axes font size
	title:	String of the plot title
Returns	**fig**:	Figure of the roll angle versus residuals plots and glint model fit if applied

pycheops.Dataset.save

A function that saves the current `Dataset` object to a pickle file in the current working directory.

`pycheops.Dataset.save`(*self*):

pycheops.Dataset.should_I_decorr

A function that fits combinations of trends in flux versus time, x and y centroid positions, roll angle, background, and contamination for a dataset and assess whether the dataset needs to be decorrelated. This is done by calculating the Bayesian Information Criteria (BIC) of each combination under the assumption that the combination that induces the lowest BIC best describes any trends. This combination is returned to the user along with the BIC value. Users have the option to mask out regions of the light curve in order to avoid trend fitting over features such as eclipses or transits.

`pycheops.Dataset.should_I_decorr`(*self*, *mask_centre=0*, *mask_width=0*):

Parameters	**mask_centre**:	Integer or float of the time at the mask centre
	mask_width:	Integer or float of the mask width in the same units as the time array
Returns	**min_BIC**:	A float of the minimum BIC produced by the trend fitting
	decorr_params:	A list of the parameters which should be decorrelated against

pycheops.Dataset.trail_plot

A function that shows the chains of the MCMC parameters from the eclipse or transit fit with the parameter values against step number plotted. Users can define the parameters to plot or choose "all". A figure of the chains is returned.

`pycheops.Dataset.trail_plot`(*self*, *plotkeys=['T_0', 'D', 'W', 'b']*, *width=8*, *height=1.5*):

Parameters	**plotkeys**:	Array of eclipse or transit properties to plot or "all"
	width:	Integer or float of the subplot width
	height:	Integer or float of the subplot height
Returns	**fig**:	Figure of the MCMC trails for the conducted eclipse or transit fit

pycheops.Dataset.transit_noise_plot

A function that calculates the transit noise of a dataset by inserting a simulated transit of inputted width into the light curve, sliding along the light curve, and determining the depth at which the S/N = 1 at each step with the uncertainties on the data calculated using two methods; scaled error and minimum error. The light curve and transit noise estimates are subsequently plotted, with the noises returned to the users as a dictionary if return_values is set to True.

`pycheops.Dataset.transit_noise_plot`(*self*, *width=3*, *steps=500*, *fname=None*, *figsize=(6,4)*, *fontsize=11*, *return_values=False*, *requirement=None*, *local=False*, *verbose=True*):

Parameters	**width**:	Integer or float of transit width (hours) of simulated inserted transit
	steps:	Integer number of transit noise calculations to be conducted
	fname:	String of file name used to save figure
	figsize:	Length = 2 list of produced figure size
	fontsize:	Integer or float of figure axes font size
	return_values:	Boolean on returning the calculated noises in the result dictionary
	requirement:	Integer or float of required noise level to be plotted
	local:	Boolean on using data near the inserted transit centre time
	verbose:	Boolean on the printing of information to the screen
Returns	**d**:	Dictionary of calculated noises if return_values is set to True

pycheops.Dataset.view_report

A function that shows the DRP report of the *CHEOPS* visit contained in the Dataset object with the option to change the PDF viewer used.

`pycheops.Dataset.view_report`(*self*, *pdf_cmd=None*, *configFile=None*):

Parameters	**pdf_cmd**:	String of the command to launch a PDF
	configFile:	String of the file location and name of the PYCHEOPS configuration file

pycheops.funcs.transit_width

A function that takes the stellar radius to planet semi-major axis ratio (R_*/a), planetary-to-stellar radii ratio (R_p/R_*), impact parameter (b), and orbital period (P), and calculates and returns the transit width in the same units as the inputted period.

`pycheops.funcs.transit_width`(*r*, *k*, *b*, *P=1*):

Parameters	**r**:	Integer or float of the stellar radius to planet semi-major axis ratio
	k:	Integer or float of the planetary-to-stellar radii ratio
	b:	Integer or float of the impact parameter
	P:	Integer or float of the planet orbital period
Returns	**width**:	Float value of the calculated transit width in the same units as the inputted orbital period

pycheops.ld.stagger_power2_interpolator

A class that creates an object that can be used to determine the parameters of the power-2 limb-darkening law that are interpolated from the Stagger grid of models based upon stellar parameters (`T_eff`, `log_g`, `Fe_H`) given to the created object.

`pycheops.ld.stagger_power2_interpolator`(*self*, *passband='CHEOPS'*):

Parameters	**passband**:	String of the spacecraft, instrument, or passband name
Returns	**interpolated grid**:	List of limb-darkening coefficients (c, alpha, h_1, h_2)

pycheops.models.EclipseModel

A function that constructs an eclipse model used to fit data using the Python `lmfit` package.

`pycheops.models.EclipseModel`(*self*, *independent_vars=['t']*, *prefix="*, *nan_policy='raise'*, ***kwargs*):

Parameters	**independent_vars**:	A list of independent variables to build the model against, set to time by default
	prefix:	String to append to the beginning of model name
	nan_policy:	String of the policy of NaN values when fitting the model
	****kwargs**:	Additional keyword arguments, such as parameters used to build the model, values, and constraints
Returns	**model**:	The produced eclipse model

pycheops.models.EBLMModel

A function that constructs an eclipse and transit model used to fit data using the Python `lmfit` package.

`pycheops.models.EBLMModel`(*self*, *independent_vars=['t']*, *prefix="*, *nan_policy='raise'*, ***kwargs*):

Parameters	**independent_vars**:	A list of independent variables to build the model against, set to time by default
	prefix:	String to append to the beginning of model name
	nan_policy:	String of the policy of NaN values when fitting the model
	****kwargs**:	Additional keyword arguments, such as parameters used to build the model, values, and constraints
Returns	**model**:	The produced eclipse and transit model

pycheops.models.FactorModel

A function that constructs a factor model used to detrend data using the Python `lmfit` package.

`pycheops.models.FactorModel`(*self*, *independent_vars=['t']*, *prefix="*, *nan_policy='raise'*, *dx=None*, *dy=None*, *sinphi=None*, *cosphi=None*, *bg=None*, *contam=None*, *smear=None*, *deltaT=None*, ***kwargs*):

Parameters	**independent_vars**:	A list of independent variables to build the model against, set to time by default
	prefix:	String to append to the beginning of model name
	nan_policy:	String of the policy of NaN values when fitting the model
	dx:	A list of the centroid x position over the dataset
	dy:	A list of the centroid y position over the dataset
	sinphi:	A list of the sine of the roll angle over the dataset
	cosphi:	A list of the cosine of the roll angle over the dataset
	bg:	A list of the background over the dataset
	contam:	A list of the contamination over the dataset
	smear:	A list of the smear over the dataset
	deltaT:	A list of the change in temperature over the dataset
	****kwargs**:	Additional keyword arguments, such as parameters used to build the model, values, and constraints
Returns	**model**:	The produced factor model

pycheops.models.PlanetModel

A function that constructs an eclipse, transit, and thermal phase curve model used to fit data using the Python `lmfit` package.

`pycheops.models.PlanetModel`(*self*, *independent_vars=['t']*, *prefix="*, *nan_policy='raise'*, ***kwargs*):

Parameters	**independent_vars**:	A list of independent variables to build the model against, set to time by default
	prefix:	String to append to the beginning of model name
	nan_policy:	String of the policy of NaN values when fitting the model
	****kwargs**:	Additional keyword arguments, such as parameters used to build the model, values, and constraints
Returns	**model**:	The produced eclipse, transit, and thermal phase curve model

pycheops.models.ThermalPhaseModel

A function that constructs a thermal phase curve model used to fit data using the Python `lmfit` package.

`pycheops.models.ThermalPhaseModel`(*self*, *independent_vars=['t']*, *prefix="*, *nan_policy='raise'*,

***kwargs*):

Parameters	**independent_vars**:	A list of independent variables to build the model against, set to time by default
	prefix:	String to append to the beginning of model name
	nan_policy:	String of the policy of NaN values when fitting the model
	****kwargs**:	Additional keyword arguments, such as parameters used to build the model, values, and constraints
Returns	**model**:	The produced thermal phase curve model

pycheops.models.TransitModel

A function that constructs a transit model used to fit data using the Python `lmfit` package.

`pycheops.models.TransitModel`(*self*, *independent_vars=['t']*, *prefix="*, *nan_policy='raise'*, ***kwargs*):

Parameters	**independent_vars**:	A list of independent variables to build the model against, set to time by default
	prefix:	String to append to the beginning of model name
	nan_policy:	String of the policy of NaN values when fitting the model
	****kwargs**:	Additional keyword arguments, such as parameters used to build the model, values, and constraints
Returns	**model**:	The produced transit model

pycheops.MultiVisit

A class that creates a MultiVisit object that loads multiple saved Dataset objects of the same target and runs StarProperties to retrieve stellar parameters of the host star. The individual light curves can then be decorrelated separately and have transits, eclipses, or both fitted simultaneously using fitting routines that use the Python `emcee` package. Plotting functions can be used to view and assess the fits.

`pycheops.MultiVisit`(*self*, *target=None*, *datadir=None*, *ident=None*, *id_kws={'dace':True}*, *verbose=True*):

Parameters	**target**:	String of the target name
	datadir:	String of the directory of the saved dataset pickle files
	ident:	String of the target identifier in the table retrieved by `StarProperties`
	id_kws:	Dictionary of keywords to pass to `StarProperties`
	verbose:	Boolean on the printing of information to the screen

pycheops.MultiVisit.corner_plot

A function that produces a corner plot of the posterior distributions of selected eclipse or transit fitted properties with the option to over-plot prior values and to plot the tick labels with the figure returned.

`pycheops.MultiVisit.corner_plot`(*self*, *plotkeys=None*, *show_priors=True*, *show_ticklabels=False*, *kwargs=None*):

Parameters	**plotkeys**:	Array of eclipse or transit properties to plot or "all"
	show_priors:	Boolean on the plotting of the prior values
	show_ticklabels:	Boolean on the plotting of the tick labels
	kwargs:	Key word arguments to parse to the `corner.corner` function
Returns	**figure**:	A figure of the corner plot

pycheops.MultiVisit.fit_eclipse

A function that samples the posterior probability distributions of eclipse fitting to multiple datasets with models constructed defined by given parameters using the Python `emcee` package with the number of sampling and burn-in steps, walkers, and samples to be thin inputted. The eclipse parameters and extra priors can be input in an integer or float value, length = 2 list of upper and lower limit values, or ufloat value and uncertainty object. Decorrelation against roll angle is done automatically by default and there are options to include the fitting of eclipse depth variation (EDV). The stellar noise of the observations can be modelled by constructing shot and jitter term kernel using the Python `celerite2` package. The posterior probability distributions are returned to the user.

`pycheops.MultiVisit.fit_eclipse`(*self*, *steps=128*, *nwalkers=64*, *burn=256*, *T_0=None*, *P=None*, *D=None*, *W=None*, *b=None*, *L=None*, *f_c=None*, *f_s=None*, *a_c=None*, *edv=False*, *edv_prior=1e-3*, *extra_priors=None*, *log_sigma_w=None*, *log_omega0=None*, *log_S0=None*, *log_Q=None*, *unroll=True*, *nroll=3*, *unwrap=False*, *thin=1*, *init_scale=1e-2*, *progress=True*):

Parameters	**steps**:	Integer number of sampling steps for the MCMC to perform
	nwalkers:	Integer number of walkers in the MCMC
	burn:	Integer number of burn-in steps for the MCMC to perform
	T_0:	Transit centre time (days)
	P:	Orbital period (days)
	D:	Transit depth (0.0-1.0))
	W:	Transit width (phase)
	b:	Impact parameter
	L:	Eclipse depth (0.0-1.0)
	f_c:	Orbital eccentricity and longitude of periastron component
	f_s:	Orbital eccentricity and longitude of periastron component
	a_c:	Light travel time (days)
	edv:	Boolean on whether to conduct fitting of EDV
	edv_prior:	Float of the range of EDVs to probe
	extra_priors:	Dictionary of additional parameters to include in the fitting
	log_sigma_w:	Logarithm of sigma of the jitter term of the kernel
	log_omega0:	Logarithm of omega0 of the shot-term kernel
	log_S0:	Logarithm of S0 of the shot-term kernel
	log_Q:	Logarithm of Q of the shot-term kernel
	unroll:	Boolean on whether to automatically decorrelate against roll angle
	nroll:	Integer of the roll angle frequency order to decorrelate up to
	unwrap:	Boolean on whether to first decorrelate against `Dataset` derived roll angle
	thin:	Integer of the factor of samples to be removed from the MCMC
	init_scale:	Float of the initial scale of steps to be taken
	progress:	Boolean on printing the progress of the sampler
Returns	**result**:	The result of the MCMC fit to the data

pycheops.MultiVisit.fit_eblm

A function that samples the posterior probability distributions of simultaneous eclipse and transit fitting to multiple datasets with models constructed defined by given parameters using the Python `emcee` package with the number of sampling and burn-in steps, walkers, and samples to be thin inputted. The eclipse and transit parameters and extra priors can be input in an integer or float value, length = 2 list of upper and lower limit values, or ufloat value and uncertainty object. Decorrelation against roll angle is done automatically by default and there are options to include the fitting of eclipse depth variation (EDV) and transit timing variation (TTV). The stellar noise of the observations can be modelled by constructing shot and jitter term kernel using the Python `celerite2` package. The posterior probability distributions are returned to the user.

`pycheops.MultiVisit.fit_eblm`(*self*, *steps=128*, *nwalkers=64*, *burn=256*, *T_0=None*, *P=None*, *D=None*, *W=None*, *b=None*, *h_1=None*, *h_2=None*, *ttv=False*, *ttv_prior=3600*, *L=None*, *a_c=None*, *edv=False*, *edv_prior=1e-3*, *extra_priors=None*, *log_sigma_w=None*, *log_omega0=None*, *log_S0=None*, *log_Q=None*, *unroll=True*, *nroll=3*, *unwrap=False*, *thin=1*, *init_scale=1e-2*, *progress=True*):

Parameters	**steps**:	Integer number of sampling steps for the MCMC to perform
	nwalkers:	Integer number of walkers in the MCMC
	burn:	Integer number of burn-in steps for the MCMC to perform
	T_0:	Transit centre time (days)
	P:	Orbital period (days)
	D:	Transit depth (0.0-1.0))
	W:	Transit width (phase)
	b:	Impact parameter
	h_1:	First limb-darkening coefficient
	h_2:	Second limb-darkening coefficient
	ttv:	Boolean on whether to conduct fitting of TTV
	ttv_prior:	Float of the range of TTVs to probe
	L:	Eclipse depth (0.0-1.0)
	a_c:	Light travel time (days)
	edv:	Boolean on whether to conduct fitting of EDV
	edv_prior:	Float of the range of EDVs to probe
	extra_priors:	Dictionary of additional parameters to include in the fitting
	log_sigma_w:	Logarithm of sigma of the jitter term of the kernel
	log_omega0:	Logarithm of omega0 of the shot-term kernel
	log_S0:	Logarithm of S0 of the shot-term kernel
	log_Q:	Logarithm of Q of the shot-term kernel
	unroll:	Boolean on whether to automatically decorrelate against roll angle
	nroll:	Integer of the roll angle frequency order to decorrelate up to
	unwrap:	Boolean on whether to first decorrelate against `Dataset` derived roll angle
	thin:	Integer of the factor of samples to be removed from the MCMC
	init_scale:	Float of the initial scale of steps to be taken
	progress:	Boolean on printing the progress of the sampler
Returns	**result**:	The result of the MCMC fit to the data

pycheops.MultiVisit.fit_report

A function that produces a report of the eclipse or transit fit produced by the `fit_eclipse`, `fit_eblm`, or `fit_transit` functions that includes the model, fit statistics, model variables, and correlations between the variables.

`pycheops.MultiVisit.fit_report`(*self*, ***kwargs*):

Parameters	****kwargs**:	Key word arguments to parse to the `lmfit.fit_report` function
Returns	**report**:	A report of the eclipse or transit fit

pycheops.MultiVisit.fit_transit

A function that samples the posterior probability distributions of transit fitting to multiple datasets with models constructed defined by given parameters using the Python `emcee` package with the number of sampling and burn-in steps, walkers, and samples to be thin inputted. The transit parameters and extra priors can be input in an integer or float value, length = 2 list of upper and lower limit values, or ufloat value and uncertainty object. Decorrelation against roll angle is done automatically by default and there are options to include the fitting of transit timing variation (TTV). The stellar noise of the observations can be modelled by constructing shot and jitter term kernel using the Python `celerite2` package. The posterior probability distributions are returned to the user.

`pycheops.MultiVisit.fit_transit`(*self*, *steps=128*, *nwalkers=64*, *burn=256*, *T_0=None*, *P=None*, *D=None*, *W=None*, *b=None*, *f_c=None*, *f_s=None*, *h_1=None*, *h_2=None*, *ttv=False*, *ttv_prior=3600*, *extra_priors=None*, *log_sigma_w=None*, *log_omega0=None*, *log_S0=None*, *log_Q=None*, *unroll=True*, *nroll=3*, *unwrap=False*, *thin=1*, *init_scale=1e-2*, *progress=True*):

Parameters	**steps**:	Integer number of sampling steps for the MCMC to perform
	nwalkers:	Integer number of walkers in the MCMC
	burn:	Integer number of burn-in steps for the MCMC to perform
	T_0:	Transit centre time (days)
	P:	Orbital period (days)
	D:	Transit depth (0.0-1.0))
	W:	Transit width (phase)
	b:	Impact parameter
	f_c:	Orbital eccentricity and longitude of periastron component
	f_s:	Orbital eccentricity and longitude of periastron component
	h_1:	First limb-darkening coefficient
	h_2:	Second limb-darkening coefficient
	ttv:	Boolean on whether to conduct fitting of TTV
	ttv_prior:	Float of the range of TTVs to probe
	extra_priors:	Dictionary of additional parameters to include in the fitting
	log_sigma_w:	Logarithm of sigma of the jitter term of the kernel
	log_omega0:	Logarithm of omega0 of the shot-term kernel
	log_S0:	Logarithm of S0 of the shot-term kernel
	log_Q:	Logarithm of Q of the shot-term kernel
	unroll:	Boolean on whether to automatically decorrelate against roll angle
	nroll:	Integer of the roll angle frequency order to decorrelate up to
	unwrap:	Boolean on whether to first decorrelate against `Dataset` derived roll angle
	thin:	Integer of the factor of samples to be removed from the MCMC
	init_scale:	Float of the initial scale of steps to be taken
	progress:	Boolean on printing the progress of the sampler
Returns	**result**:	The result of the MCMC fit to the data

pycheops.MultiVisit.massradius

A function that calculates the mass, radius, and density of the host star and target planet, alongside mass ratio, semi-major axis, and planetary surface gravity. The function takes user inputs for stellar mass and radius as well as the semi-amplitude of the planetary orbit and the mass ratio. These values can be input in an integer or float value, length = 2 list of upper and lower limit values, ufloat value and uncertainty object, or `lmfit.parameter` value, minimum, and maximum object.

A plot of the planetary mass versus radius can be produced with theoretical internal structure models over-plotted and is returned with a dictionary of the calculated stellar, planetary, and orbital properties.

`pycheops.MultiVisit.massradius`(*self*, *m_star=None*, *r_star=None*, *K=None*, *q=0*, *jovian=True*, *return_samples=False*, *plot_kws=None*, *verbose=True*):

Parameters	**m_star**:	Host star mass in solar units
	r_star:	Host star radius in solar units
	K:	Planet orbit semi-amplitude in units of m/s
	q:	Planet to star mass ratio
	jovian:	Boolean on printing the values relative to Jupiter or Earth
	plot_kws:	Dictionary detailing properties of the produced plot, such as over-plotted models and plot title
	return_samples:	Boolean on returning the calculated posterior samples in the result dictionary
	verbose:	Boolean on the printing of information to the screen
Returns	**result**	Dictionary of the determined stellar and planetary properties
	fig:	Figure of the mass versus radius plot

pycheops.MultiVisit.plot_fit

A function that plots the data and model of fitted eclipses or transits for multiple visits and the residuals to those fits with options to bin and offset the data, and avoid plotting the model over gaps in the data.

`pycheops.MultiVisit.plot_fit`(*self*, *title=None*, *detrend=False*, *binwidth=0.001*, *add_gaps=True*, *gap_tol=0.005*, *renorm=True*, *data_offset=None*, *res_offset=None*, *phase0=None*, *xlim=None*, *data_ylim=None*, *res_ylim=None*, *figsize=None*, *fontsize=12*):

Parameters	**title**:	String of the plot title
	detrend:	Boolean on conducting a separate detrending of the dataset
	binwidth:	Integer or float of width of each bin for the binned data in units of days
	add_gaps	Boolean on over-plotting the fitted model on gaps in the data
	gap_tol	Integer or float of the maximum gap in the data that should be over-plotted with the model in units of days
	renorm	Boolean on renormalising all individual datasets to 1
	data_offset	Float value of flux offset to be applied between individual fitted datasets
	res_offset	Float value of flux offset to be applied between individual sets of residuals
	phase0	Float value of the phase from which each individual dataset is plotted
	xlim	Length = 2 list of the x-axis limits
	data_ylim	Length = 2 list of the y-axis limits of the data
	res_ylim	Length = 2 list of the y-axis limits of the residuals
	figsize:	Length = 2 list of produced figure size
	fontsize:	Integer or float of figure axes font size
Returns	**fig**:	The figure of the eclipses or transits fit and residual plots

pycheops.MultiVisit.trail_plot

A function that shows the chains of the MCMC parameters from the eclipse or transit fit with the parameter values against step number plotted. Users can define the parameters to plot or choose "all". A figure of the chains is returned.

`pycheops.MultiVisit.trail_plot`(*self*, *plotkeys=None*, *plot_kws={'alpha':0.1} width=8*, *height=1.5*):

Parameters	**plotkeys**:	Array of eclipse or transit properties to plot or "all"
	plot_kws:	Dictionary detailing properties of the produced plot, such as marker size, shape, and colour
	width:	Integer or float of the subplot width
	height:	Integer or float of the subplot height
Returns	**fig**:	Figure of the MCMC trails for the conducted eclipse or transit fit

pycheops.MultiVisit.ttv_plot

A function that plots the fitted TTVs from the `fit_transit` function against the centre time of each transit in the `MultiVisit` object.

`pycheops.MultiVisit.ttv_plot`(*self*, *plot_kws=None*, *figsize=(8,5)*):

Parameters	**plot_kws**:	Dictionary detailing properties of the produced plot that is passed to `plt.errorbar`
	figsize:	Length = 2 list of produced figure size
Returns	**fig**:	Figure of the fitted TTVs

pycheops.MultiVisit.tzero

A function that calculates the transit centre time closest to the middle of the multiple visit time series given a provided centre time of an individual transit in BJD and the planet period in days.

`pycheops.MultiVisit.tzero`(*self*, *BJD_0*, *P*):

Parameters	**BJD_0**:	Float or integer of a transit centre time in BJD
	P:	Float or integer of the period of the planet (days)
Returns	**mid-transit time**:	Float of the transit centre time closest to the middle of the time series

pycheops.PlanetProperties

A class that creates an object that can be utilised to retrieve planetary parameter values from DACE or TEPCat, and determines the eccentricity and argument of periastron for a given planet. The planetary identifier is used to search in DACE or TEPCat and if object is found the planet parameter values are provided in the returned report along with the derived eccentricity and argument of periastron. Any user inputted values with be reported and used in all calculations over-writing those retrieved from DACE or TEPCat.

`pycheops.PlanetProperties`(*self*, *identifier*, *force_download=False*, *configFile=None*, *query_dace=True*, *query_tepcat=True*, *T0=None*, *P=None*, *ecosw=None*, *esinw=None*, *depth=None*, *width=None*, *K=None*, *verbose=True*):

Parameters	**identifier**:	String of the target whose parameters are to be retrieved
	force_download:	Boolean on if the catalogue should be downloaded regardless of the presence of a local version of the catalogue
	configFile:	String of the directory of the PYCHEOPS configuration file
	query_dace:	Boolean on if parameters should be retrieved from the DACE planet table
	query_tepcat:	Boolean on if parameters should be retrieved from TEPCat
	T0:	Integer or float of the user inputted transit centre time to be reported
	P:	Integer or float of the user inputted period to be reported
	ecosw:	Integer or float of the user inputted eccentricity and argument of periastron components to be reported
	esinw:	Integer or float of the user inputted eccentricity and argument of periastron components to be reported
	depth:	Integer or float of the user inputted transit depth to be reported
	width:	Integer or float of the user inputted transit width to be reported
	K:	Integer or float of the user inputted radial velocity semi-amplitude to be reported
	verbose:	Boolean on the printing of information to the screen
Returns	**report**:	A report of the DACE or TEPCat retrieved, user inputted, or calculated stellar values

pycheops.StarProperties

A class that creates an object that can be utilised to retrieve stellar parameter values from SWEET-Cat or DACE, and determines the stellar density and limb-darkening coefficients for a given object. The coordinates of the target are obtained by querying the SIMBAD Astronomical Database using the inputted object name with the coordinates used as inputs for querying SWEET-Cat or DACE. If the object is found in SWEET-Cat or DACE the stellar parameter values are provided in the returned report along with the derived stellar density and limb-darkening coefficients. Any user inputted values with be reported and used in all calculations over-writing those retrieved from SWEET-Cat or DACE.

`pycheops.StarProperties`(*self*, *identifier*, *force_download=False*, *dace=False*, *match_arcsec=5*, *configFile=None*, *teff=None*, *logg=None*, *metal=None*, *verbose=True*):

Parameters	**identifier**:	String of the target whose parameters are to be retrieved
	force_download:	Boolean on if the catalogue should be downloaded regardless of the presence of a local version of the catalogue
	dace:	Boolean on if parameters should be retrieved from the DACE stellar table
	match_arcsec:	Integer or float of the radius around the target's coordinates used to search for data in SWEET-Cat
	configFile:	String of the directory of the PYCHEOPS configuration file
	teff:	Integer or float of the user inputted effective temperature to be reported
	logg:	Integer or float of the user inputted gravity to be reported
	metal:	Integer or float of the user inputted metallicity to be reported
	verbose:	Boolean on the printing of information to the screen
Returns	**report**:	A report of the SWEET-Cat or DACE retrieved, user inputted, or calculated stellar values

References

[1] Baraffe, I., Chabrier, G., & Barman, T. 2008. *Structure and evolution of super-Earth to super-Jupiter exoplanets. I. Heavy element enrichment in the interior.* Astronomy and Astrophysics, Volume 482, pp. 315-332

[2] Benz W., Ehrenreich D., Isaak K. 2018. CHEOPS*: CHaracterizing ExOPlanets Satellite.* In: Deeg H., Belmonte J. (eds) Handbook of Exoplanets. Springer, Cham.

[3] Campante, T. L. 2016. *The Asteroseismic Potential of TESS: Exoplanet-host Stars.* The Astrophysical Journal, Volume 830, Issue 2, id. 138, 15 pp.

[4] Claret, A.. 2019. *Tables of Limb-darkening and Gravity-darkening Coefficients for the Space Mission Gaia.* Research Notes of the American Astronomical Society, Volume 3, id. 17

[5] Ehrenreich, D., et al. 2019. CHEOPS *Observers Manual.* European Space Agency.

[6] Hoyer, S., et al. 2019. *Expected Performances of the Characterising Exoplanet Satellite (CHEOPS) III. Data Reduction Pipeline: Architecture and Simulated Performances.* Astronomy & Astrophysics. Volume 635, id. A24, 14 pp.

[7] Lendl, M. C., et al. 2013. *The Hot Dayside and Asymmetric Transit of WASP-189 b Seen by CHEOPS.* Astronomy & Astrophysics,

[8] Maxted, P. F. L. 2018. *Comparison Of The Power-2 Limb-Darkening Law From The STAGGER-Grid To Kepler Light Curves Of Transiting Exoplanets.* Astronomy & Astrophysics, Volume 616, id. A39, 13 pp.

[9] Maxted, P. F. L. & Gill, S. 2019. *Qpower2: A Fast And Accurate Algorithm For The Computation Of Exoplanet Transit Light Curves With The Power-2 Limb-Darkening Law.* Astronomy & Astrophysics, Volume 622, id. A33, 7 pp.

[10] Moya, A., et al. 2018. *Empirical Relations For The Accurate Estimation Of Stellar Masses And Radii.* The Astrophysical Journal Supplement Series, Volume 237, id. 21, 20 pp.

[11] Pereira, F., et al. 2019. *Gaussian Process Modelling Of Granulation And Oscillations In Red Giant Stars.* Monthly Notices of the Royal Astronomical Society, Volume 489, 5764 pp.

[12] Rodrigo, L., et al. 2017. *Linear Models for Systematics and Nuisances.* Research Notes of the American Astronomical Society, Volume 1, id. 7.

[13] Santos, N. C., et al. 2013. *SWEET-Cat: A Catalogue Of Parameters For Stars With ExoplanETs. I. New Atmospheric Parameters And Masses For 48 Stars With Planets.* Astronomy & Astrophysics, Volume 556, id. A150, 11 pp.

[14] Silva, D. & Skilling, J. 2006. *Data Analysis - A Bayesian Tutorial.* Oxford University Press.

[15] Southworth, J. 2011. *Homogeneous studies of transiting extrasolar planets - IV. Thirty systems with space-based light curves.* Monthly Notices of the Royal Astronomical Society, Volume 417, pp. 2166-2196.

[16] Trotta, R. 2007. *Applications of Bayesian model selection to cosmological parameters.* Monthly Notices of the Royal Astronomical Society, Volume 378, pp. 72-82.

[17] Zeng, L., Sasselov, D. D., & Jacobsen, S. B. 2016. *Mass-Radius Relation for Rocky Planets Based on PREM.* The Astrophysical Journal, Volume 819, id. 127, 5 pp.