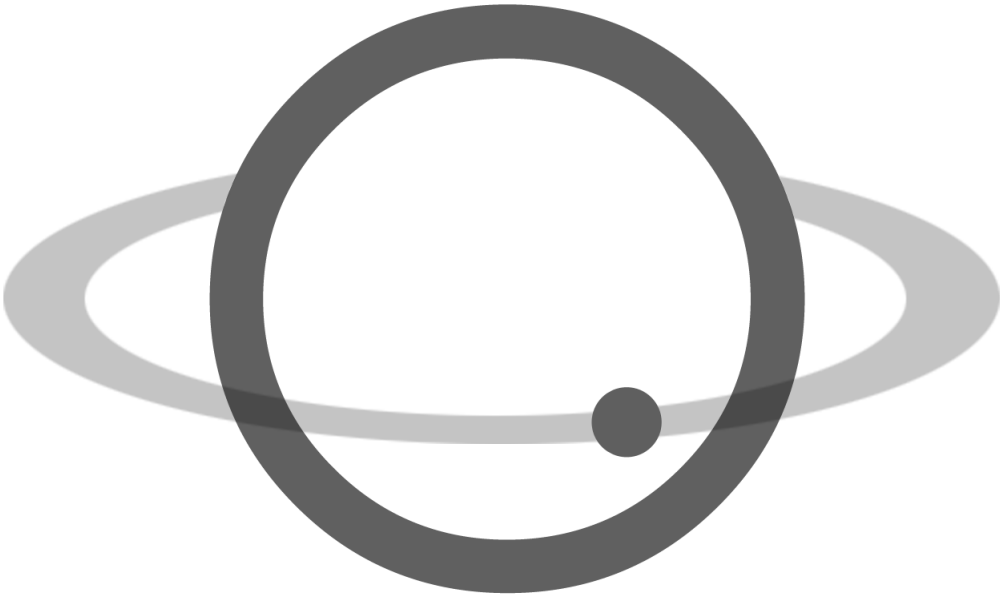
OSCAAR: Open Source differential photometry Code for Amateur Astronomical Research

Brett M. Morris and the OSCAAR Team

August 31, 2013



Contents

1 Introduction **4**
- 1.1 System Requirements . . . 4
- 1.2 Installing OSCAAR . . . 5

2 Collecting Data **6**
- 2.1 Navigating The Sky . . . 6
- 2.2 Telescope Tracking . . . 6
- 2.3 Defocusing . . . 7
- 2.4 Theory: Photon Noise . . . 8
- 2.5 Dark Fields and Flat Frames . . . 8
- 2.6 Imaging Software . . . 8

3 The OSCAAR GUI **9**
- 3.1 Transit predictions with The OSCAAR Custom Ephemeris Calculator . . . 9
  - 3.1.1 Advanced Options . . . 13
- 3.2 Choosing Your Input Files . . . 13
- 3.3 Locating Your Stars with DS9 . . . 14
- 3.4 Making a Master Flat . . . 15
- 3.5 Running Parameters . . . 16
  - 3.5.1 Smoothing Constant . . . 16
  - 3.5.2 Tracking and Photometry Plots . . . 17
  - 3.5.3 Track Zoom . . . 19
  - 3.5.4 CCD Gain . . . 19
  - 3.5.5 Aperture Radius . . . 20
  - 3.5.6 Notes . . . 20
  - 3.5.7 Ingress and Egress Times . . . 20
- 3.6 Results . . . 21
- 3.7 Loading Previously Saved Results . . . 22

4 Light Curve Fitting **22**
- 4.1 Introduction To Fitting With Markov Chain Monte Carlo (MCMC) . . . 23
- 4.2 Using OSCAAR 's MCMC Module . . . 24
- 4.3 Understanding MCMC Results . . . 27

5 Algorithm Notes **27**
- 5.1 `oscaar.photometry.phot()` . . . 27
- 5.2 `oscaar.astrometry.trackSmooth()` . . . 27
- 5.3 The dataBank Object . . . 29
  - 5.3.1 `dataBank.scaleFluxes()` . . . 29
  - 5.3.2 `dataBank.calcMeanComparison()` . . . 31
  - 5.3.3 `dataBank.computeLightCurve()` . . . 32

6 Troubleshooting **32**

7 Contributing to OSCAAR **32**

8 Acknowledgements **33**

1 Introduction

OSCAAR 's core is a differential photometry package built for users of any experience level at observatories of any size. The process of differential photometry can be applied to many astrophysical phenomena including transiting extrasolar planets, variable stars, rotating asteroids and more. This package is prepared specially for transiting exoplanet observations, though often other photometric observations can be analyzed with OSCAAR without any tweaking to the source code.

Development for OSCAAR began in the summer of 2011 to take a series of images obtained at the University of Maryland Observatory in College Park, MD, USA and churn out light curves of transiting extrasolar planets. If you're looking to do something similar, you've found the right code! The core of the code is written in Python, but it has been designed to be operated with a graphical user interface (GUI) for users who have never seen Python before. That being said, experienced programmers will find that the guts of OSCAAR provide a well-documented toolkit for even the most demanding of photometric measurements. Since the v1.0 release, OSCAAR has been used by astronomers at all levels, from undergraduates to observatory-directing IAU members.

This document is intended to guide you through the observing techniques you need to make your first observations and using OSCAAR to analyze them. It is currently still in the process of being written. **This document is a live document, and parts may be incomplete**. We will make a stable version of the documentation for you to refer to as soon as we can!

Here we summarize the contents of this documentation. Section 1 lists the dependencies required to run OSCAAR and explains how to install OSCAAR on your computer. If you are new to photometry, you may consider reading Section 2, which will summarize observing techniques to make successful photometric measurements. In Section 3, we will detail how to run OSCAAR from the graphical user interface (GUI) or other ways without directly coding in Python. In Section 4, we'll cover how to use OSCAAR to fit a model to your transit light curve. In Section 5 we discuss the classes and methods that are built into OSCAAR that users with programming experience may find useful to design their own analysis tools.

1.1 System Requirements

OSCAAR **will run on the following operating systems:**

Windows Vista and later
Mac OS 10.6, 10.7 and 10.8 (with caveats)
Ubuntu 12.04 and later or similar

See instructions throughout this section denotated by italics that are specific to certain operating systems.

There are several packages in addition to Python (v2.7) that must be installed on your machine before running OSCAAR . They are all free and available for download via links below. We apologize for the number of packages that are necessary to download before using OSCAAR , but we think that these packages enable for the most efficient open-source distro that we can provide. We think you'll find that if you're going to build on the code within OSCAAR , this palette of packages provides most of the tools you need for scientific computation in Python.

Python 2.7 (*not v3.0+*): The core language of oscaar
NumPy and **SciPy**: Python modules for efficient scientific computations
PyFITS 3.1.0+: A Python module for reading/writing FITS files
matplotlib: A Python module for publication quality, interactive plotting

wxPython 2.8+: A Python GUI toolkit
PyEphem: A Python module for ephemeris calculations
Windows only — **setuptools**: A Python module for installing the OSCAAR package

Though the modules used in this package have been stable historically and probably will not go away any time soon, we want to note here that changes implemented in future versions of these packages could potentially break features in OSCAAR . If you notice a broken feature, let us know by posting an issue on our Issue Tracker (see Section 6).

Note for Linux Users: You only need one command to download and install most of the dependencies:

```
sudo apt-get install python-numpy python-scipy python-matplotlib python-wxgtk2.8
```

The only ones left are PyFITS and PyEphem.

Note for Mac OS 10.6 and 10.7 Users: Python will already be installed on your machine, but it likely isn't the most recent version of Python 2.7. Download Python 2.7.X if it's not. Mac OS 10.6 users will also need Xcode, the Apple developer tools app that allows your computer to do things like program for OS X or iOS. Xcode can be downloaded from the App Store for free. However, if you're running OS X 10.6.x and you don't want to upgrade to OS X 10.7 you will need to download the older version of Xcode, v3.2. To do so, visit the Developer Downloads page and search for "Xcode 3.2". You will be prompted for your Apple ID to sign in as an Apple Developer. If you do not have a Developer account, you will need to create one at this stage. This will be a small time commitment. Install this but **note**: this is a large download, about 2.4 GB. On OS X 10.7.0 or later, you'll instead need the GCC compiler. Apple provides this in their 'Command Line Tools' at their Developer site. You are now prepared to download Python packages such as PyEphem via their "Source code (.tar.gz)" links. Install them as suggested by each pacakge's INSTALL doc.

Note for Mac OS 10.8 Users: When you download python2.7 or any other OSCAAR dependencies, there is an error message that pops up because the "GateKeeper", a function added in MacOsX 10.8, will not let you download things from an unidentified source. All of the dependencies have to use solutions found in their ReadMe.txt files to download them because "GateKeeper" will block all the downloads. wxPython can present additional difficulties if you are downloading it on a machine running modern versions of OS X. The standard links to OS X compatible binaries on the wxPython page are misleading. You'll have your best shot at downloading a working version of wxPython for your OS X-running machine if you download the less prominently labeled version "wxPython2.9-osx-cocoa-py2.7". This one is the one to get.

1.2 Installing OSCAAR

OSCAAR is currently in beta release, so there is no "stable" package to download just yet. Instead, each copy that you download will have the latest up-to-the-second updates. To download OSCAAR via GitHub, go to http://github.com/OSCAAR/OSCAAR/zipball/master to begin downloading and unarchive the ".zip" file. Next, open up your Command Prompt (Windows) or Terminal (Mac OS/Linux) and go to the unarchived OSCAAR directory, using the command "cd <name of destination directory> to move from one directory to another. Once at the "OSCAAR-master" directory, enter

```
python setup.py install
```

to install OSCAAR in your system's Python directories. If you are working on an account without administrative privileges, you can add the user option to download OSCAAR to only your user python directories:

```
python setup.py install --user
```

Next, change directories to some other directory. If you were to import `OSCAAR` from this unarchived OSCAAR directory, Python would use the source files that you downloaded, rather than the ones you installed (the difference is subtle but important).

Note for Windows Users: The Command Prompt will not recognize the command "python" until you type:

```
set path=%path%;C:\python27
```

This command will need to be entered for every Command Prompt you open unless you add a new path variable in Windows. After this command has been entered, you will be able to proceed with installation as described above.

2 Collecting Data

2.1 Navigating The Sky

It is one hour before the photometric event that you want to observe, you have taken your flat fields, and you're ready to slew to your target. You punch in the RA and declination of the target, and your telescope lumbers over to that part of the sky, but you can't recognize the pattern of stars that come out on your first image. Is that star near the center of the field actually your target, or are you looking at the wrong part of the sky?

While it is not necessary for running `OSCAAR` , we highly recommend that you download the open-source planetarium software Stellarium for navigating the sky and planning. This free, user-friendly package is supported on nearly every operating system and boasts some advanced features that will make it easy for you to find your targets.

One feature of particular utility for the exoplanet community is that the object finding search bar that you use to find objects in Stellarium is linked to the professional astronomical database SIMBAD to resolve nearly any object by any name. SIMBAD is updated relatively quickly, so when you want to go out and observe that new exoplanet but your old planetarium software doesn't know about the latest WASP or HAT target, Stellarium will resolve the name through SIMBAD and take you to the coordinates published for that object[1].

Stellarium also has great "Flip Scene" buttons that allow you to mirror flip your views of the sky to match the flips that your telescope optics do. That way you can match up your observed field of view with the stars in Stellarium without having to imagine complicated image transpositions in your head.

2.2 Telescope Tracking

Differential photometry is generally done on a series of images of the same patch of the sky over a long period of time. You'll need a telescope that is well polar-aligned so that the telescope's tracking keeps the stars in nearly the same spot on your detector throughout the duration of your observations. It is a tremendous challenge to align most small telescopes well enough to track a star perfectly over several hours, so `OSCAAR` is built to monitor and correct for the drift in star positions on the detector over time. Just make sure your target object and a

[1] Of course, this feature only works when your machine is connected to the internet, so if you do not have an internet connection at your observatory, you'll still need to query for your target before you get to the observatory, print out your star charts.

few comparison stars stay in the field throughout the whole observation[2]. The star tracking algorithm follows each star individually over time. If your photometric target is an asteroid, for example, that moves relative to the comparison stars over time, OSCAAR will happily track the asteroid and the comparison stars independently.

2.3 Defocusing

In precision photometry, relying on individual pixels is dangerous. Pixel defects occur frequently that can cause a pixel to read much higher counts than they actually receive (these are called "hot pixels") or sometimes much lower counts ("dead pixels"). If your target object is perfectly focused on a few pixels, you may be putting all of your photometric-eggs in one basket. Thus it is often advised that you **do not focus the telescope perfectly** when doing photometric measurements. If you can, defocus the telescope significantly so that you spread out the starlight over a few of pixels, and your pixel-to-pixel variations will play a less-significant role in the introduction of unwanted systematic noise and bias. See Figure 1 for examples.

Some photometry codes prefer objects focused in Gaussian-like shapes, but OSCAAR is written to accurately track stars of unconventional shapes. At the University of Maryland Observatory, we make most of our measurements on a 6in refractor. We defocus the stars until they look like small donuts (the hole is an artifact of the optical path of the refractor), and we've found that our photometry comes out best this way. OSCAAR won't complain if your stars are donut-shaped, Gaussian or somewhere in between.

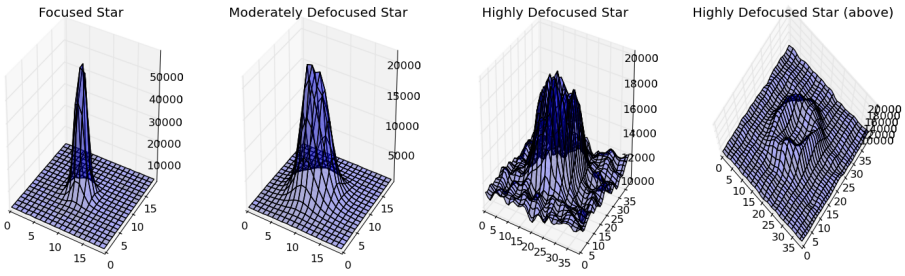


Figure 1: Images of stars with varying levels of defocusing. The x and y axes are the pixel indices, intensity in counts is plotted along the z. OSCAAR can track any of these stars with precision — even when the shape of the star is not approximately a Gaussian, like the highly defocused star on the right which is shaped like a donut.

One must keep in mind that by spreading out the light over many pixels, the counts measured by each pixel is lower. As a result, defocusing is most helpful when observing bright objects. If the source you are observing is bright enough, you can defocus significantly without losing too much signal. However, it is important to note that the quality of your light curve will be directly related to how much more signal than noise you detected, as we will discuss in Section 2.4. Use defocusing sparingly (or not at all) when imaging dim targets[3].

[2]If you need to re-center your target in the middle of an observation, OSCAAR will only be able to continue to track the stars without special code tweaking if you change the stars' centroid positions by a few pixels (i.e., ∼2 or 3 pixels) at a time in between each exposure. **The current version of OSCAAR is not resilient against sudden discontinuous tracking anomalies.**

[3]For example, we use a sharp focus for stars dimmer than $V = 10$ on our 152mm refractor at the University of Maryland Observatory.

2.4 Theory: Photon Noise

There is a fundamental physical limit to the signal-to-noise ratio that can be achieved in photometric measurements. Since photometry is the process of counting photons, there is a type of statistical uncertainty introduced into each measurement that goes by many names: *photon noise, Poisson uncertainty, shot noise,* etc. This uncertainty is unavoidable in counting measurements and is easy to calculate — the photon noise, or the uncertainty in a measurement of N photons, σ_N is

$$\sigma_N = \sqrt{N}. \tag{1}$$

You can see directly from Equation 1 that the fractional uncertainty in the signal (σ_N/N) decreases with increasing N as $1/\sqrt{N}$, so the more signal you have, the lower the limiting fundamental uncertainty. Of course, in practice there are many additional noise sources that will add uncertainty into your data and increase the scatter in your light curves.

2.5 Dark Fields and Flat Frames

Collecting dark frames and flat fields is standard practice for removing systematic effects from CCD images. Flat fields correct for dust and other inconsistencies in the optical path that artificially brighten or dim the objects in your images. Dark frames remove some flaws in the CCD like hot pixels and dark current variations. Some CCD imaging software like MaxIm DL have easy preset routines for taking dark frames. We recommend taking ≥ 8 dark frames for each set of observations you take. `OSCAAR` will take the mean of these sets and apply them appropriately to each image of your data set.

Collecting good flat fields can sometimes be more of an art than a science, but good flats are important for good photometry. For this reason, we've incorporated two methods of "master flat" making routines into `OSCAAR` , that combine raw flat fields into one master flat. The first method is called the "Standard" method by the `OSCAAR` GUI, which takes a mean of all of the input flat frames. This is the fastest computational method. Standard master flats are ideal for "dome flats", where the flat fields are obtained by imaging a screen. Many astronomers prefer "twilight" or "sky" flats, in which the telescope takes images of the sky as the sun sets, and light from the sky acts as the uniform light source. We developed a corresponding twilight master flat setting for `OSCAAR` , which fits a linear function to the intensity of each pixel over time, and uses the best-fit intercept as the normalizing factor for the master flat. Since loops are slow in Python, this method may take a few minutes to produce a master flat, but the payoff that you gain in photometric precision is worth the wait. At the time of writing, we're currently investigating how to implement these algorithms in `C` to help us speed up these expensive computations.

2.6 Imaging Software

A bunch of imaging packages could suite your needs for photometry with `OSCAAR` . At the University of Maryland Observatory, we use MaxIm DL to handle our imaging. Any imaging software that enables you to take a time-series of CCD images will do.

Observing software like MaxIm DL allow you to choose your imaging **binning**, which enables the detector to read-out pixels in groups. For example, a 2×2 binning will take a square of four pixels and save them as one composite pixel. This reduces the read-out time, the size of the output files, and the run time of scripts that

have to read and manipulate those images. We recommend that you use 2×2 binning when you can for these reasons. It is often unimportant to save images at the full-resolution of the detector, especially when you are using defocusing anyway, as described in Section 2.3.

3 The OSCAAR GUI

OSCAAR can be run from a graphical user interface that we call the oscaarGUI . It's designed to control the running parameters that control the science algorithms so that you can tweak different features in the code by interacting with the GUI. The GUI included in OSCAAR 2.0 was developed by contributor Daniel Galdi (UMD) using the wxPython package.

Once OSCAAR is installed, you should be able to open a Command Prompt (Windows) or Terminal (Mac OS/Linux) and type

```
python -c "from oscaar import oscaarGUI"
```

to open the oscaarGUI . It will look something like Figure 2.

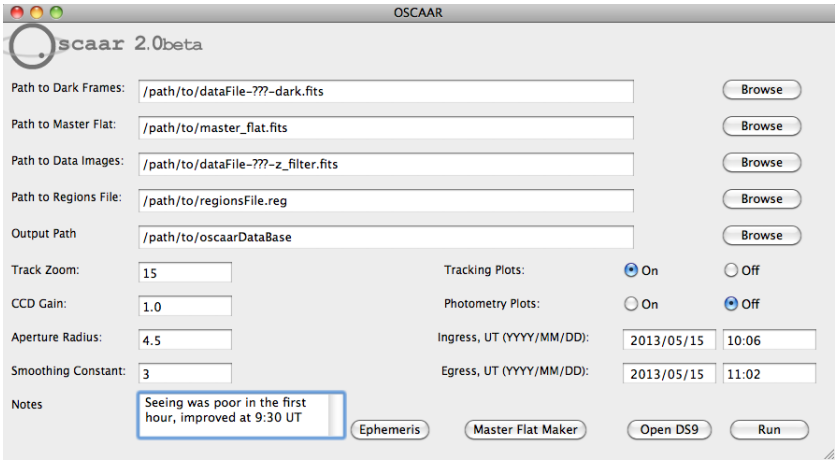


Figure 2: The OSCAAR graphical user interface (GUI).

3.1 Transit predictions with The OSCAAR Custom Ephemeris Calculator

You've seen the weather report and you know that your observatory will have clear skies this week — but which planets will transit this week and at what times? To answer this question, click the Ephemeris button at the bottom of the oscaarGUI .

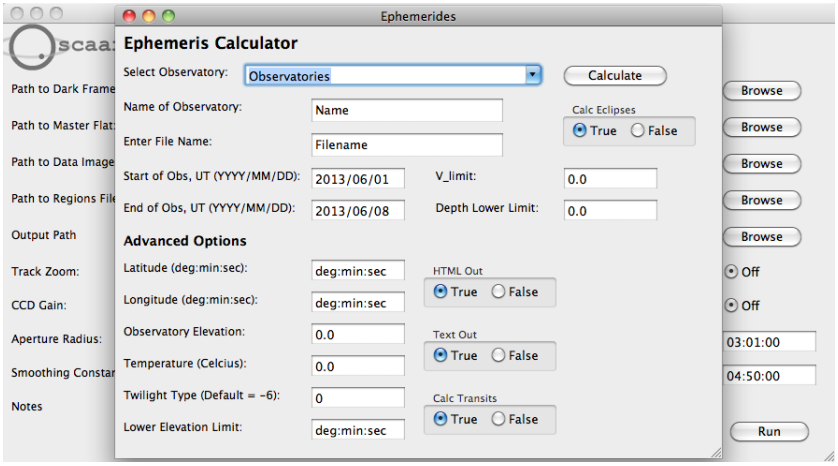


Figure 3: The OSCAAR Custom Ephemeris Calculator.

This new window is the `Ephemeris` tool, which will calculate transit time predictions for you. Select from the major observatories or friends of OSCAAR listed in the `Select Observatory` field, or choose Enter New Observatory if yours is not listed, and enter the range of dates that you'd like to observe in the `Beginning of Obs` and `End of Obs` fields in `YYYY/MM/DD` format. These dates defined in Universal Time (UT).

We've found in our experience using transit prediction services that it's easy to get drowned in too many transit event suggestions that you don't have the precision to observe. You may know, for instance, that your telescope aperture is too small or the sky brightness from light pollution at your observatory is too bright to do photometry on stars dimmer than a particular visible V magnitude. We built features into this ephemeris generator so you can set the `Apparent Mag. Upper Limit` to filter out any results for transiting planets that are dimmer than a "limiting V magnitude" for good photometry that you've entered. At the University of Maryland Observatory, located < 15 km from the White House and within the Washington D.C. beltway, we have plenty of light pollution. The bright sky background makes it hard to get decent photometric precision on stars with magnitudes $V > 12$.

Similarly, your photometric signal-to-noise will limit the transit depths that you can observe. If a transit only causes a fractional decrease in brightness of the star on the order of 10^{-4}, you probably won't have the precision to be able to detect it at a small observatory. Thus you can filter out shallow (small depth) transits by setting the `Depth Lower Limit` field. At the University of Maryland observatory, we generally try to observe transits deeper than 0.010 magnitudes (reminder: depths are measured as positive numbers, even though they represent decreases in fluxes).

When you press `Calculate`, OSCAAR will download a copy of the latest known transiting system parameters from exoplanets.org. This website created by Wright et al. (2011) catalogs all known system parameters for each planetary system, and is updated frequently as new publications a released and new planets are discovered. OSCAAR will download a copy of the entire database and calculate when the next transits and secondary eclipses will occur for each system using published transit times and period measurement. It also uses the right ascension and declination measurements provided by exoplanets.org to calculate whether or not each transit event will happen above or below the horizon from your observing location, and during day or night, by performing ephemeris calculations with `PyEphem`. Then the transit or eclipse events that occur completely above the horizon (from

ingress to egress entirely), at night, and that satisfy the V limit and depth limit criteria you set are returned.

To provide you with a consistently formatted, nicely rendered table of events, `OSCAAR` generates the output ephemeris in HTML and opens it in your web browser. The columns of this table are sortable — just click on the title row of each column to sort. We know that you may have this ephemeris open on your computer while you're observing at night, and we don't want you to blind yourself looking at a brightly colored ephemeris, so we included two style sheets so that you can switch between. Try changing the"Day" to "Night" mode with the buttons above the ephemeris table for a dark red color palette that may disturb your night vision less than the bright "Day" style sheet. Outputs can also be written in comma-separated value format (CSV).

Table 1: **Ephemeris Cheat Sheet**

Field	Definition	Example values
Name of Observatory	The name of your observatory	Mauna Kea, Hawaii
Enter File Name	Only enter a file name if you're entering a new observatory	`maunaKea.par`
Start of Obs.	The start date in Universal Time of your ephemeris, `YYYY/MM/DD`	2013/07/02
End of Obs.	The end date in Universal Time of your ephemeris, `YYYY/MM/DD`	2013/07/09
Apparent Mag. Upper Limit	The calculator will filter out any target stars with visible V magnitudes greater than this (alternatively with the advanced option it will filter by K magnitude)	13.0
Depth Lower Limit	The calculator will filter out any transit events that have a smaller expected transit depth than this	0.005
Latitude	Latitude of your observatory, `deg:min:sec`. North is positive, south is negative.	19:49:15.6
Longitude	Longitude of your observatory, `deg:min:sec`. East is positive, west is negative.	-155:28:24
Observatory Elevation	Elevation of your observatory in meters	4123.0
Temperature	Air temperature at time of observation in degrees C	0.0
Lower Elevation Limit	Filter out targets below the lowest altitude that your telescope can observe, `deg:min:sec`	30:00:00
Twilight Type	Tweaks the definition of when "sunset" has occurred based on the various definitions of "twilight"	[Choose from list]
Show Local Times	This includes columns in the table for the conversion from UT to LT, *where LT is the local time for the computer running* `OSCAAR` *, NOT for the location of the entered observatory*	[Choose on/off]
Band Type	Bandpass for brightness measurements displayed in ephemeris	[Choose from V/K]
Calculate Eclipses	Toggle to calculate the times of eclipse events as well as or in place of transits	[Choose on/off]
Calculate Transits	Toggle to calculate the times of transit events as well as or in place of eclipses	[Choose on/off]

3.1.1 Advanced Options

The parameters in the lower half of the ephemeris calculator tool allow you to calculate the sky altitudes and directions at ingress and egress with extra-high precision, and to set the behavior of the calculator to give you more control over the results you see. The latitude and longitude of your location can be set in the format: `degrees:mm:ss.ss`. Your observatory elevation can be set in meters, and temperature in degrees Celsius. Then there are two parameters that will have a strong affect which transits make it through your filters.

The `Twilight Type` parameter determines how many degrees below the horizon the Sun must be for `OSCAAR` to consider the Sun to have "set" and for nighttime observations to begin. In practice, as you know, the sky isn't dark enough to start observations precisely at sunset, since twilight sky brightness lingers for some extra time. There are three names for different definitions of *twilight* — civil, nautical and astronomical twilight, which correspond to a minimum solar elevation of -6, -12 and $-18°$. You can chose which of these levels of solar elevation will dictate the beginning (and the end — the same definitions work for the morning hours) of astronomical observations by tweaking this parameter.

The `Lower elevation limit` parameter sets how high above the horizon the object must be before it is visible from your observatory. For photometric observations, it's a good rule of thumb to only observe objects above 30° elevation, since at smaller angles the thick cross-section of air that you must look through compounds the problems of removing telluric effects. You may, for example, have trees on the horizon in all directions at your observatory that limit your lowest-possible elevation for observing to 35°, or your observatory dome or roof may not accommodate observations below a similar elevation. You can account for these factors in this parameter so that you only see events that are observable with your equipment, from your observatory. The format for the `Lower elevation limit` parameter is `degrees:mm:ss.ss`.

3.2 Choosing Your Input Files

The first step to running `OSCAAR` on your data from the `oscaarGUI` is to select which `.fits` files you want to use. `OSCAAR` allows you to access your data files a few different ways. Our first task is to select the "path to dark frames" (see Figure 2). A *path* is essentially a list of directions to where a file is located on your computer. On Windows you may recognize that paths to files on yourC hard drive are sometimes listed as "`C:\some folder\some other folder\...`", and on Mac OS X they typically begin with "`/Users/your user name/some folder/...`". The easiest way to give the paths to *all* of the files that you will be analyzing is to use a feature of `OSCAAR` 's path selectors that takes advantage of Python's `glob` module.

The imaging software that you use may allow you to save each image from your CCD with a pre-formatted name and an iterating index for each image. For example, in our sample data set we simulate this behavior by giving all of the dark frames names with the pattern "simulatedImg-000d.fits", "simulatedImg-001d.fits", "simulatedImg-002d.fits", etc. You could press the `Browse` button in the `oscaarGUI` and then hold shift and click through every dark frame that you want to select, or you could use a nifty shortcut...

If you press the `Browse` button and select one of the dark frames, the text box to the left of the `Browse` will be filled with the path to that file. Let's say that this text box is filled with

"`C:\folderA\folderB\simulatedImg-000d.fits`".

We can use the "?" character to indicate a wildcard, so that any file that satisfies the rest of the pattern with any character in place of the "?" will be accepted. For the above example, if we enter

"C:\folderA\folderB\simulatedImg-???d.fits"

we will get back all of the files that have that pattern with any numbers (or letters) in place of the question marks.

Let's say all of the images in "folderB" are dark images, and you want to just select all of those files. You can use the "*" character in place of the name of the file, and all files in the directory will be selected, for example,

"C:\folderA\folderB*"

will grab all files in "folderB". But what if there was a text file or some other non-FITS file in "folderB"? In that case, you can specify the file extension and/or last few characters so that you get the right file, like this:

"C:\folderA\folderB*.fits",

which will catch all FITS files in "folderB".

Go ahead and fill in these fields for the dark frame and data image paths. When you use any of these rules, OSCAAR will figure out all files that match and use those files for you. The next time you load the oscaarGUI , the files that were grabbed will be listed in the text box, separated by commas.

The "Output Path" field asks you to pick a path to the file that OSCAAR will produce with the results of all of its algorithms. Pick a place where you can put this output data file and give the file an appropriate name.

3.3 Locating Your Stars with DS9

The next task you'll have to do to prepare OSCAAR for analysis is to tell it what stars you care about in your images. If you have a set of images, there could be tens or hundreds of stars in each image, some of them close to the edges, some of them with binary companions; some of them ideal control stars, some of them not. In order to ensure that the stars being picked as control stars are appropriate choices, OSCAAR has the user enter each of the stars into OSCAAR with the help of SAOImage DS9.

To begin, click the Open DS9 button at the bottom of the GUI (see Figure 2) to start DS9 and open the first image from the series of images you will process. Set the scale and zoom so that you can see most of the image and most of the stars in it. With the mouse set to Pointer (Edit >Pointer), click on the target star (the exoplanet host star or variable star). A green circle will appear over the star. Double clicking inside the circle will yield a dialogue box which contains the pixel coordinates of the circle's center and radius, see Figure 4.

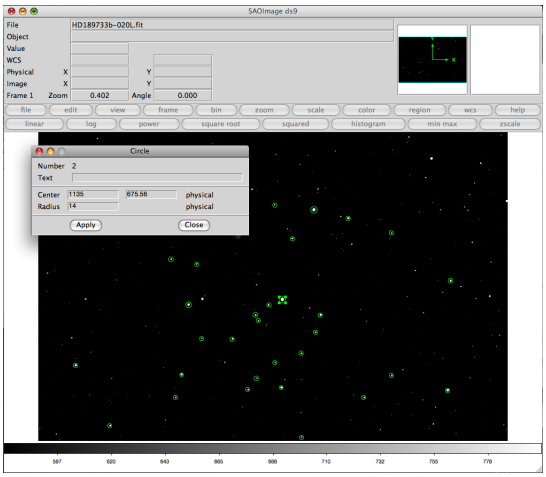


Figure 4: Using DS9 to locate the target and comparison stars.

The first star that you select will be treated as the target star, and any subsequent choices will be comparison stars. Repeat this process for as many comparison stars as you like (try to use as few as possible for speed). Avoid picking stars near any obvious defects on the detector, any stars less than 150 pixels from the edge of the image, or stars with a neighboring star close nearby.

When you're done choosing control stars, go to `Regions >Save Regions...` and save a regions file in a directory where you will be able to find it later. This file will contain the pixel coordinates of the stars at the beginning of the observation, which tells `OSCAAR` which stars to track. You may want to check that the test star is in fact the first star in the regions file. Open the regions file in a text editor and check that the first line resembling "`circle(100,600,10)`" has the proper (x,y) pixel coordinates and radius (in this example, the position is (100,600) with a radius of 10). If the first circle coordinate line does not point to the target star, find the circle coordinate that does, move that line to the top of the list, and save the file. Select this regions file (with the ".reg" extension) from the `Browse` button of the `oscaarGUI` for the "Path to Regions File" field.

3.4 Making a Master Flat

Now that you have properly set the path to the darks and the data images, you must now make your master flat. First click the "Master Flat Maker" button at the bottom of the `OSCAAR` window, and a dialog box should pop up that looks like Figure 5.

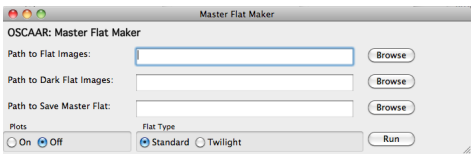


Figure 5: `OSCAAR` 's Master flat maker allows you to make master flats out of raw images.

Now you must set the paths to the flat images and the dark flat images in the same way you set the image

paths previously. You must also enter the path and filename for the master flat. To do this, click the `Browse` button and navigate to the location you would like to save the master flat.

There are two options for how you can make your flat, either the "Standard" or "Twilight" flat type. The standard flat maker algorithm is a simple mean combination of the flat frames that you enter into the flat maker GUI. If you took dome flats or bright sky flats, this is the right option for you. If you took flats at twilight as the sun was setting and the sky background was your "screen", the twilight flat algorithm will fit a linear function to the intensity of each pixel over time, and use the intercept as the normalization factor for the flat. The twilight flat algorithm is much slower than the standard method, because those linear fits are computationally expensive in an interpreted language like Python. We hope to code up alternative versions of these routines that you can experiment with on your dataset, some of which may access much faster, compiled code in `C` or `Cython` (an extended version of Python that compiles like `C` code).

Now click the `Run` button on the master flat maker and close it when it is done. You should now choose the master flat file that you generated when selecting the "Path to Master Flat" in the `oscaarGUI` .

3.5 Running Parameters

The next step in running `OSCAAR` is to properly set all of the initial running parameters. These have default values set, but these may need to be modified for your particular data set.

3.5.1 Smoothing Constant

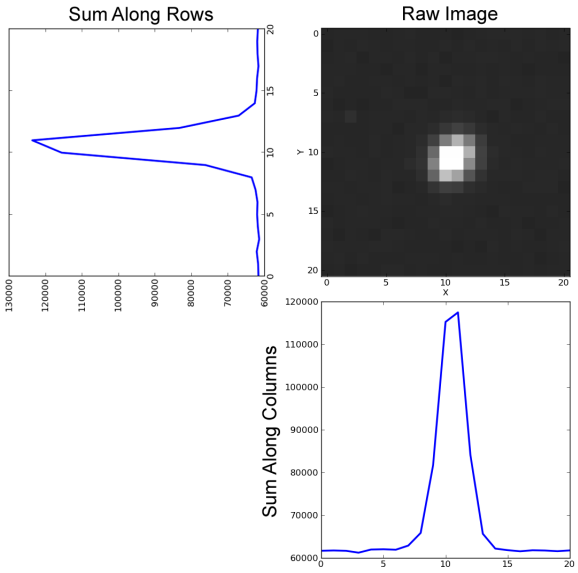


Figure 6: Sums along the columns and rows around a star.

By default, oscaar v2.0 tracks stars with a fun algorithm described in Section 5.2. This algorithm finds the centroid of the star by looking for certain features in the first numerical derivative of the spatial intensity profile of the image of the star. These "intensity profiles" are the sums of pixel counts along the rows and columns

near the star being tracked, see Figure 6 for an example. In order to find the centroids accurately using this method, it helps to smooth out the stars by blurring the image artificially, so that background noise and bad pixels are not incorrectly interpreted as features of the stellar intensity profile. Of course, the artificial smoothing is non-destructive; the images used to measure photometry are not smoothed.

Since it is not uncommon to defocus telescopes for photometry, you may want more or less smoothing in your images. The running parameter `Smoothing Constant` adjusts how much smoothing is applied in the tracking algorithm. `Smoothing Constant = 0` will not smooth the image at all (not recommended), and values around ~ 3 will be significantly smeared (non-integer values are accepted). See Figure 7 to see what how various values of the `Smoothing Constant` affect a raw stellar intensity profile.

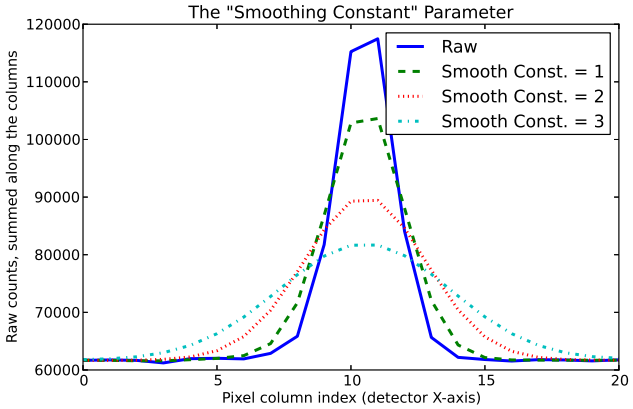


Figure 7: A range of `Smoothing Constant` values from 0 to 3 are applied to the stellar intensity profile along the columns surrounding one star (see Figure 6 for the origin of the intensity profile).

3.5.2 Tracking and Photometry Plots

In order to visualize the effects of the running parameters that you entered on the photometric process, you can have oscaar plot visualizations of various procedures as they are executed. This will add significantly to the runtime of your analysis, as the plotting package `matplotlib` is not as fast as `OSCAAR` 's algorithms, however it will enable you to hone in on which running parameters you need, and allow you to troubleshoot if the results produced by oscaar are not what you expected. There are two plot settings to choose from: `Track plots` and `Photometry plots`. `Track plots` will show you the centroid solutions as they are calculated in real time, along with some guide lines, see Figure 8 for an example.

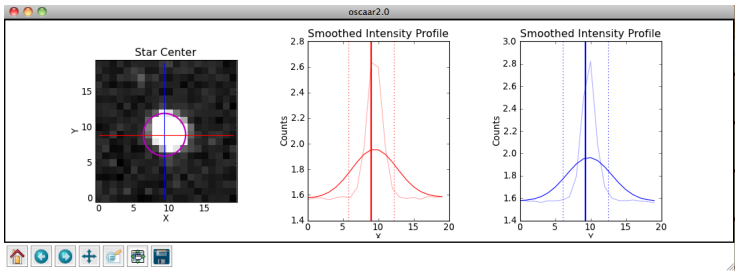


Figure 8: `Track plots` set to `On` and `Photometry plots` set to `Off`. The leftmost image shows the star with a cross-hair indicating the centroid, also circled in magenta (the radius of this circle is not meaningful). The plots on the right indicate the sums of the intensities in pixels along the rows and columns, in the transparent curves. The solid curves represent the smoothed intensity profiles, which are used to find the centroid. The bold vertical lines mark the best-fit stellar centroid in each axis.

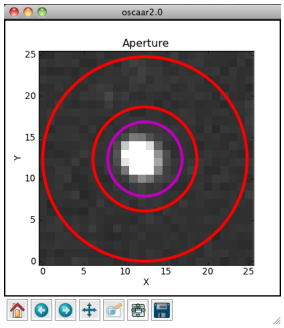


Figure 9: `Photometry plots` set to `On` and `Track plots` set to `Off`. The innermost circle (magenta) marks the `Aperture Radius` (see Section 3.5.5) measured from the stellar centroid found for this star by the tracking algorithm. All pixels that fall inside of this radius are summed to calculate the flux from this star. The next two outer concentric circles (red) circumscribe the "sky annulus," within which the sky background is measured.

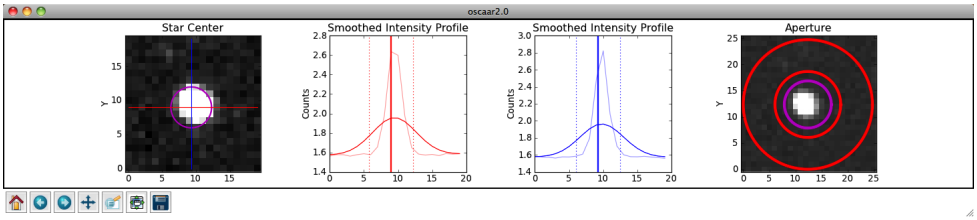


Figure 10: `Track plots` and `Photometry plots` both set to `On`. See Figures 8 and 9 for more details on each subplot. Note: the `Smoothing Constant` may be too large on this image — see for example how broad the smoothed profile is compared to the raw flux.

3.5.3 Track Zoom

OSCAAR tracks each star through consecutive exposures by looking for the each centroid near to the detector location that it had in the previous exposure. Since the stars will drift somewhat from frame to frame if your telescope is not perfectly aligned, we define a width around the previous centroid within which to search for the centroid in the following frame. The ideal width of this search box will depend on the plate scale of your device, how defocused the telescope is, how poor the alignment is, etc.

See, for example, Figure 8. The `Track Zoom` parameter here was 20 pixels, as you can tell by the width of the image sampled along both axes. The limiting factor in how small the `Track Zoom` parameter should be is how far the star will move between this exposure and the next. The columns/rows where the slope of the intensity profile is at extrema (see the vertical dotted lines in Figure 8) need to fall inside of the image when cropped down to the size of `Track Zoom`. If the stellar centroid moved by 10 pixels in the next exposure for the star and zoom in Figure 8, OSCAAR would not be able to find one of the extrema required to find the stellar centroid, and the tracking algorithm would not produce meaningful centroid estimates.

3.5.4 CCD Gain

The gain of a CCD is a hardware parameter that is specific to your CCD and the possible settings that you have set on it. Without getting too much into detector physics, CCDs essentially convert photons to electrons, amplify the number of electrons, and then count the amplified electrons. The number of amplified electrons counted is often more briefly referred to as the number of "counts", or less transparently as "analog-to-digital units (ADU)". If we want to be precise about how we propagate uncertainties, we need to know how many electrons were on each pixel before amplification, so the detector gain comes into our calculations.

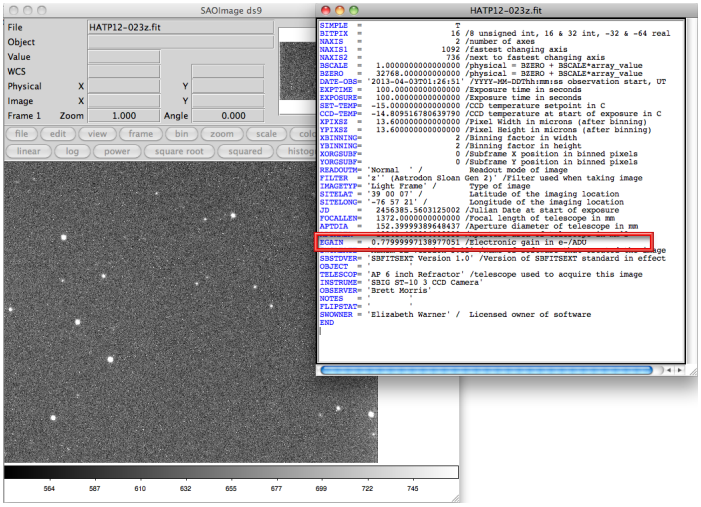


Figure 11: **Finding the gain**: Here's what the FITS header looks like when viewed in DS9, produced by Elizabeth Warner's 152mm refractor with an SBIG ST-10 CCD at the University of Maryland Observatory. The detector gain, labeled on this CCD as "EGAIN" is highlighted with a red box.

To find the gain of your detector, open one of your images in DS9. Click through the drop down menus

from `File > Display Fits Header`. The Fits Header is a text file within each FITS file that records some notes on every exposure you take. Often there will be a line called `EGAIN` in there, with a comment resembling "/ Electronic gain in e-/ADU". This is the gain parameter to enter into oscaar.

3.5.5 Aperture Radius

The `Aperture Radius` is the radius measured outward from the stellar centroid within which we will measure the flux from the source. The `Aperture Radius` should fully enclose the source, and if possible, not extend any farther than necessary (though determining what is "necessary" may require some experimentation).

In the duration of an observation, changes in earth's atmosphere can cause the "width" of a star on your detector to be larger or smaller than it was at the beginning of the night. If the star gets wider over time, the dim outer edges of the star may begin to exceed the boundaries that you set by the `Aperture Radius`, and some of the flux will not be counted. The differential photometry script would interpret this as the star getting dimmer, which would ruin your light curve. Therefore, when selecting the `Aperture Radius` parameter, we suggest that you experiment with it. Try one value with the `Tracking Plots` turned on (see Section 3.5.2 for details) and watch the apertures as they are applied to the images of the stars. Try a few different values (they need not be integers), and pick the smallest aperture that is sure to catch the whole star.

3.5.6 Notes

This text field is meant for you to enter any notes you might want to know later. All of your running parameters will be saved in the data file so you won't need to copy them here, but if you want to label the run in any particular way, there's a great little box here specifically for that purpose.

3.5.7 Ingress and Egress Times

If you are observing a transiting exoplanet, `OSCAAR` can do its most precise photometry if it knows when your target is "in-transit", meaning the planet is occulting the disk of star, or "out-of-transit", meaning the planet is not occulting the star. This is because it uses mathematical techniques that look for changes in each comparison star compared to the target star in order to determine which comparison stars are the best to use (see Section 5.3.2). If you compared the target to comparisons while the planet was in-transit, there would be a real (and important) difference between the two that we want to measure accurately. For this reason we input the ingress and egress times and only compare the target and comparison stars during out-of-transit exposures.

Enter the times in MM/DD/YYYY; hh:mm format, where the hours are on the 24-hour scale, in Universal Time. Seconds are insignificant.

Table 2: **Running Parameters Cheat Sheet**

Field	Definition	Example values
Smoothing Constant	Adjusts how much smoothing is applied in the tracking algorithm	2.0
Track Plots	Shows you the centroid solutions as they are calculated in real time	[Choose on/off]
Phot. Plots	Toggles real time photometry plots on or off	[Choose on/off]
Track Zoom	a width around the previous centroid within which to search for the centroid in the following frame.	20 pixels
Aperature Radius	radius measured outward from the stellar centroid within which we will measure the flux from the source	[Varies]
Depth Lower Limit	The calculator will filter out any transit events that have a smaller expected transit depth than this	0.005
Ingress and Egress, `YYYY/MM/DD`	The start and end date in Universal Time of the transit	2012/06/21

3.6 Results

After you click `Run` and `OSCAAR` has completed its differential photometry calculations, a light curve will pop up, like this one:

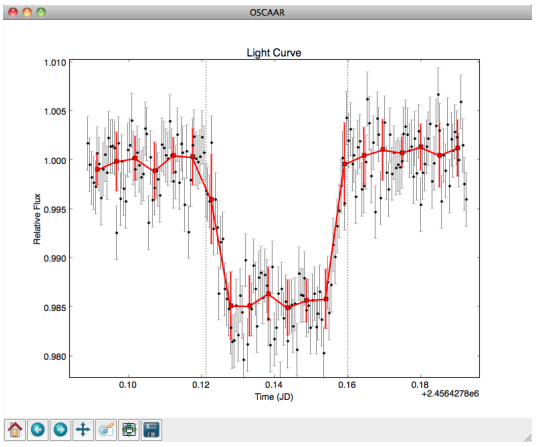


Figure 12: Sample light curve generated at the end of a successful `OSCAAR` run, using sample data produced by `generateSampleData.py`

The black points represent the relative flux of the target star compared to the comparisons, with some uncertainty (see Figure 12). The red squares bin up the black points into large bins so you can see the overall trends in noisy data. The vertical dotted lines mark the ingress and egress times that you set in Section 3.5.7.

The plot is generated with the Python package `matplotlib`, which supports some great interactive features. If you click the magnifying glass icon in the bottom left of the window, you can click anywhere on the plot and drag a box over a region that you'd like to zoom into. Press the home icon to return to the original view of the plot. Click the crossed-arrows icon to use the cursor to move the plot around in its box, or right click and drag to squeeze or stretch the plot. If you find that your experimentation has given you a view of the data that you'd like to hold on to, click the floppy-disk icon to save a hard copy of the plot (acceptable formats: ps, emf, gif, svg, eps, jpeg, raw, bmp, jpg, tiff, pdf, tif, png, and more).

3.7 Loading Previously Saved Results

There should now be a file where you placed the "Output Path". If you want to look to see the raw data, or the data in various stages of processing, you can click the "Oscaar" drop down menu (next to `File` and `Help` at the top), and select "Load old outputs". The small window that pops up allows you to pick the old output file that you generated and then plot its contents in various ways. This can be useful for troubleshooting, if your light curve doesn't look the way you think it should. Light curves can be bad if the raw fluxes of one of the stars was measured improperly, for instance, so clicking the "Plot Raw Fluxes" button will produce a plot of the raw fluxes for each star, which may help you to debug.

4 Light Curve Fitting

Once you've produced your first light curve, the next question to ask is: "what does it tell us about the planet?" The shape of the transit light curve is dictated by the physical properties of the planetary system, and by quantifying the shape of the transit light curve we can measure the radius of the planet, the distance from the planet to the star, and more. In a seminal paper in 2002 by Mandel & Agol, a convenient formalism was worked out for calculating analytical transit light curves given a few parameterizations of the physical properties of the system. Those parameters are:

R_p/R_s, the ratio of the radius of the planet to the radius of the star

i, inclination, the "tilt" of the planet's orbit from Earth's line of sight. $i = 90°$ is an edge on orbit (the planet would appear to cross the center of the star), and $i = 0°$ is a face on orbit,

a/R_s, the ratio of the semi-major axis of the planet orbit to the radius of the star,

t_0, the mid-transit time, the time (typically in Julian Date) half-way between ingress and egress,

γ_1 and γ_2, the limb-darkening coefficients, parameterize how the profile of the stellar brightness as it increases from the limb towards the center of the star.

A planetary system can be described with more parameters too, for example, if the orbit has non-zero eccentricity.

There are many ways to fit the analytical transit model to your data that you've produced with OSCAAR . Of course, you could always retrieve the raw data from the raw data file saved by OSCAAR , described in Section 5.3, and do a fit however you like. Alternatively, you can use the fitting module in OSCAAR to use Markov Chain Monte Carlo to fit transit light curves.

4.1 Introduction To Fitting With Markov Chain Monte Carlo (MCMC)

MCMC is a bit of a buzz-word in the sciences today, which can obscure how simple it is. Here we'll try to give you the most satisfying watered-down explanation of the procedure of MCMC fitting that we can. In this explanation, we've **bolded** some key phrases that will turn up in the GUI that you'll use to control the MCMC routines in OSCAAR . For a thorough explanation of MCMC from the go-to peer-reviewed paper on the subject, check out Ford (2005).

In any fitting process, you've got a bunch of fitting parameters that you need to wiggle around to find out which values for those parameters give you the best-fit model to your data. In more traditional least-squares fitting routines, you typically vary each parameter a little bit, check if the fits are getting better or worse, then keep moving in the "better" direction for each parameter until a best-fit solution is reached. This process is much like a visit to the eye doctor where the doctor tries a new lens, asks "better or worse?", he and you continue this process of changing lenses in the "better" direction until you narrow it down to one "best-fit" lens for your eye.

In MCMC fitting methods, rather than intelligently moving in a particular direction, we let randomness sample the parameter space for us (the randomness is where the "Monte Carlo" part of the name comes from). Let's imagine that we're trying to fit a line to some data, with a parameter for the slope and a parameter for the intercept. We'll start with an **initial guess** for the slope (m) and intercept (b), let's say our guesses are $m = 1$ and $b = 0$. MCMC will then get a (pseudo-)random number from a normal distribution centered on $m = 1$ for the slope, and a separate normal distribution centered on $b = 0$ (one distribution sampled per parameter). It may be greater than or less than the original parameter, and may be very close to one or significantly far away – let's say our new, random guesses are $m = 1.4$ and $b = -0.02$. We then do a χ^2 test to see if the new guess gives a better or worse fit than $m = 1$ and $b = 0$ did; if the new guess was better, then we now restart the process by randomly sampling numbers from a normal distribution centered on $m = 1.4$ and $b = -0.02$. If the new guess was worse, we discard that guess and start with new random samplings centered on the original parameters again. The "chain" part of the MCMC routine is this procession of parameters that have been sampled from one step (or "link") to the next. We'll use these chains, or series of trial values for each fit parameter, in a bit.

This process is typically repeated millions of times, and some work with statistics can show that if you continue the process for long enough, the parameters will slowly approach the "best-fit" solution. There's no way to know a priori what **number of steps** you'll need to iterate before you will converge on the true best-fit parameters, so running very long chains to be sure is common practice. We call the "time" or number of iterations needed to produce a good best-fit solution the *Markov chain mixing time*, and we define a *well-mixed chain* as one that that has been running for "long enough" (whatever that is) to produce solutions. One of the qualities of a chain that must be attended to is the **acceptance rate**, or the ratio of the number of new trial values that have been accepted and the total number of new values tried. According to Ford (2005), ideal acceptance rates can vary depending on the number of dimensions in the parameter space from ~ 0.25 to ~ 0.44.

One of the nifty things about MCMC when compared to other fitting methods is that it simultaneously gives you the best-fit parameters and the uncertainties on those best-fit parameters. We get those uncertainties by looking at the chains. When we've arrived at our solution for our linear fit, we'll have collected a long list of slope values that we tried along the way starting with our initial guess $[1, 1.4, ...]$.Typically, the early iterations

in these very long chains wander about the parameter space a lot before getting near the best-fit solutions, so when we analyze these chains, we ignore some fraction of the beginning of the chain. We call this the **burn**, for example, and we'll typically burn the first 20% or so of the chain.

Now we'll take the distribution of the values of the chain using that last 80% of the chain. The distribution of the values in the chain reflects the width of the parameter space near to the best-fit solution that produces fits nearly as good as the best-fit parameters do. Typically in transit light curve fits, the values in each chain will be normal distributions, and you can use those normal distributions to estimate 1σ uncertainties on the best-fit solution that you calculated. If the chain is not a normal distribution, you can have asymmetrical uncertainties (the "plus" uncertainty is not always equivalent to the "minus" uncertainty).

To save our computers some memory, we don't always choose to store every single step in each chain for later to compute uncertainties with, instead we set a **save interval** in our MCMC code store that saves the value of each chain at every nth step. As long as you chose a small enough number for the save interval, it won't effect your estimation of the uncertainties to be using a smaller bit of the original chain.

You may notice that this explanation has thus far neglected to cover how we generate the normal distributions that we sample to produce the chains. That's because `OSCAAR` will take care of this step for you, so if you don't want to, you don't have to know much more about that step to use `OSCAAR` . In short, you need to choose a width for these normal distributions for each parameter defined as the β for each fitting parameter. Choosing the wrong β will give you bad acceptance rates and the chains may not converge, but much like the number of steps in an MCMC routine, there is no a priori way to know what value for β is appropriate. Since choosing the proper β is so important, we've incorporated a tool in the `OSCAAR` MCMC module that finds good β values for you each time that you run a chain. It does this by running a short chain and testing the acceptance rate, tweaking the β and testing the acceptance rate again until the acceptance rate is within 10% of the one that you wanted.

4.2 Using `OSCAAR` 's MCMC Module

First, a note to our open source code readers (skip this paragraph if you just want to learn to use the GUI): Since MCMC chains run for so many steps, they can take a long time to compute. To minimize that time and maximize the legibility of the code, we've implemented most of the MCMC algorithm in frequently-commented `Python`, but we wrote a function that produces Mandel & Agol (2002) analytical transit light curve models in `C`, since that one function is called many millions of times per chain. Since this function is a direct implementation of the formalism laid out in Mandel & Agol (2002), we don't feel too bad making that bit of code less transparent for our non-`C` fluent users, since they are likely equally fluent in the math described in that paper, and writing that function in `C` reduces the runtime of any given chain by several orders of magnitude.

To begin using `OSCAAR` 's MCMC module, go to `OSCAAR > Fitting Routines`, enter an `OSCAAR` output (`.PKL` file) and click `MCMC Fit`. There may be other buttons next to that one in the near future if new fitting methods are implemented in `OSCAAR` . You'll then get a new frame containing a slew of options that you can set to begin your MCMC run. Begin by selecting the aperture radius which produced the light curve that you'd like to fit from the `Select Aperture Radius` drop-down menu.

Like in any fit, this MCMC fitting routine requires initial guesses for the fitting parameters. This routine allows you to fit for R_p/R_s, a/R_s, i and t_0 (see intro to Section 4 for definitions), and you must enter initial estimates for these parameters. If the planetary system that you observed is in the exoplanets.org database, you can simply enter the name of the planet in the `Planet Name` box and click `Update Parameters` to fill in peer-reviewed results from a journal article for the four fitting parameters and the orbital period. If the system

you're working on has not yet been added to the database, you can add the guesses by hand. Each fitting parameter has a corresponding initial β guess, but you should be able to leave these at the defaults. `OSCAAR` will determine the proper β values for you after you initialize the run.

You can then set the save interval, fraction of steps to burn from the chain, goal acceptance rate of the run, and total number of accepted iterations. Some example values for these parameters are shown in the table below. Then, you can press the `Run and Plot` button to initialize the fitting routine. The first thing that you should see is an animated plot that adds colored dots in successive iterations. We've sort of "gamified" the process of attempting to find ideal β values by just showing you this plot rather than any numbers. If the colored points are moving closer to the zone in between the dotted red lines on the plot, then the βs will eventually produce the goal acceptance rate, and you can just wait until they do enter the space between the red dotted lines. When they do, the MCMC run will begin and you'll see a loading bar. If they never do, you should start the run again with different βs or different initial guesses for the best-fit parameters. The most likely problem for non-converging βs is that one of the initial guesses was incorrect.

Table 3: **MCMC Cheat Sheet**

Field	Definition	Example values
Planet Name	exoplanets.org name of target planet for loading presets	HAT-P-7 b
Rp/Rs	Ratio of the planet to star radius	0.11
a/Rs	Ratio of semi-major axis to star radius	14.1
Inclination	Inclination of the planet's orbit in degrees	89
t0	Mid-transit time in Julian Date	2456427.94255932
Period	Orbital period of the planet in days	2.2057
gamma1	Limb-darkening coefficient, linear	0.3
gamma2	Limb-darkening coefficient, quadratic	0.2
Eccentricity	Orbital eccentricity in degrees (usually assume 0)	0
Pericenter	Longitude of pericenter in degrees (usually assume 0)	0
Save interval	Save every *n*th step in the chain	500
Burn Fraction	Fraction of steps to burn from chain for uncertainty calculations	0.20
Acceptance	Goal acceptance rate	0.30
Number of steps	Total number of accepted steps to complete	5000000

4.3 Understanding MCMC Results

5 Algorithm Notes

This section is meant as a sort of appendix to give you lots of details on a few methods within the OSCAAR source code that are important to the photometric process. The LaTeX format of this document allows us to easily typeset equations and figures onto nicely spaced pages, so some algorithm descriptions are best explained here. However if you're looking to do some hardcore coding with OSCAAR, you may be looking for a long, expansive dictionary of most methods in the OSCAAR source. If that's what you're looking for, head over to oscaar.readthedocs.org.

5.1 `oscaar.photometry.phot()`

5.2 `oscaar.astrometry.trackSmooth()`

The "**point spread function**" (PSF) for a particular observing setup is the shape of the image of a perfect point source (like a star, for example), which in practice is never a perfect "point" — it will always have some radial spread. For well-focused telescopes, the PSF usually resembles an Airy function, which can be well-approximated by a Gaussian. However, as suggested in Section 2.3, you may not always want to focus the telescope perfectly. Each telescope will produce a different PSF when significantly defocused, so assuming a Gaussian PSF would prevent us from using `OSCAAR` on defocused observations. This was a problem that we found with `OSCAAR` v1.0, so we developed a new method with some excellent advice from Professor Drake Deming (UMD).

`trackSmooth()` does centroid-finding by summing up the intensity of the pixels near the star along both the rows and columns. The sums along the rows and columns will produce intensity profiles in two axes similar to Figure 6. One feature of these profiles that holds for even strongly defocused PSFs is the sharp rise and decline in the intensity of the star on either side of the stellar centroid. Typically there is a well-defined absolute maximum and minimum in the first numerical derivative of the sum along the columns, as in Figure 13. The trick here is to take the midpoint between those extrema as the centroid, since searching for maxima and minima are computationally cheap.

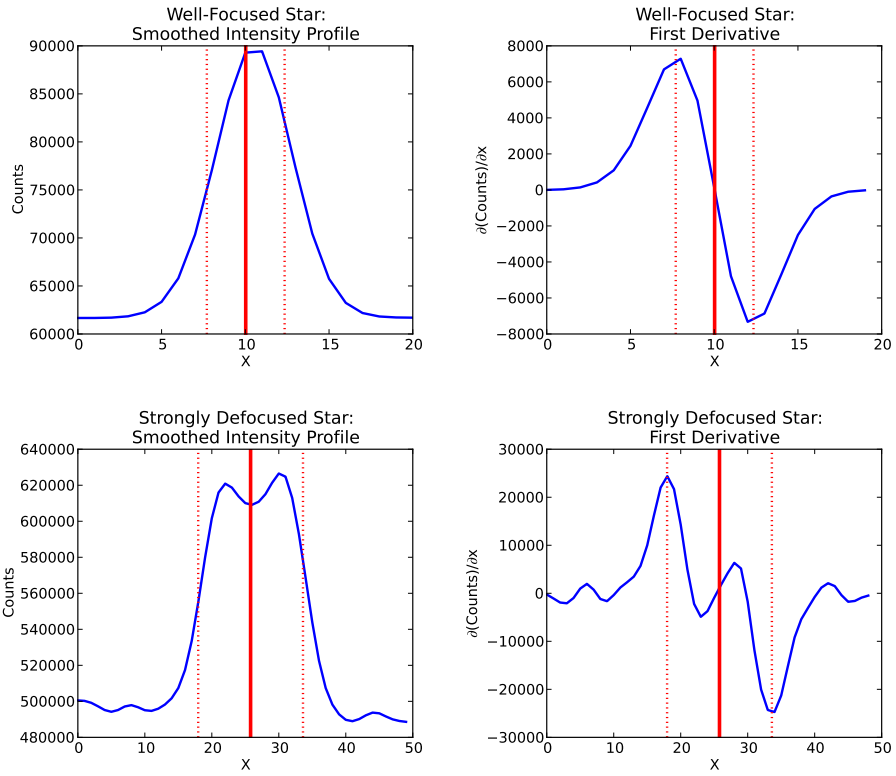


Figure 13: **Stellar Centroid Tracking**: The top row of plots show the sum of intensities in each pixel along the columns a well-focus star, similar to Figure 6, and the bottom row shows the same for a highly defocused star (see discussion on defocusing in Section 2.3). The plots on the left show the counts on the detector summed along the columns of pixels, and smoothed with non-zero smoothing constants. The plots on the right show the first spatial derivative of the intensity profiles, with clear extrema at the pixel locations corresponding to the points of greatest positive and negative slope in the intensity profiles. A quadratic is fit to the three points closest to each of the extrema, yielding sub-pixel precision on the coordinates of the extrema (marked above with the dotted vertical red lines). The midpoint between these best-fit extrema is taken as the stellar centroid, marked with the solid vertical red line in the plots above. This process is repeated for the other axis (sum along the rows) to get the other of the two dimensional centroid coordinates for each star.

In order to increase our centroid precision, we then use a little linear algebra to fit a quadratic to the three pixels nearest the maximum or minimum. The midpoints between the apexes of the best-fit quadratic near the maximum and minimum is taken as the centroid. This process is repeated for the sums along the rows and the columns to get the centroid in both axes.

Observational data is never clean. There may be dead pixels or hot pixels that would appear as sharp spikes or troughs in the intensity profiles. These pixels, if unaccounted for, will appear to the tracking algorithm to be the sharp edges of a star in the image. This is why we run the image through the smoothing routine — to smear out any sharp artifacts in the image, so that the tracking algorithm only anchors itself on the overall features of the image and not faulty individual pixels. This smoothing routine can apply varying degrees of smoothing to

each image, and choosing a proper amount of smoothing is covered in Section 3.5.1.

5.3 The `dataBank` Object

Storing and managing all of the data that comes with differential photometry scripts can be a nightmare, so we've developed the `dataBank` class to keep things organized. Python is an *object oriented* language, which means we can make *objects* in it. Objects in `OSCAAR` allow us to organize our data and the functions that we will use on those data (called *methods*).

One of the goals of `OSCAAR` is to have lots of different methods for each analysis task so that you can try different ways of reducing your data without having to change out large chunks of code. If you used `trackSmooth()` to find stellar centroids (covered in Section 5.2), but thought a routine that fits two-dimensional gaussians to the images of the stars would work better, you could write a method for the `dataBank` class that would take the same input parameters as `trackSmooth()` and return the same outputs. Then all you would have to do is change the word "trackSmooth" to the name of your gaussian fitting method in the `differentialPhotometry.py` script to use the new algorithm. All of the other parts of the code could stay the same and it would be easy to change back to `trackSmooth()` if you wanted to.

In the next subsections, we'll discuss some of the methods in the `dataBank` class. Since all of the photometry data is stored in this class, most of the differential photometry algorithms that manipulate the raw fluxes to calculate a light curve are written in this class as well. Therefore this section can also be though of as a play-by-play description of the standard `OSCAAR` differential photometry routine.

5.3.1 `dataBank.scaleFluxes()`

Each star that you observe will have a different apparent magnitude, and the dimmer the star, the more significantly the flux measurements will be affected by sources of noise. If we're going to compare the flux variations of each comparison star to the target star, which are all affected by noise to different degrees, we need to make sure that we're accounting for the lower signal-to-noise ratio for the dim stars and higher signal-to-noise ratio for the bright ones.

To accomplish this, we do a linear regression to find a multiplicative coefficient to apply to each comparison star. Multiplying this coefficient to the raw fluxes of the comparison stars "scales their fluxes" to the amplitude of the raw fluxes of the target star. It equally scales the scatter of those flux measurements, which has the effect of squeezing down the scatter on brighter stars after scaling or stretching out the scatter in the fluxes of dimmer stars. Figure 14 shows the raw fluxes from three simulated stars, calculated by `OSCAAR` , each with a different brightness. You can see the slight decrease in brightness of the target star, shown here in red, at ingress and egress (denoted by the vertical dotted lines).

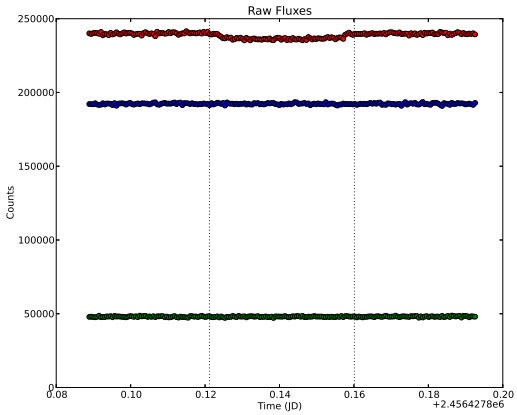


Figure 14: Raw fluxes from three simulated stars, calculated by `OSCAAR` , each with a different brightness. You can see the slight decrease in flux of the target star, shown here in red points, at ingress and egress (denoted by the vertical dotted lines).

When the fluxes of these comparison stars in Figure 14 are run through the `dataBank.scaleFluxes()` routine, their fluxes get stretched to the amplitude of the target star fluxes, shown in Figure 15. You can see that the scatter in the green comparison star's fluxes are stretched significantly larger, conserving the relative signal-to-noise ratios of each star. Now we could do a χ^2 or similar analysis to compare how closely the variations in each each comparison star's fluxes match those of the target star.

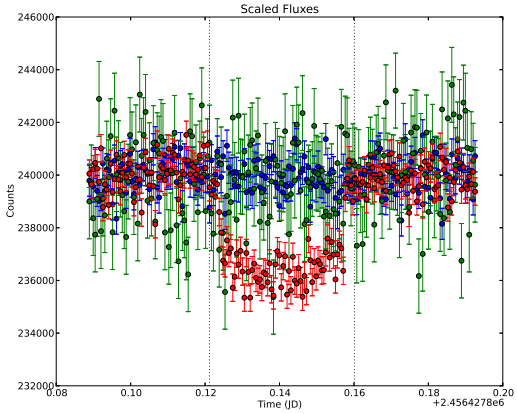


Figure 15: Result after scaling the raw fluxes in Figure 14 using the `dataBank.scaleFluxes()` method. Note that the fluxes of the dimmest star have the largest rms scatter, and the brightest stars have the smallest scatter after scaling.

5.3.2 `dataBank.calcMeanComparison()`

In order to make a transit light curve, we want to divide each flux measurement of the target star by the flux of a star that has no intrinsic flux variations, and ideally a star of similar spectral type to the target. How are we to know which star is the right one to use as a comparison star? To make matters worse, your detector may not be evenly responsive over the whole of the imaging surface, so stars that fall near or far away from the target star on the detector may be better to use as comparisons than others.

We often take the mean of a bunch of comparison stars so that they can collectively make a "composite" comparison star, or "mean" comparison star, in the wording of the name of this algorithm. If one star varies a little bit or falls on a worse part of the detector, it may not introduce significant variability into the mean of the comparison stars since it gets diluted by all the non-varying fluxes of the other stars.

Let's consider the case with two comparison stars. Ideally, we would have to perfect comparison stars imaged by a perfect detector. Their spatial positions on the CCD would not affect the measurements of their fluxes and they would have no intrinsic flux variations. In this case, the vector of mean comparison star fluxes $\mathbf{f}_{\bar{C}}$, (though unnecessary) could be produced by taking the fluxes of each comparison star $\mathbf{f}_{C,0}$ and $\mathbf{f}_{C,1}$, and taking the average:

$$\mathbf{f}_{\bar{C}} = a \cdot \mathbf{f}_{C,0} + b \cdot \mathbf{f}_{C,1} \tag{2}$$

where a and b here are scalar multiplier coefficients for the vectors of fluxes, with $a = b = 0.5$ in this case. We would then proceed to calculate the light curve $\mathbf{f}_{relative}$ by simple division

$$\mathbf{f}_{relative} = \frac{\mathbf{f}_T}{\mathbf{f}_{\bar{C}}} \tag{3}$$

where $\mathbf{f}_T$ is the vector of target fluxes.

Now let's say that the image of the first comparison star fell on a few dead pixels on the detector, or that it has some slight intrinsic variability that does not make it a good comparison star. If we were to use Equation 2 with $a = b = 0.5$ we'd be considering the first and second comparison stars with equal weight in making the mean comparison star. A more useful "mean" comparison star wouldn't use the mean of the two comparison stars, rather it would take some other linear combination of the fluxes with different coefficients to make the composite comparison star $\mathbf{f}_{\bar{C}}$. In this case, we would multiply the first comparison star fluxes $\mathbf{f}_{C,0}$ by a small coefficient, perhaps $a = 0.2$, and since $a + b = 1$, $b = 0.8$. This will "put a heavier weight on" the fluxes of the second comparison star in the calculation of the mean comparison star, and should produce a better light curve.

The `dataBank.calcMeanComparison()` algorithm takes the fluxes of each normalized comparison star with fluxes $\mathbf{f}_{C,i}$ and tries different weights on each star. It will minimize the difference between the out-of-transit portions of the target star fluxes and the mean comparison star fluxes by varying a set of coefficients $\mathbf{c}_i$ that get multiplied by each comparison star flux such that $\Sigma_i c_i = 1$.

$$\begin{aligned} \mathbf{f}_{\bar{C}} &= c_0 \cdot \mathbf{f}_{C,0} + c_1 \cdot \mathbf{f}_{C,1} + c_2 \cdot \mathbf{f}_{C,2} + ... && (4) \\ &= \Sigma_i \; c_i \cdot \mathbf{f}_{C,i} && (5) \end{aligned}$$

The vector of the uncertainties of the composite comparison star fluxes $\sigma\mathbf{f}_{\bar{C}}$ is then

$$\begin{aligned} \sigma\mathbf{f}_{\bar{C}} &= \sqrt{(c_0 \cdot \mathbf{f}_{C,0})^2 + (c_1 \cdot \mathbf{f}_{C,1})^2 + (c_2 \cdot \mathbf{f}_{C,2})^2 + ...} && (6) \\ &= \sqrt{\Sigma_i \; c_i^2 \cdot \mathbf{f}_{C,i}^2} && (7) \end{aligned}$$

Of course, the out-of-transit exposures are the only ones used in this calculation because there is an important intrinsic flux variation in the target star during transit (that we're trying to see!). If we ran this regression technique to compare the target star to comparison stars on exposures taken during the transit, the algorithm would find the linear combination of target stars that best match the variability of the target star, and it would effectively seek to cancel out the transit. This is why we enter the ingress and egress times into OSCAAR (see Section 3.5.7), so that we can identify which exposures to use in these comparisons.

This method also effectively cleans up after us if we choose bad comparison stars, as laid out in Section 3.3. Let's say we chose a dim star that had a brighter nearby companion. The tracking algorithm may be able to accurately find the centroid of the dim star that we intended to use as a comparison star for a few exposures or maybe even for half of the exposures, but it may eventually "see" the brighter star nearby and start tracking its centroid instead. When this occurs, the flux measured for this comparison star will suddenly jump up discontinuously at the exposure when the centroid of the brighter star was found. The discontinuous flux jump in fluxes of this comparison star will be very dissimilar to continuous, telluric variations that effect the target star on a night with decent weather. The `dataBank.calcMeanComparison()` algorithm will find that only very small coefficients for the flux of this comparison star will produce a good composite comparison star, essentially giving this star no weight in the composite comparison star, counting it out.

5.3.3 `dataBank.computeLightCurve()`

We calculate the light curve vector $\mathbf{f}_{relative}$ by simple division of the flux of the target star $\mathbf{f}_T$ by the flux of the composite comparison star $\mathbf{f}_C$, generated using `dataBank.calcMeanComparison()` (see Section 5.3.2).

$$\mathbf{f}_{relative} = \frac{\mathbf{f}_T}{\mathbf{f}_{\bar{C}}} \tag{8}$$

The errors for each light curve flux are calculated by propagation of uncertainties. If $\sigma\mathbf{f}_{\bar{C}}$ is the vector of uncertainties in the composite comparison star fluxes and $\sigma\mathbf{f}_T$ is the vector of uncertainties in the fluxes of the target star, the uncertainties in the light curve fluxes $\sigma\mathbf{f}_{relative}$ are

$$\sigma\mathbf{f}_{relative} = |\mathbf{f}_{relative}|\sqrt{\left(\frac{\sigma\mathbf{f}_T}{\mathbf{f}_T}\right)^2 + \left(\frac{\sigma\mathbf{f}_{\bar{C}}}{\mathbf{f}_{\bar{C}}}\right)^2}. \tag{9}$$

6 Troubleshooting

If you notice a problem in OSCAAR or can't get something to function properly, feel free to submit an issue on our Issue Tracker on GitHub.

7 Contributing to OSCAAR

This version of OSCAAR was made by a very small group people, and we're proud of the work we've done. But OSCAAR can still be improved and we need your help! OSCAAR is used around the world by amateurs and professionals alike, and in order to keep up with the demands of providing a user-friendly differential photometry code for an international audience, we'd love to have your help if you can code in Python, or provide any feedback at all. If you'd like to help but don't know where to begin, please feel free to contact us at oscaarteam@gmail.com.

OSCAAR started as one of Brett Morris's independent undergraduate research projects at the University of Maryland, and since then has fueled independent research projects for several other undergraduates. If you are an undergraduate studying astronomy, physics or computer science and would like to contribute to oscaar, we encourage you to reach out to us. We'd love to work together!

We keep the source code on GitHub, a popular open source code repository site. There you can find the code in its most up-to-date (alpha, beta, and stable) form, the Issue Tracker where known issues are logged and new issues or comments can be posted.

8 Acknowledgements

OSCAAR has come a long way from the first iteration that Brett Morris wrote in 2011. It could not have gotten there without the help of the following advisors and colleagues: Professor Drake Deming (UMD), Dr. Avi Mandell (NASA-GSFC), Daniel Galdi (UMD), Sam Gross (UMD), Luuk Visser (UL/TUD), Harley Katz (UMD), Elizabeth Warner (UMD), Professor Alberto Bolatto (UMD), Dharmatej Mikkilineni (UMD).

B.M.M. is grateful for funding that fueled a portion of this work from Dr. Avi Mandell (NASA-GSFC) and the Goddard Center for Astrobiology at NASA's Goddard Space Flight Center via the NASA Astrophysical Data Analysis Program.