oscaar: Open Source Differential Photometry Code for Amateur Astronomical Research

Brett Morris

September 17, 2011

Contents

1 Introduction

`oscaar` is a differential photometry package built for astronomy students or amateur astronomers who want to observe astronomical objects with time-varying intensity. This could include transiting exoplanets, variable stars, rotating asteroids and more. The code was designed in the summer of 2011 to take a series of CCD images obtained at the University of Maryland Observatory and churn out light curves of exoplanet transits. The core of the code is written in Python, but it has been designed for users who have never seen Python before.

1.1 System Requirements

This software package runs on the following required software and modules:

Python 2.7.1: The main language this software is written in
NumPy 1.6.0-py2.7: A Python module for array handling
PyFITS 2.4.0: A Python module for FITS handling
Minuit 1-7-9 (and pyMinuit): A Python module for minimization
Matplotlib 1.0.0-py2.7: A Python module for plotting
SAO Image DS9: A standard astronomical FITS display interface

It is assumed that the user has access to a Unix shell through which they can execute this program.

1.2 Disclaimer

Though the modules used in this package have been historically stable and probably will not go away any time soon, I do not take any responsibility for helping you find these modules or for problems that arise as a result of changes to future versions of these modules. Sorry!

2 Collecting Data

2.1 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 an unreasonable challenge for most small telescopes to track perfectly over several hours, so `oscaar` is built to keep track of the drift in star positions on

the detector in your data. Just make sure your object of interest stays in the field throughout the whole observation.

2.2 Exposure Length

Chose an exposure length that avoids saturation of stars in the field that you will use for photometry. As an arbitrary rule of thumb, avoid doing photometry with any sources that are near or above 75% of saturation for good results. If you tell `oscaar` to calculate differential magnitudes of a star that is too close to saturation, `oscaar` will ignore that star.

If your tracking is poor and stars smear across the images, make sure to choose a shorter exposure length to keep the stars as point-like as possible. The code looks for Gaussian sources, and if the star profiles are too elongated `oscaar` won't recognize them.

Since noise can often obscure transits in non-professional observations, light curves are frequently presented with median binning of the data show the overall trend without distracting noise. You may want to consider this when picking your exposure length so that you will have a sufficient number of median points to resolve the transit you are trying to detect. If you use `oscaar`'s GUI to display the data, you'll be able to turn on and off a median binning routine that does this for you.

2.3 Dark Fields and Flat Frames

Dark frame and flat field collection is standard practice for removing systematic effects from CCD images. Flat fields correct for dust and other inconsistencies in the optical path that will effect your data. Dark frames correct for 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. Collecting good flat fields can sometimes be more of an art than a science, but good flats are important for good photometry. Take 1-5 dark frames and flat fields 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.

2.4 Picking a target

When you chose your target for differential photometry, you need to be sure there are other stars in the imaging field. Here we'll define some terms that are important from here on:

Test Star: The host star to an exoplanet or a variable star that you're testing for intrinsic variations in luminosity.

Control Star: A star other than the test star that you will use as a basis for determining the intrinsic variations in the brightness of the test star.

There must be at least one control star in order to do differential photometry, and the more the better. There is no magic number of control stars to have, but if you have the opportunity to fit more control stars in the field by rotating your CCD or effectively "zooming out," it will be worth your while. Based on prior experience with `oscaar`, ∼25 stars should do.

Photometry of bright stars is always preferable to dim stars, but of course there are more dim stars than bright stars in most of the images you will take. In order to avoid error caused by noise with very dim stars, pick control stars with peak intensities more than double the average background intensity in the area surrounding the star.

Different transiting exoplanets change their hosts' luminosity by different amounts. This parameter is called "**depth**" and is measured in units of millimagnitudes (mmag). The greater the depth of a transit, the more likely you will be capable to detect the transit. Since some of these depths are so small, it might is a good idea to perfect measuring short period variable stars with high amplitude luminosity oscillations (like YZ Boo) before you move on to exoplanets. Once you characterize your ability to measure the large intrinsic variations of variable stars, you will be able to characterize your ability to detect small depth transiting exoplanets.

3 Running `oscaar`

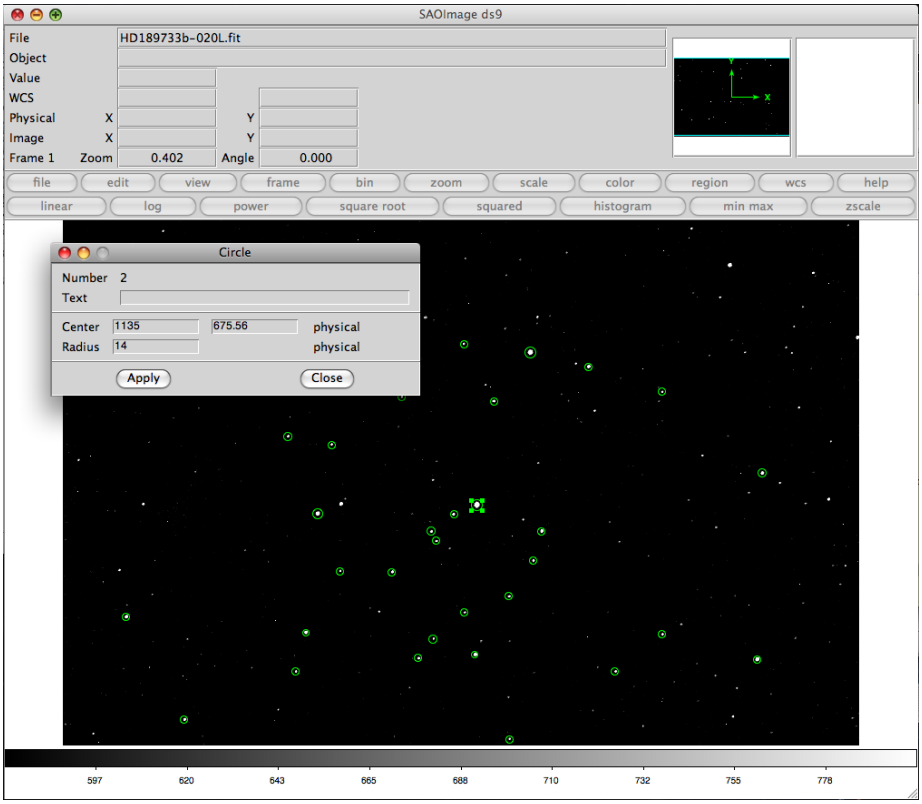
3.1 Locating Your Stars with DS9

The first and most intensive 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 hundreds of stars in each image, some of them close to the edges, some of them with close binaries; 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, they are entered by the user in DS9.

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`), first double click on the test star (the variable or exoplanet host). A green circle will appear over the star along with a dialogue box which contains the pixel coordinates of the circle's center and radius.

The region radius will indicate to `oscaar` how much space around the center

of the star to consider when tracking the star. Try to set this radius so that the star is inside the circle with some extra room. 10-15 pixels is typically good. The smaller the radius you pick, the quicker the code will run, but it is important to capture the entire star in this radius.



Screenshot 1: Using DS9 to locate test and control stars

Repeat this process for as many control stars as you like (try to keep it under 30 for speed). The code will differentiate between the test star and the control stars chronologically, so as long as the first star you circled was your test star, the order no longer matters when picking the controls. Avoid picking stars near big dust spots on the images, stars less than 150 pixels from the edge of the image, or stars with another star 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. 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 line doesn't have the right coordinates, find the line that does and move that line to the top of the list.

3.2 The Initial Parameters File

In the running directory of the code, there is a file called `init.par`. This file is where all of your interaction with the code happens. It is written in an original syntax that I designed. It is intended to be a user friendly means of directing `oscaar` without having to know any Python.

Each line is a statement of what feature you would like to turn on or off, or what number you'd like to set a running parameter to. Below are explanations of all of the features you can access and the options that they accept. Do keep in mind that **these entries are case sensitive**.

The subsections that follow describe each of the commands in `init.par`. Once these are tweaked to your liking, you will be ready to run the code.

3.2.1 Locating Your Data

`oscaar` is built to find files by their paths. If you name the sky image files "TrES3-001.fit," "TrES3-002.fit" ... "TrES3-160.fit" and name the dark frames "dark-001.fits," "dark-002.fits" ... "dark-005.fits," it will be easy to set up the code to differentiate between those files. If the files are not already easily distinguishable, make them look more like the ones above now.

The first lines of `init.par` concern the locations of the files you will be analyzing. They are:

`darkLoc`: The path to the dark frame files
`flatLoc`: The path to the flat field files
`imagLoc`: The path to the series of sky images for photometry
`regsLoc`: The path to the regions file from DS9

These commands expect a path as the input. For example, I could indicate where my dark frames are if they are named in the scheme from the example above with the command:

```
darkLoc /Users/bmorris/Desktop/Exoplanets/20110608/darks*
```

The code will read your input just like any Unix shell, so you can use `*`'s, and `?`'s creatively to ensure that you're getting the right files for each set of images that you are trying to point to. If you want to check that you will be

pointing to the right images, you can open a Unix shell and try

```
ls /the/path/you/want/to/try
```

and see if it returns a list of the appropriate files.

3.2.2 Star Tracking

The next set of options controls the star tracking algorithm. Since very well-aligned telescopes generally track imperfectly over a few hours, in most runs stars drift across the detector slightly. `oscaar` will take the initial star positions that you saved in the regions file from DS9 and track the stars as they move across the CCD using a Guassian fitting routine.

This bit of code is the slowest portion of the code, as fitting a 20x20 element, two dimensional Gaussian to noisy data is never an easy task. Once you do the tracking on your stars using `oscaar`, you can reuse the tracking information if you need to redo the aperture photometry or differential photometry (I'll talk about those next). If you don't delete the output produced by the tracking part of the code, you won't have to run it again and spend time waiting for fits. This nifty independence of each part of the code will be a common theme in the rest of these instructions.

To turn the tracking algorithm on or off, simply type `on` or `off` in lower case letters after the word track, ie:

```
track on
```

And it's as simple as that. If you'd like to see how the code works a little deeper or troubleshoot a problem that you think might be related to tracking, you may want to see a plot of the Gaussian fitting as it is performed. If you are adventurous and would like to try it, throw in

```
trackplot on
```

The resulting plots are three dimensional and can be rotated and manipulated. The red wire-frame is the data and the blue surface is the fit.

In order to help the fitting algorithm find the object faster, enter an estimate for the sigma parameter, corresponding to the radial width of the star in pixels, using the estsig parameter:

```
estsig 2.0
```

If you try to start a run and the fitting algorithm is barfing errors like those shown in Section 3.3.1, you likely need to pick a new estsig value.

3.2.3 Aperture Photometry

In this documentation I've talked about "aperture photometry" and "differential photometry" as if they are different things that are intimately related. Aperture photometry is the process of looking at the intensity of a star by determining the flux through a digital aperture. Differential photometry is the process of taking instrumental magnitudes obtained from aperture photometry of different stars, averaging them and comparing them to another star to look for variation of one star over time. This section will briefly discuss using the algorithm that does the aperture photometry.

Let's turn on the aperture photometry algorithm:

```
aper on
```

The fit produced in the star tracking stage has two important output values: an (x,y) pixel coordinate for the center of the star and a σ parameter determining the radial width of that star. The code will consider a "source" and "sky" aperture around the star based on a scaling parameter that you enter. The default is to make a source aperture 5.5 the σ of the source, so let's set this aperture radius:

```
aprad 5.5
```

Just in case one of the control stars was too close to saturating the CCD, there is an extra check built in to the code to ignore stars more than 85% of the saturation limit. Since each CCD could have a different saturation count limit, you can enter the limit on your detector into the `init.par` file as follows for a CCD with a 60,000 count limit:

```
satur 60000
```

If you were to try to compare photometry between different CCDs, you would need to keep track of the different gains (sometimes represented as a "K") of each device. In order to make the measurements as consistent as possible across different devices, there is a gain parameter for you to specify. If you do not know the gain for your device, look at the header information of any of your FITS files recorded by this device. The gain is noted in the "EGAIN" field. For my SBIG ST-10, I'll set it to:

```
Kccd 0.7799
```

Note the upper case `K`, this is the only in the upper case letter in the `init.par` file.

Similar to the tracking algorithm options, there is an aperture photometry option for displaying all plots. This is not recommended for common use, but if you need to troubleshoot a problem or check that your aperture radius is appropriate, you may want to see a few plots controlled by the `aperplot` parameter. These plots will show each star as the aperture photometry algorithm is being applied to it, along with two red circles representing the source aperture radius (controlled by `aprad`) and an outer circle showing the sky aperture between the source radius and some further radius that will be used to find the background intensity. To turn on these plots:

```
aperplot off
```

3.2.4 Differential Photometry

Finally it is time to tell `oscaar` to do the differential photometry. The differential photometry algorithm is activated with

```
diff on
```

which calculates the magnitude of the test star with respect to the control stars. The process is repeated to compare each control star against all of the other control stars, as well. This way you can visually inspect the control stars to make sure they aren't variable.

All that's left to do now is to visualize the data! Turn on the built-in GUI with

```
diffgui on
```

Instructions for using the GUI can be found in Section 3.4.

3.3 Initializing `oscaar`

`init.par` is now all set up. `oscaar` is ready to run! To run oscaar, open a Unix shell and change directories to the oscaar directory. Execute the Python script that controls `oscaar`:

```
$ python photom15.py
```

This file is specific to `oscaar1.1.0`. Different versions have "photom.py" files with different numbers.

When the code is running, the following will be printed to standard out:

```
Loading and averaging dark frames...
IMAGE 1/353, STAR 1/3************************
{'amp': 20953, 'sigma': 2, 'xcenter': 15, 'ycenter': 13, 'offset': 435}
IMAGE 1/353, STAR 2/3************************
{'amp': 1143, 'sigma': 2, 'xcenter': 9, 'ycenter': 8, 'offset': 391}
IMAGE 1/353, STAR 3/3************************
{'amp': 591, 'sigma': 2, 'xcenter': 9, 'ycenter': 9, 'offset': 390}
⋮
```

For each star, the `oscaar` runs the tracking algorithm and prints the output from the Gaussian fit, then runs the aperture photometry algorithm. The important fit parameters are displayed so you can check that they are reasonable while the code is running. Here the values are abbreviated as integers for clarity, though they will appear as long decimals ("floats" in Python speak) in your standard out.

3.3.1 Recognizing Errors

Sometimes the script will return something resembling the following:

```
IMAGE 76/353, STAR 2/3************************
eigenvalues:
-0.0322264
0.631025
0.960443
1.12215
2.31861
matrix forced pos-def by adding 0.034545 to diagonal
{'amp': 1757, 'sigma': 1, 'xcenter': 11, 'ycenter': 9, 'offset': 324}
```

This indicates that the fitting algorithm had some difficulty, but it eventually found the proper fit. It is OK if this happens in bursts as `oscaar` analyses a bad image, as long as the output eventually looks normal again.

It is **not** OK if the following is printed:

```
IMAGE 1/353, STAR 1/3************************
⋮
invalid value encountered in double_scalars
MnHesse: 2nd derivative zero for parameter 0
MnHesse fails and will return diagonal matrix
MnHesse: 2nd derivative zero for parameter 0
MnHesse fails and will return diagonal matrix
ERROR: Fit not found for this frame.
```

As mentioned earlier, if you choose a bad `estsig` value in the `init.par` file, this error may be the fitting algorithm dying as a result.

If a star is not located where the `Regions` file tells `oscaar` to look, the fitting algorithm will fail with the error above. If this happens you should kill the script with `Ctrl+C`. Open your `Regions` file in DS9 and check that the star that failed is properly marked, and open the image where the code failed and look for irregularities (like a plane or satellite passing in the image, the tracking on your telescope dying, the star running off the edge of the detector, etc.).

If the image is bad for any reason, you can rename that image so that it no longer matches what the `init.par` file is looking for and it gets skipped over. Then start the script again. If you do this to many consecutive images, the tracking algorithm might not be able to track the stars before and after the break. If this is the case, you may need to do the `oscaar` analysis in separate chunks with unique `Regions` files corresponding to before and after the break.

If you can't find anything that seems to be wrong, it is best to discontinue use of this control star. Open the regions file, go to the `circle(x,y,r)` line corresponding to the failed star and comment it out with a preceding `#`:

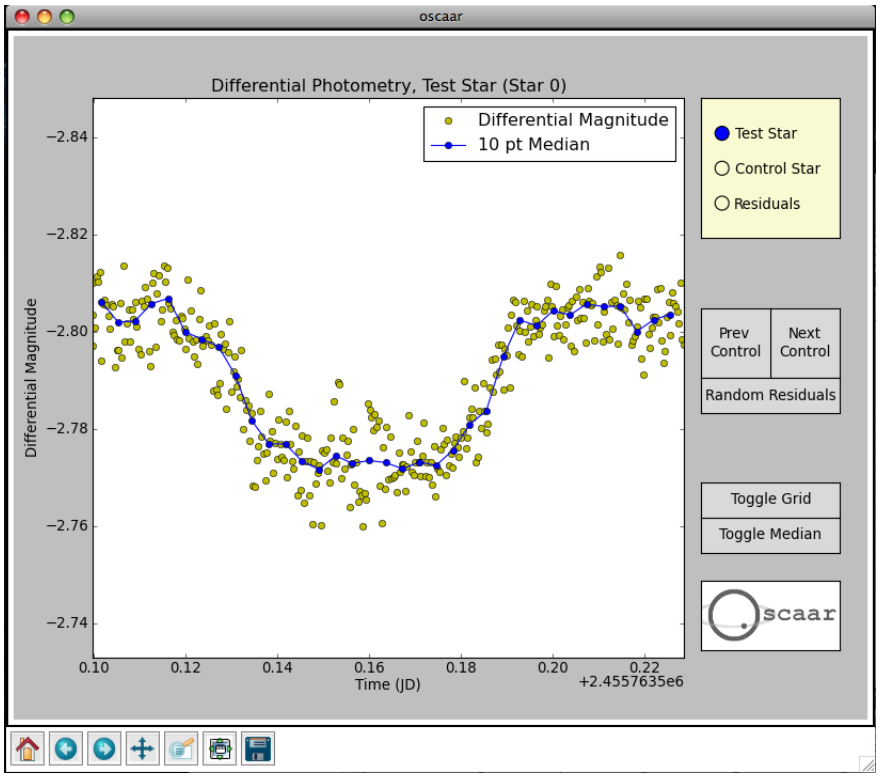
```
#circle(x,y,r)
```

3.4 Using the oscaar GUI

When `diffgui` is `on`, the built-in GUI for `oscaar` will open with your data. This can be explicitly triggered from the command line with

```
$ python differgui2.py
```

in `oscaar1.1.0`. The following interface will be displayed:

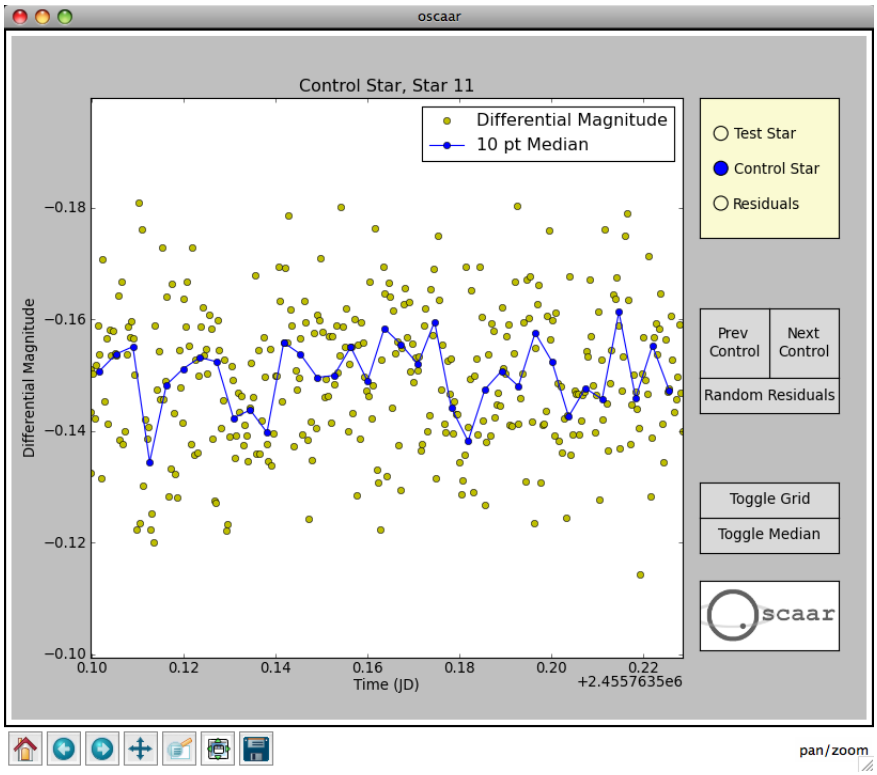


Screenshot 2: The starting view in `oscaar`'s GUI

This shows the light curve of your test star. The example in Screenshot 2 is a light curve of transit of exoplanet HD 189733b. The vertical axis corresponds to the differential magnitude of the star with some constant arbitrary vertical displacement. Note that the dimmer astronomical magnitudes correspond to larger numbers. The horizontal axis is time in Julian Date, as recorded in the FITS header of your test star.

This plot is interactive. You can click on the middle button at the bottom of the window (the axes icon) to select a panning tool. The magnifying glass icon is a rectangular zoom tool. Click the home icon to return to the initial (default) view. Click the floppy disk icon to save the image to a PNG.

You can check for intrinsic variability in your control stars by clicking the "Control Star" button in the yellow box in the upper right. You can go through all of the control stars by clicking the "Prev Control" and "Next Control" buttons.



Screenshot 3: Checking control stars in `oscaar`'s GUI

Even good control stars will show noise like this one. The scaling of the magnitude axis changes from star to star, so even though they may all look similar, the scale is almost certainly changing.

The "Residuals" plots may be a bit of a misnomer. They represent the difference between the differential magnitude measurements of two random control stars. This is mainly a tool for troubleshooting. If there are big, discontinuous jumps in these "Residual" plots or trends in the data apart from noise around zero, there is likely something wrong with the analysis of that control star. To check different pairs of stars, click "Random Residuals".

Grid lines and the median binning can be toggled with the "Toggle Grid" and "Toggle Median" buttons.

4 `oscaar` For Programmers

4.0.1 Navigating `oscaar` Output Files

`oscaar` outputs files into different directories within the `oscaar1.1.0` running directory.

Directory	Contents	Contents Format
filelists	Names of files that `oscaar` will need	List of paths
aper_out	Aperture photometry results	Inst. mag., error
track_out	Tracking algorithm results	x-pos., y-pos., sigma
diff_out	Differential photometric magnitude	Diff. mag., error
time_out	JD time extracted from test star headers	List of JD

Note: The differential magnitude error algorithm is still not well refined.

The `oscaar` running directory is intended to be copied for each new run you do to preserve the running parameters for your records. If you start a new run in an `oscaar` directory with these output directories already generated, `oscaar` will ask you before overwriting them.

4.0.2 `oscaar` API

You are welcome to write your own script that reads the different output files, and manipulates the data however you like. This will give you the power to script any analysis you can think of. The script that does the differential photometry in `oscaar1.1.0` is `differcalc2.py`. It uses object orientation to manage all of the different bits of data that are involved. The classes used in `differcalc2.py` are defined in `diffmodule3.py`, and you may find them useful to handle the myriad of data elements.

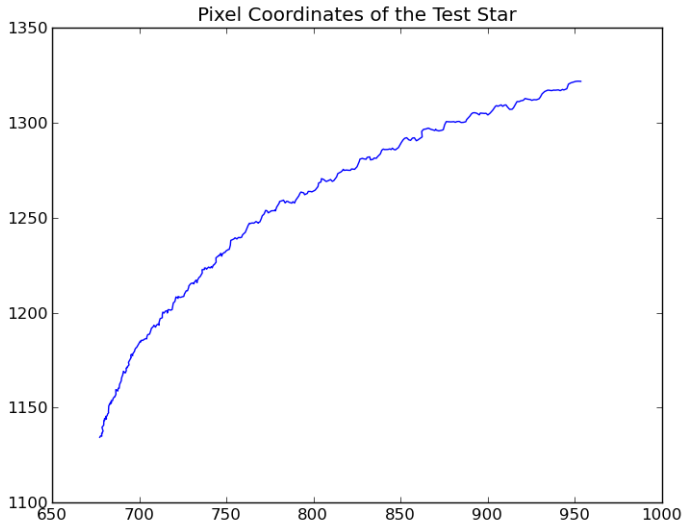
For example, the sample script below will plot the pixel coordinates of the test star throughout the run by objectively accessing the x and y positions of the star:

```
import matplotlib.pyplot as plt
import diffmodule3 as dm ## Classes for "fluxArray" and "starObj"

teststar = dm.starObj('001') ## Initialize a star object
x = teststar.trackx() ## Access the x and y positions
y = teststar.tracky()

plt.plot(x,y) ## Plot the star positions
plt.title('Pixel Coordinates of the Test Star')
plt.show()
```

which may produce a plot that looks something like this:



Screenshot 4: Making a quick script to see the star position on the detector throughout the run

Open up an interactive Python shell, import the module and do a `help(differmodule3)` to see all of the available object classes and methods.