# OSCAAR Documentation

***Release 2.0beta***

**Coders**

March 22, 2016

Contents

CHAPTER 1

# oscaar

## 1.1 oscaar Package

### 1.1.1 astrometry Package

#### trackSmooth Module

oscaar v2.0 Module for differential photometry Developed by Brett Morris, 2011-2013

oscaar.astrometry.trackSmooth.**quadraticFit**(*derivative*, *ext*)

Find an extremum in the data and use it and the points on either side, fit a quadratic function to the three points, and return the x-position of the apex of the best-fit parabola.

Called by oscaar.trackSmooth()

**Parameters** **derivative** : numpy.ndarray

The first derivative of the series of points, usually calculated by np.diff()

**ext** : string

Extremum to look find. May be either "max" or "min"

**Returns** **extremum** : float

The (non-integer) index where the extremum was found

oscaar.astrometry.trackSmooth.**trackSmooth**(*image*, *est_x*, *est_y*, *smoothingConst*, *plottingThings*, *preCropped=False*, *zoom=20.0*, *plots=False*)

Method for tracking stellar centroids.

**Parameters** **image** : numpy.ndarray

FITS image read in by PyFITS

**est_x** [float] Inital estimate for the x-centroid of the star

**est_y** [float] Inital estimate for the y-centroid of the star

**smoothingConstant** [float] Controls the degree to which the raw stellar intensity profile will be smoothed by a Gaussian filter (0 = no smoothing)

**preCropped** [bool] If preCropped=False, image is assumed to be a raw image, if preCropped=True, image is assumed to be only the portion of the image near the star

**zoom** [int or float] How many pixels in each direction away from the estimated centroid to consider when tracking the centroid. Be sure to choose a large enough zoom value the stellar centroid in the next exposure will fit within the zoom

**plots** [bool] If plots=True, display stellar intensity profile in two axes and the centroid solution

**Returns** **xCenter** : float

The best-fit x-centroid of the star

**yCenter** [float] The best-fit y-centroid of the star

**averageRadius** [float] Average radius of the SMOOTHED star in pixels

**errorFlag** [bool] Boolean corresponding to whether or not any error occured when running oscaar.trackSmooth(). If an error occured, the flag is True; otherwise False.

Core developer: Brett Morris Modifications by: Luuk Visser, 2-12-2013

### 1.1.2 c Package

#### `setup` Module

### 1.1.3 extras Package

#### Subpackages

##### eph Package

###### `EphGUI` Module

**`calculateEphemerides` Module** Ephemeris calculating tool that uses transit data from exoplanets.org and astrometric calculations by PyEphem to tell you what transits you'll be able to observe from your observatory in the near future.

Exoplanets.org citation: Wright et al.2011 http://arxiv.org/pdf/1012.5676v3.pdf

Core developer: Brett Morris

`oscaar.extras.eph.calculateEphemerides.`**`calculateEphemerides`**(*parFile*)

**INPUTS**

parFile – path to the parameter file

`oscaar.extras.eph.calculateEphemerides.`**`gd2jd`**(**date*)

gd2jd.py converts a UT Gregorian date to Julian date.

Usage: gd2jd.py (2009, 02, 25, 01, 59, 59)

To get the current Julian date: import time gd2jd(time.gmtime())

Hours, minutesutes and/or seconds can be omitted – if so, they are assumed to be zero.

Year and month are converted to type INT, but all others can be type FLOAT (standard practice would suggest only the final element of the date should be float)

oscaar.extras.eph.calculateEphemerides.**jd2gd**(*jd*, *returnString=False*)

Task to convert a list of julian dates to gregorian dates description at http://mathforum.org/library/drmath/view/51907.html Original algorithm in Jean Meeus, "Astronomical Formulae for Calculators"

2009-02-15 13:36 IJC: Converted to importable, callable function

Note from author: This script is buggy and reports Julian dates which are off by a day or two, depending on how far back you go. For example, 11 March 1609 converted to JD will be off by two days. 20th and 21st century seem to be fine, though.

Note from Brett Morris: This conversion routine matches up to the "Numerical Recipes" in C version from 2010-2100 CE, so I think we'll be ok for oscaar's purposes.

**ephemeris Module** Created on Feb 19, 2013

methods for calculating ephemerides

@author: bmmorris

**class** oscaar.extras.eph.ephemeris.**eph**(*startSem*, *endSem*)

#### Methods

**IDdoubleEclipses**(*axis*, *color='k'*, *ls='-'*)
Identify nights with two well separated transits

**IDdoubleTransits**(*axis*, *color='k'*, *ls='-'*)
Identify nights with two well separated transits

**IDdoubles**(*axis*, *color='k'*, *ls='-'*)

**addStar**(*name*, *Tc*, *pmTc*, *P*, *pmP*, *V*, *K*)

**getAllKmags**()

**getAllVmags**()

**getKmag**(*star*)

**getMidE**(*star*)

**getMidT**(*star*)

**getVmag**(*star*)

**midEclipse**(*Tc*, *pmTc*, *P*, *pmP*, *name*)

**midTransit**(*Tc*, *pmTc*, *P*, *pmP*, *name*)

**plotEclipses**(*axis*, *format*, *alphaSetting*, *showLabels=False*, *showMags=False*, *yOffset=0.0*)

**plotTransits**(*axis*, *format*, *alphaSetting*, *scaleSize=False*, *showLabels=False*, *showMags=False*, *yOffset=0.0*)

oscaar.extras.eph.ephemeris.**gd2jd**(**date*)

gd2jd.py converts a UT Gregorian date to Julian date.

Usage: gd2jd.py (2009, 02, 25, 01, 59, 59)

To get the current Julian date: import time gd2jd(time.gmtime())

Hours, minutesutes and/or seconds can be omitted – if so, they are assumed to be zero.

Year and month are converted to type INT, but all others can be type FLOAT (standard practice would suggest only the final element of the date should be float)

oscaar.extras.eph.ephemeris.**jd2gd**(*jd*, *returnString=False*)

Task to convert a list of julian dates to gregorian dates description at http://mathforum.org/library/drmath/view/51907.html Original algorithm in Jean Meeus, "Astronomical Formulae for Calculators"

2009-02-15 13:36 IJC: Converted to importable, callable function

Note from author: This script is buggy and reports Julian dates which are off by a day or two, depending on how far back you go. For example, 11 March 1609 converted to JD will be off by two days. 20th and 21st century seem to be fine, though.

**runEphemerisGenerator Module** This tool generates custom transit and/or eclipse ephemerides for your observatory and the dates you choose. You can also choose a limiting target star magnitude (V) so that you don't get a list flooded by targets that are too dim to observe at your with your equipment. Similarly, you can choose a lower limit on the transit depths, so that you'll generate a calendar of events that you'll be able to produce observe with decent signal/noise.

The exoplanet data are loaded from exoplanets.org, so please run this script while connected to the internet so that you can have the most up-to-date exoplanet data.

Core developer: Brett Morris (NASA GSFC)

#### knownSystemParameters Package

**getLatestParams Module**

oscaar.extras.knownSystemParameters.getLatestParams.**downloadAndPickle**()

oscaar.extras.knownSystemParameters.getLatestParams.**internet_connected**()

If internet connection is available, return True.

**returnSystemParams Module**

oscaar.extras.knownSystemParameters.returnSystemParams.**RpOverRs**(*planet*)

Ratio of planet radius to stellar radius, derived from transit depth since depth=(Rp/Rs)^2

oscaar.extras.knownSystemParameters.returnSystemParams.**aOverRs**(*planet*)

Returns semimajor axis over stellar radius (a/Rs)

oscaar.extras.knownSystemParameters.returnSystemParams.**depth**(*planet*)

Transit depth = (Rp/Rs)^2

oscaar.extras.knownSystemParameters.returnSystemParams.**eccentricity**(*planet*)

oscaar.extras.knownSystemParameters.returnSystemParams.**epoch**(*planet*)

Tc at mid-transit. Units: days

oscaar.extras.knownSystemParameters.returnSystemParams.**inclination**(*planet*)

oscaar.extras.knownSystemParameters.returnSystemParams.**period**(*planet*)

Units: days

oscaar.extras.knownSystemParameters.returnSystemParams.**transiterParams**(*planet*)

Return accepted values for the fitting routine

### 1.1.4 `__init__` Module

### 1.1.5 `IO` Module

OSCAAR v2.0 Module for differential photometry

Developed by Brett Morris, 2011-2013

`oscaar.IO.`**`cd`**(*a=None*)
Change to a different directory than the current one.

**Parameters** **a** : string

Location of the directory to change to.

#### Notes

If *a* is empty, this function will change to the parent directory.

`oscaar.IO.`**`cp`**(*a*, *b*)
Copy a file to another location.

**Parameters** **a** : string

Path of the file to be copied.

**b** : string

Location where the file will be copied to.

`oscaar.IO.`**`load`**(*inputPath*)
Load everything from a oscaar.dataBank() object in a python pickle using cPickle.

**Parameters** **inputPath** : string

File location of an oscaar.dataBank() object to save into a pickle.

**Returns** **data** : string

Path for the saved numpy-pickle.

`oscaar.IO.`**`parseRegionsFile`**(*regsPath*)
Parse a regions file for a set of data.

**Parameters** **regsPath** : string

Location of the regions file to be parsed.

**Returns** **init_x_list** : array

An array containing the x-values of the parsed file.

**init_y_list** : array

An array containing the y-values of the parsed file.

`oscaar.IO.`**`plottingSettings`**(*trackPlots*, *photPlots*, *statusBar=True*)
**Description :** Function for handling matplotlib figures across OSCAAR methods.

**Parameters** **trackPlots** : bool

Used to turn the astrometry plots on and off.

**photPlots** : bool

Used to turn the aperture photometry plots on and off.

**statusBar** : bool, optional

Used to turn the status bar on and off.

**Returns** **[fig, subplotsDimensions, photSubplotsOffset]** : [figure, int, int]

An array with 3 things. The first is the figure object from matplotlib that will be displayed while OSCAAR is running. The second is the integer value that designates the x and y dimensions of the subplots within the figure plot. The third is the the number correlating to the location of the aperture photometry plots, which depends on the values of trackPlots and photPlots.

**statusBarFig** : figure

A figure object from matplotlib showing the status bar for completion.

**statusBarAx** : figure.subplot

A subplot from a matplotlib figure object that represents what is drawn.

#### Notes

This list returned by plottingSettings() should be stored to a variable, and used as an argument in the phot() and trackSmooth() methods.

`oscaar.IO.`**`save`**(*data*, *outputPath*)

Save everything in oscaar.dataBank object <data> to a python pickle using cPickle.

**Parameters** **data** : string

File location of an oscaar.dataBank() object to save.

**outputPath** : string

Path to which the numpy-pickle will be saved.

### 1.1.6 `dataBank` Module

oscaar v2.0 Module for differential photometry Developed by Brett Morris, 2011-2013 & minor modifications by Luuk Visser

**class** `oscaar.dataBank.`**`dataBank`**(*initParFilePath=None*)

Methods for easily storing and accessing information from the entire differential photometry process with OSCAAR.

Core Developer: Brett Morris

#### Methods

**`calcChiSq`**()

Calculate the for the fluxes of each comparison star and the fluxes of the target star. This metric can be used to suggest which comparison stars have similar overall trends to the target star.

**`calcChiSq_multirad`**(*apertureRadiusIndex*)

Calculate the for the fluxes of each comparison star and the fluxes of the target star. This metric can be used to suggest which comparison stars have similar overall trends to the target star.

**calcMeanComparison**(*ccdGain=1*)
Take the regression-weighted mean of some of the comparison stars to produce one comparison star flux to compare to the target to produce a light curve.

The comparison stars used are those whose chi-squareds calculated by self.calcChiSq() are less than 2*sigma away from the other chi-squareds. This condition removes outliers.

**calcMeanComparison_multirad**(*ccdGain=1*)
Take the regression-weighted mean of some of the comparison stars to produce one comparison star flux to compare to the target to produce a light curve.

The comparison stars used are those whose are less than away from the other :math:‘$chi^2$‘s. This condition removes outlier comparison stars, which can be caused by intrinsic variability, tracking inaccuracies, or other effects.

**centroidInitialGuess**(*expNumber*, *star*)
Gets called for each exposure. If called on the first exposure, it will return the intial centroid guesses input by the DS9 regions file. If any other image and only one regions file has been submitted, it will return the previous centroid as the initial guess for subsequent exposures. If multiple regions files have been submitted, it will return the initial guesses in those regions files when the image path with index `expNumber` is equivalent to the path stored for that regions file's "Reference FITS image".

**Parameters** **expNumber** : int

The index of the exposure currently being analyzed. The image gets called by its index from the list of image paths returned by getPaths().

**star** : str

The key from `allStarsDict` that corresponds to the star for which you'd like a centroid initial guess.

**Returns** **est_x** : float

Estimated centroid position of the star `star` along the *x*-axis of pixels for exposure index `expNumber`

**est_y** : float

Estimated centroid position of the star `star` along the *y*-axis of pixels for exposure index `expNumber`

**computeLightCurve**(*meanComparisonStar*, *meanComparisonStarError*)
Divide the target star flux by the mean comparison star to yield a light curve, save the light curve into the dataBank object.

INPUTS: meanComparisonStar - The fluxes of the (one) mean comparison star

RETURNS: self.lightCurve - The target star divided by the mean comparison star, i.e., the light curve.

**computeLightCurve_multirad**(*meanComparisonStars*, *meanComparisonStarErrors*)
Divide the target star flux by the mean comparison star to yield a light curve, save the light curve into the *dataBank* object.

**Parameters** **meanComparisonStar** : list

The fluxes of the (one) mean comparison star

**Returns** self.lightCurves:

The fluxes of the target star divided by the fluxes of the mean comparison star, i.e., the light curve

self.lightCurveErrors:

The propagated errors on each relative flux in *self.lightCurves*

**czechETDstring**(*apertureRadiusIndex*)
Returns a string containing the tab delimited light curve data for submission to the *Czech Astronomical Society's Exoplanet Transit Database*, for submission here: http://var2.astro.cz/ETD/protocol.php

**Parameters** **apertureRadiusIndex** : int

Index of the aperture radius from which to use for the light curve fluxes and errors.

**getAllChiSq**()
Return :math:‘$chi^2$‘s for all stars

**getAllFlags**()
Return flags for all stars

**getDict**()
Return dictionary of all star data called ‘‘allStarsDict‘.

**getErrors**(*star*)
Return the errors for one star, where the star parameter is the key for the star of interest.

**getErrors_multirad**(*star*, *apertureRadiusIndex*)
Return the errors for one star, where the star parameter is the key for the star of interest.

**getFlag**(*star*)
Return the flag for the star with key *star*

**getFluxes**(*star*)
Return list of fluxes for the star with key `star`

**Parameters** **star** : str

Key for the star from the `allStarsDict` dictionary

**Returns** **fluxes** : list

List of fluxes for each aperture radius

**getFluxes_multirad**(*star*, *apertureRadiusIndex*)
Return the fluxes for one star, where the star parameter is the key for the star of interest.

**getKeys**()
Return the keys for all of the stars

**getMeanDarkFrame**()

**getPaths**()
Return the paths to the raw images to be used

**getPhotonNoise**()
Calculate photon noise using the lightCurve and the meanComparisonStar

RETURNS: self.photonNoise - The estimated photon noise limit

**getScaledErrors**(*star*)
Return the scaled fluxes for one star, where the star parameter is the key for the star of interest.

**getScaledErrors_multirad**(*star*, *apertureRadiusIndex*)
Return the scaled errors for star and one aperture, where the star parameter is the key for the star of interest.

**getScaledFluxes**(*star*)
Return the scaled fluxes for one star, where the star parameter is the key for the star of interest.

**getScaledFluxes_multirad**(*star*, *apertureRadiusIndex*)
Return the scaled fluxes for star and one aperture, where the star parameter is the key for the star of interest.

**getTimes**()
Return all times collected with dataBank.storeTime()

**outOfTransit**()
Boolean array where *True* are the times in *getTimes()* that are before ingress or after egress.

**Returns** List of bools

**parseInit**(*initParFilePath=None*)
Parses *init.par*, a plain text file that contains all of the running parameters that control the *differentialPhotometry.py* script. *init.par* is written by the OSCAAR GUI or can be edited directly by the user.

**Parameters** **initParFilePath** : str

Optional full path to the init.par file to use for the data

**parseRawRegionsList**(*rawRegionsList*)
Split up the *rawRegionsList*, which should be in the format:

<first regions file>,<reference FITS file for the first regs file>;<second> regions file>, <reference FITS file for the first regs file>;....

into a list of regions files and a list of FITS reference files.

**parseRegionsFile**(*regPath*)
Parses the regions files (.REG) created by DS9. These files are written in plain text, where each circuluar region's centroid and radius are logged in the form "circle(*x-centroid*,'y-centroid','radius')". This method uses regular expressions to parse out the centroids.

**Parameters** **regPath** : string

Path to the regions file to read

**Returns** **init_x_list** : list

Initial estimates for the x-centroids

**init_y_list** : list

Initial estimates for the y-centroids

**plot**(*pointsPerBin=10*)
Produce a plot of the light curve, show it. Over-plot 10-point median binning of the light curve.

**Parameters** **pointsPerBin** : int, optional (default=10)

Integer number of points to accumulate per bin.

**plotCentroidsTrace**(*pointsPerBin=10*)
Plot all centroid positions for a particular aperture radius, for each comparison star. The plot will be in (*x*,'y') coordinates to visualize the physical image drift (this is not a plot as a function of time).

**Parameters** **pointsPerBin** : int, optional (default=10)

Integer number of points to accumulate per bin.

**apertureRadiusIndex** : int, optional (default=0)

Index of the aperture radius list corresponding to the aperture radius from which to produce the plot.

**plotComparisonWeightings**(*apertureRadiusIndex=0*)
Plot histograms visualizing the relative weightings of the comparison stars used to produce the "mean comparison star", from which the light curve is calculated.

**Parameters** **apertureRadiusIndex** : int, optional (default=0)

Index of the aperture radius list corresponding to the aperture radius from which to produce the plot.

**plotLightCurve**(*pointsPerBin=10*, *apertureRadiusIndex=0*)
Produce a plot of the light curve, show it. Over-plot 10-point median binning of the light curve.

**Parameters** **pointsPerBin** : int, optional (default=10)

Integer number of points to accumulate per bin.

**apertureRadiusIndex** : int, optional (default=0)

Index of the aperture radius list corresponding to the aperture radius from which to produce the plot.

**plotLightCurve_multirad**(*pointsPerBin=10*)

**plotRawFluxes**(*apertureRadiusIndex=0*, *pointsPerBin=10*)
Plot all raw flux time series for a particular aperture radius, for each comparison star.

**Parameters** **pointsPerBin** : int, optional (default=10)

Integer number of points to accumulate per bin.

**apertureRadiusIndex** : int, optional (default=0)

Index of the aperture radius list corresponding to the aperture radius from which to produce the plot.

**plotScaledFluxes**(*apertureRadiusIndex=0*, *pointsPerBin=10*)
Plot all scaled flux time series for a particular aperture radius, for each comparison star.

**Parameters** **pointsPerBin** : int, optional (default=10)

Integer number of points to accumulate per bin.

**apertureRadiusIndex** : int, optional (default=0)

Index of the aperture radius list corresponding to the aperture radius from which to produce the plot.

**scaleFluxes**()
When all fluxes have been collected, run this to re-scale the fluxes of each comparison star to the flux of the target star. Do the same transformation on the errors.

**scaleFluxes_multirad**()
When all fluxes have been collected, run this to re-scale the fluxes of each comparison star to the flux of the target star. Do the same transformation on the errors.

**setFlag**(*star*, *setting*)
Set flag for star with key <star> to <setting> where setting is a Boolean

**storeCentroid**(*star*, *exposureNumber*, *xCentroid*, *yCentroid*)
Store the centroid data collected by *trackSmooth*

**Parameters** **star** : string

Key for the star for which the centroid has been measured

**exposureNumber** : int

Index of exposure being considered

**xCentroid** : float

*x*-centroid of the star

**yCentroid** : float

*y*-centroid of the star

**storeFlux**(*star*, *exposureNumber*, *rawFlux*, *rawError*)

Store the flux and error data collected by *phot*

**Parameters** **star** : string

Key for the star from the `allStarsDict` dictionary

**exposureNumber** : int

Index of exposure being considered

**rawFlux** : float

flux measured, to be stored

**rawError** : float

flux uncertainty measured, to be stored

**storeFluxes**(*star*, *exposureNumber*, *rawFluxes*, *rawErrors*)

Store the flux and error data collected by oscaar.phot()

**Parameters** **star** : str

Key for the star from the *allStarsDict* dictionary

**exposureNumber** : int

Index of exposure being considered

**rawFluxes** : list of floats

flux measured, to be stored

**rawErrors** : list of floats

photon noise measured, to be stored

**storeTime**(*expNumber*)

Store the time in JD from the FITS header. Parameters

**exposureNumber** [string] Index of exposure being considered

**uncertaintyString**()

**Returns** **savestring** : string

A string formatted for human-readable results from the MCMC process, with the best-fit parameters and the uncertainties

**updateMCMC**(*bestp*, *allparams*, *acceptanceRate*, *dataBankPath*, *uncertainties*)

Assigns variables within the dataBank object for the results of an MCMC run.

**Parameters** **bestp** : list

Best-fit parameters from the MCMC run. The list elements correspond to [<ratio of planetary radius to stellar radius>,<ratio of semi-major axis to stellar radius>,<inclination>,<mid-transit time>].

**allparams** : 2D matrix

This matrix represents the many "states", "trails" or "links in the chain" that are accepted and saved throughout the Metropolis-Hastings process in the MCMC scripts. From allparams we can calculate the uncertainties on each best-fit parameter.

**acceptanceRate** : float

The final acceptance rate achieved by the chain; the ratio of the number of accepted states and the number of states attempted

**dataBankPath** : string

Path to the dataBank object pickle (aka "OSCAAR pkl") to update

**uncertainties** : list of lists

uncertainties on each of the best-fit parameters in *bestp*

### 1.1.7 `differentialPhotometry` Module

### 1.1.8 `fitting` Module

`oscaar.fitting.`**`fitLinearTrend`**(*xVector*, *yVector*)

Fit a line to the set {xVectorCropped,yVectorCropped}, then remove that linear trend from the full set {xVector,yVector}

`oscaar.fitting.`**`get_uncertainties`**(*param*, *bestFitParameter*)

Find the uncertainties from a MCMC parameter chain.

**Parameters** **param** : list

parameter chain from the completed MCMC algorithm

**bestFitParam** : float

the best-fit (chi-squared) minimizing value for the parameter chain

**Returns** **[plus,minus]** : list of floats

the upper and lower 1-sigma uncertainties on the best fit parameter *bestFitParameter*

`oscaar.fitting.`**`linearFunc`**(*xVector*, *params*)

`oscaar.fitting.`**`mcmc`**(*t*, *flux*, *sigma*, *initParams*, *func*, *Nsteps*, *beta*, *saveInterval*, *verbose=False*, *loadingbar=True*)

Markov Chain Monte Carlo routine for fitting. Takes a set of fluxes *flux* measured at times *t* with uncertainties *sigma*. Input fitting function *func* is fed initial parameters *initParams* and iterated through the chains a total of *Nsteps* times, randomly sampled from normal distributions with widths *beta*, and every *saveInterval*-th state in the chain is saved for later analysis.

**Parameters** **t** : list

times

**flux** : list

fluxes

**sigma** : list

uncertainties in fluxes

**initParams** : list

initial parameter estimates, *x_0* in Ford 2005

**func** : function

fitting function

**Nsteps** : int

number of iterations

**beta** : list

widths of normal distribution to randomly sample for each parameter

**saveInterval** : int

number of steps between "saving" the accepted parameter in the chain. Must satisfy `Nsteps % saveInterval ==0`.

**Returns** **bestp** : list

parameters at minimum chi^2

**x_0toN** : array

trace of each parameter at each save step

acceptanceRate: float

the final acceptance rate of the chain

`oscaar.fitting.`**`mcmc_iterate`**(*t*, *flux*, *sigma*, *initParams*, *func*, *Nsteps*, *beta*, *saveInterval*, *verbose=False*)

MCMC routine specifically for optimizing the beta parameters with the optimizeBeta() function.

**Parameters** **t** : list

time

**flux** [list] fluxes

**sigma** [list] uncertainties in fluxes

**initParams** [list ] initial parameter estimates, *x_0* in Ford 2005

**func** [function ] fitting function

**Nsteps** [int] number of steps to try in the chains

**beta** [list] widths of normal distribution to randomly sample for each parameter

**Returns** **acceptanceRateArray** : list

Acceptance rates for each beta_mu

**class** `oscaar.fitting.`**`mcmcfit`**(*dataBankPath*, *initParams*, *initBeta*, *Nsteps*, *saveInterval*, *idealAcceptanceRate*, *burnFraction*)

#### Methods

**`plot`**(*num=0*)

**`run`**(*updatepkl=False*, *apertureRadiusIndex=0*)

Run the MCMC algorithms:

**Parameters** **updatepkl** : bool, optional

update the OSCAAR save pkl file from which the data had been loaded with the MCMC best fit parameters, parameter chains, and acceptance rate.

**apertureRadiusIndex** : int, optional

Integer index of the aperture radius for which you'd like to compute the MCMC fit, from the aperture radius range list

`oscaar.fitting.`**`optimizeBeta`**(*t*, *flux*, *sigma*, *initParams*, *func*, *beta*, *idealAcceptanceRate*, *plot=True*)

The *beta* input parameters for the MCMC function determine the acceptance rate of the Metropolis-Hastings algorithm. According to Ford 2005, the ideal acceptance rate is ~0.25 - ~0.44. This routine is designed to take an initial guess for each of the beta parameters and tweak them until they produce good acceptance rates for each parameter. This is achieved by randomly perturbing each initial parameter with the small perturbation by randomly sampling a normal distribution with a width given by the initial beta vector *beta*. optimizeBeta() then tries running an MCMC chain briefly to find the acceptance rate for that beta parameter. If the acceptance rates are two high, for example, then the beta is too low, and optimizeBeta() will increase beta. This process continues until the beta vector produces acceptance rates within 10% of the *idealAcceptanceRate*, which according to Ford (2005) should be between 0.25-0.44.

**Parameters** **t** : list

time

**flux** : list

fluxes

**sigma** : list

uncertainties in fluxes

**initParams** : list

initial parameter estimates, *x_0* in Ford 2005

**func** : function

fitting function

**beta** : list

widths of normal distribution to randomly sample for each parameter

**idealAcceptanceRate** : float

desired acceptance rate to be produced by the optimized *beta*

**Returns** **beta** : list

the beta vector optimized so that running a MCMC chain should produce acceptance rates near *idealAcceptanceRate* (vector)

`oscaar.fitting.`**`updatePKL`**(*bestp*, *allparams*, *acceptanceRate*, *pklPath*, *uncertainties*)

Load an OSCAAR pkl, add the MCMC parameters to the file, save it again.

**Parameters** **bestp** : list

best-fit values for each parameter

**allparams** : array

2D array where each saved state of the chains is stored along one dimension, for each fitting parameter (along the other)

**acceptanceRate** : float

the final acceptance rate acheived in the chain

**pklPath** : str

path to the pkl to overwrite.

### 1.1.9 `mathMethods` Module

oscaar v2.0 Module for differential photometry

Developed by Brett Morris, 2011-2013

`oscaar.mathMethods.`**`chiSquared`**(*vector1*, *vector2*)
Return (chi-squared) of two vectors

`oscaar.mathMethods.`**`medianBin`**(*time*, *flux*, *medianWidth*)
Produce median binning of a flux vector

**Parameters** **time** : list or numpy.ndarray

List of times in time series

**flux** : list or numpy.ndarray

List of fluxes, one for each time in *time* vector

**medianWidth** : int

Width of each bin in units of data points

**Returns** **[binnedTime, binnedFlux, binnedStd]** : [list, list, list] or [numpy.ndarray, numpy.ndarray, numpy.ndarray]

The times, fluxes and uncertainties on each binned point, where *binnedTime* is the time for each bin, *binnedFlux* is the median flux in each bin, and *binnedStd* is the standard deviation of the points within each bin

`oscaar.mathMethods.`**`regressionScale`**(*comparisonFlux*, *targetFlux*, *time*, *ingress*, *egress*, *returncoeffs=False*)
Use a least-squares regression to stretch and offset a comparison star fluxes to scale them to the relative intensity of the target star. Only do this regression considering the out-of-transit portions of the light curve.

**Parameters** **comparisonFlux** : numpy.ndarray

Flux of a comparison star

**targetFlux** : numpy.ndarray

Flux of the target star

**time** : numpy.ndarray

List of times for each flux measurement in JD

**ingress** : float

Time of ingress (JD, assuming time list is in JD)

**egress** : float

Time of egress (JD, assuming time list is in JD)

**Returns** **scaledVector** : numpy.ndarray

Rescaled version of the comparisonFlux vector using the above described process

oscaar.mathMethods.**ut2jd**(*ut*)
Convert times from Universal Time (UT) to Julian Date (JD)

**Parameters** **ut** : string

Time in Universal Time (UT)

**Returns** **jd** : float

Julian Date (JD)

oscaar.mathMethods.**ut2jdSplitAtT**(*ut*)
Convert times from Universal Time (UT) to Julian Date (JD), splitting the date and time at the "T"

**Parameters** **ut** : string

Time in Universal Time (UT)

**Returns** **jd** : float

Julian Date (JD)

### 1.1.10 oscaarGUI Module

### 1.1.11 other Module

oscaar.other.**gd2jd**(**date*)
gd2jd.py converts a UT Gregorian date to Julian date.

Usage: gd2jd.py (2009, 02, 25, 01, 59, 59)

To get the current Julian date: import time gd2jd(time.gmtime())

Hours, minutesutes and/or seconds can be omitted – if so, they are assumed to be zero.

Year and month are converted to type INT, but all others can be type FLOAT (standard practice would suggest only the final element of the date should be float)

oscaar.other.**jd2gd**(*jd*, *returnString=False*)
Task to convert a list of julian dates to gregorian dates description at http://mathforum.org/library/drmath/view/51907.html Original algorithm in Jean Meeus, "Astronomical Formulae for Calculators"

2009-02-15 13:36 IJC: Converted to importable, callable function

Note from author: This script is buggy and reports Julian dates which are off by a day or two, depending on how far back you go. For example, 11 March 1609 converted to JD will be off by two days. 20th and 21st century seem to be fine, though.

Note from Brett Morris: This conversion routine matches up to the "Numerical Recipes" in C version from 2010-2100 CE, so I think we'll be ok for oscaar's purposes.

oscaar.other.**overWriteCheck**(*filename*, *checkfiles*, *varcheck*)
Checks to see if a particular file should be overwritten based on whether varcheck is on or off

### 1.1.12 photometry Module

oscaar v2.0 Module for differential photometry Developed by Brett Morris, 2011-2013

oscaar.photometry.**multirad**(*image*, *xCentroid*, *yCentroid*, *apertureRadii*, *plottingThings*, *annulusOuterRadiusFactor=2.8*, *annulusInnerRadiusFactor=1.4*, *ccdGain=1*, *plots=False*)

Method for aperture photometry.

**Parameters** **image** : numpy.ndarray

FITS image opened with PyFITS

**xCentroid** : float

Stellar centroid along the x-axis (determined by trackSmooth or equivalent)

**yCentroid** : float

Stellar centroid along the y-axis (determined by trackSmooth or equivalent)

**apertureRadii** : list

List of aperture radii (floats) to feed to phot().

**annulusInnerRadiusFactor** : float

Measure the background for sky background subtraction fron an annulus from a factor of *annulusInnerRadiusFactor* bigger than the *apertureRadius* to one a factor *annulusOuterRadiusFactor* bigger.

**annulusOuterRadiusFactor** : float

Measure the background for sky background subtraction fron an annulus a factor of *annulusInnerRadiusFactor* bigger than the *apertureRadius* to one a factor *annulusOuterRadiusFactor* bigger.

**ccdGain** : float

Gain of your detector, used to calculate the photon noise

**plots** : bool

If 'plots'=True, display plots showing the aperture radius and annulus radii overplotted on the image of the star

**Returns** **rawFlux** : float

The background-subtracted flux measured within the aperture

**rawError** : float

The photon noise (limiting statistical) Poisson uncertainty on the measurement of *rawFlux*

**errorFlag** : bool

Boolean corresponding to whether or not any error occured when running oscaar.phot(). If an error occured, the flag is True; otherwise False.

Core developer: Brett Morris (NASA-GSFC)

oscaar.photometry.**phot**(*image*, *xCentroid*, *yCentroid*, *apertureRadius*, *plottingThings*, *annulusOuterRadiusFactor=2.8*, *annulusInnerRadiusFactor=1.4*, *ccdGain=1*, *plots=False*)

Method for aperture photometry.

**Parameters** **image** : numpy.ndarray

FITS image opened with PyFITS

**xCentroid** : float

Stellar centroid along the x-axis (determined by trackSmooth or equivalent)

**yCentroid** : float

Stellar centroid along the y-axis (determined by trackSmooth or equivalent)

**apertureRadius** : float

Radius in pixels from centroid to use for source aperture

**annulusInnerRadiusFactor** : float

Measure the background for sky background subtraction fron an annulus from a factor of *annulusInnerRadiusFactor* bigger than the *apertureRadius* to one a factor *annulusOuterRadiusFactor* bigger.

**annulusOuterRadiusFactor** : float

Measure the background for sky background subtraction fron an annulus a factor of *annulusInnerRadiusFactor* bigger than the *apertureRadius* to one a factor *annulusOuterRadiusFactor* bigger.

**ccdGain** : float

Gain of your detector, used to calculate the photon noise

**plots** : bool

If 'plots'=True, display plots showing the aperture radius and annulus radii overplotted on the image of the star

**Returns** **rawFlux** : float

The background-subtracted flux measured within the aperture

**rawError** : float

The photon noise (limiting statistical) Poisson uncertainty on the measurement of *rawFlux*

**errorFlag** : bool

Boolean corresponding to whether or not any error occured when running oscaar.phot(). If an error occured, the flag is True; otherwise False.

Core developer: Brett Morris (NASA-GSFC)

### 1.1.13 `systematics` Module

`oscaar.systematics.`**`meanDarkFrame`**(*darksPath*)

Returns the mean dark frame calculated from each dark frame in *darksPath*. If there is only one file present in *darksPath*, use the dimensions of that image to produce a dummy dark frame.

**Parameters** **darksPath** : list of strings

Paths to the dark frames

**Returns** The mean of the dark frames in *darksPath*

`oscaar.systematics.`**`standardFlatMaker`**(*flatImagesPath*, *flatDarkImagesPath*, *masterFlatSavePath*, *plots=False*)

Make a master flat by taking a mean of a group of flat fields

**Parameters** **flatImagesPath** : string

Path to the flat field exposures

**flatDarkImagesPath** : string

Path to the flat field darks

**masterFlatSavePath** : string

Where to save the master flat that is created

**plots** : bool

Plot the master flat on completion when plots=True

oscaar.systematics.**twilightFlatMaker**(*flatImagesPath*, *flatDarkImagesPath*, *masterFlatSavePath*, *plots=False*)

Make a master flat using a series of images taken at twilight by fitting the individual pixel intensities over time using least-squares and use the intercept as the normalizing factor in the master flat.

**Parameters** **flatImagesPath** : string

Path to the flat field exposures

**flatDarkImagesPath** : string

Path to the flat field darks

**masterFlatSavePath** : string

Where to save the master flat that is created

**plots** : bool

Plot the master flat on completion when plots=True

### 1.1.14 timeConversions Module

oscaar.timeConversions.**dateobs2jd**(*ut*)

Convert times from Universal Time (UT) to Julian Date (JD), splitting the date and time at the "T".

**Parameters** **ut** : string

Time in Universial Time (UT), in the format: "<YYYY:MM:DD>T<HH:MM:SS>"

**Returns** **jd** : float

Julian Date (JD)

oscaar.timeConversions.**findKeyword**(*fitsFile*)

**Parameters** **fitsfile** : string

Path to a FITS file

**Returns** **(useKeyword, allKeys, conversionFunction)** : tuple

where - *useKeyword* is the FITS header keyword that should be used to find the time of the exposure, - *allKeys* is the list of all header keywords in the first exposure - *conversionFunction* is a function that will convert the time value stored in the keyword denoted by *useKeyword* to Julian Date

oscaar.timeConversions.**jd2jd**(*jd*)

oscaar.timeConversions.**mjd2jd**(*mjd*)

Converts Modified Julian Date to Julian Date. Definition of Modified Julian Date (MJD): MJD = JD - 2400000.5

**Parameters** **mjd** : float

The Modified Julian Date

**Returns** **jd** : float

, the corresponding ordinary Julian Date

### 1.1.15 `transitModel` Module

transitModel.py defines the function occultquad(), which loads the C library containing the function of the same name so that analytical transit light curves can be produced in python by passing pythonic arguments to the C code.

`oscaar.transitModel.`**`ellipe`**(*k*)
Computes polynomial approximation for the complete elliptic integral of the second kind (Hasting's approximation)

`oscaar.transitModel.`**`ellipk`**(*k*)
Computes polynomial approximation for the complete elliptic integral of the first kind (Hasting's approximation):

`oscaar.transitModel.`**`ellippi`**(*n*, *k*)
Computes the complete elliptical integral of the third kind using the algorithm of Bulirsch (1965)

`oscaar.transitModel.`**`occultquad`**(*t*, *modelParams*)

Calculates the analytical transit light curve for a planet occulting a star, according to the formalism of Mandel & Agol (2002) *[R4]*.

**Parameters** **t** : list or numpy.ndarray

List of the times sampled in Julian Date

**modelParams** [list] List of the planetary system parameters, in the following order: - : Ratio of the radius of the planet to the radius of the star - : Ratio of the semi-major axis to the radius of the star - : Orbital period - : Limb-darkening coefficient, linear - : Limb-darkening coefficient, quadratic - : Eccentricity - *longPericenter*: Longitude of pericenter - : Mid-transit time (JD)

**Returns** **F** : numpy.ndarray

Relative fluxes at each time of the time vector *t*

for Planetary Transit Searches".

The Astrophysical Journal, Volume 580, Issue 2, pp. L171-L175. 2002.

- genindex
- modindex
- search

[R1] Eric Ford. “Quantifying the Uncertainty in the Orbits of Extrasolar Planets.” The Astronomical Journal, Volume 129, Issue 3, pp. 1706-1717. 2005.

[R2] Eric Ford. “Quantifying the Uncertainty in the Orbits of Extrasolar Planets.” The Astronomical Journal, Volume 129, Issue 3, pp. 1706-1717. 2005.

[R3] Eric Ford. “Quantifying the Uncertainty in the Orbits of Extrasolar Planets.” The Astronomical Journal, Volume 129, Issue 3, pp. 1706-1717. 2005.

[R4] Mandel & Agol. “Analytic Light Curvesfrom glob import glob

## O

## A



## C



## D



## E



## F



## G

## I



## J



## L



## M



## O



## P



## Q



## R

## S



## T



## U