exoctk Documentation

The ExoCTK Group

Sep 18, 2020

CONTENTS

ExoCTK is an open-source, modular data analysis package focused primarily on atmospheric characterization of exoplanets.

The subpackages currently included are:

Contamination and Visibility Calculators Groups and Intrgrations Calculator Transit Light-Curve Fitting Tools Limb-Darkening Calculator Phase Constraint Calculator Atmospheric Forward Modeling - Currently only available through the website.

All source code can be found on GitHub.

There is also a website where all current tools are available through interactive applications.

Part I

User Documentation

Contamination and Visibility Calculator

The Contamination Overlap tool is made up of two calculators: the visibility calculator, and the contamination calculator. The visibility calculator will determine at what position angles (PAs) your target is visible and when. The output results come in the form of interactive Bokeh plots, but users also have the option of downloading their data in an ascii file format. The contamination calculator will determine the percentage of spectral contamination that lands on your target at every PA. The visibility calculator is currently available for all JWST instruments, and the contamination calculator will be released for NIRISS (Mode: Single Object Slitless Spectroscopy), NIRCam (Mode: Grism Time Series), and MIRI (Mode: Low-Resolution Spectroscopy).

CHAPTER

ONE

EXOCTK.CONTAM_VISIBILITY PACKAGE

1.1 Submodules

1.2 exoctk.contam_visibility.astro_funcx module

exoctk.contam_visibility.astro_funcx.**JWST_same_ori**(*tgt0_c1*, *tgt0_c2*, *p0*, *tgt_c1*, *tgt_c2*)
Calculates normal orientation of second target, given first target's orientation is normal. This is in Ecliptic coordinates!

Parameters

tgt0_c1: float
The RA of the first target.

tgt0_c2: float
The Dec of the first target.

p0: float
The origin.

tgt1_c1: float
The RA of the second target.

tgt1_c2: float
The Dec of the second target.

Returns

float
The normal orientation.

exoctk.contam_visibility.astro_funcx.**delta_pa_no_roll**(*pos1_c1*, *pos1_c2*, *pos2_c1*, *pos2_c2*)
Calculates the change in position angle between two positions with no roll about V1

Parameters

pos1_c1: float
The RA of the first position.

pos1_c2: float
The Dec of the first position.

pos2_c1: float
The RA of the second position.

pos2_c2: float
The Dec of the second position.

Returns

float
The change in position angle.

exoctk.contam_visibility.astro_funcx.**dist**(*obj1_c1*, *obj1_c2*, *obj2_c1*, *obj2_c2*)
Calculates the angular distance betrween two objects, positions specified in spherical coordinates

Parameters

obj1_c1: float
The RA of the first object.

obj1_c2: float
The Dec of the first object.

obj2_c1: float
The RA of the second object.

obj2_c2: float
The Dec of the second object.

Returns

float
The distance between the objects.

exoctk.contam_visibility.astro_funcx.**pa**(*tgt_c1*, *tgt_c2*, *obj_c1*, *obj_c2*)
Calculates position angle of object at tgt position.

Parameters

tgt_c1: float
The RA of the target.

tgt_c2: float
The Dec of the target.

obj_c1: float
The RA of the reference.

obj_c2: float
The Dec of the reference.

Returns

float
The position angle.

exoctk.contam_visibility.astro_funcx.**unit_limit**(*x*)
Forces value to be in [-1, 1]

Parameters

x: float, int
The value to adjust.

1.3 exoctk.contam_visibility.ephemeris_old2x module

class exoctk.contam_visibility.ephemeris_old2x.**Ephemeris**(*ephem_file*, *cnvrt=False*)
Bases: object

A class for the ephemeris of an observation.

Ephemeris constructor

Parameters

ephem_file: str
The path to the ephemeris file.

cnvrt: bool, optional
Converts into Ecliptic frame.

OP_window(*adate*, *ngc_1*, *ngc_2*, *pa*, *mdelta*, *pdelta*)
Attitude at adate must be valid, else returns (0, 0). If valid, returns (adate-mdelta, adate+pdelta) or the constraint window, which ever is smaller.

Parameters

date: float
The date of the observation.

ngc_1: flaot
The RA of the reference in radians.

ngc_2: float
The Dec of the reference in radians.

pa: float
The position angle.

mdelta: float
Delta time on low end.

pdelta: float
Delta time on high end.

Returns

tuple
The OP window.

Vsun_pos(*adate*)
The vector of the sun at the given date

Parameters

adate: datetime
The date of the observation

Returns

Vector
The position of the sun as a Vector object

`bisect_by_FOR`(*in_date*, *out_date*, *ngc_1*, *ngc_2*)
Find the midpoint in time between in and out of FOR, assumes only one "root" in interval

Parameters

in_date: float
The in date of the observation.

out_date: float
The out date of the observation.

ngc_1: flaot
The RA of the reference in radians.

ngc_2: float
The Dec of the reference in radians.

Returns

float
The midpoint in time.

`bisect_by_attitude`(*in_date*, *out_date*, *ngc_1*, *ngc_2*, *pa*)
Find the midpoint in time between in and out of FOR, assumes only one "root" in interval

Parameters

in_date: float
The in date of the observation.

out_date: float
The out date of the observation.

ngc_1: flaot
The RA of the reference in radians.

ngc_2: float
The Dec of the reference in radians.

pa: float
The position angle.

Returns

float
The midpoint in time.

`in_FOR`(*date*, *ngc_1*, *ngc_2*)
Test if in the FOR

Parameters

date: float
The date of the observation.

ngc_1: flaot
The RA of the reference in radians.

ngc_2: float
The Dec of the reference in radians.

Returns

bool
Is it in the FOR.

`is_valid`(*date*, *ngc_1*, *ngc_2*, *V3pa*)
Indicates whether an attitude is valid at a given date.

Parameters

date: float
The date of the observation.

ngc_1: flaot
The RA of the reference in radians.

ngc_2: float
The Dec of the reference in radians.

V3pa: float
The V3 position of the telescope.

Returns

bool
Is it a valid PA.

`long_term_attitude`(*date*)
Defines a long-term safe attitude as of a given date.

Parameters

date: float
The date of computation, as an mjd.

Returns

Attitude
The Attitude object at the given date.

`normal_pa`(*adate*, *tgt_c1*, *tgt_c2*)
Calculate the V3 position

Parameters

adate: datetime.datetime object
The date of the observation.

tgt_c1: float
The RA in radians.

tgt_c2: float
The Dec in radians.

Returns

float
The V3 position.

pos(*adate*)
Computes the position of the telescope at a given date using the grid of positions of the ephemeris as a starting point and applying a linear interpolation between the ephemeris grid points

Parameters

adate: datetime.datetime object
The date of the observation.

Returns

qx.Vector object
The position of the telescope as a Vector object

report_ephemeris(*limit=100000*, *pathname=None*)
Prints a formatted report of the ephemeris.

Parameters

limit: int, optional
The number of records to report.

pathname: str, optional
The path to a file to hold the report.

sun_pos(*adate*)
The coordinates of the sun at the given date

Parameters

adate: datetime.datetime object
The date of the observation.

Returns

tuple
The coordinates of the sun.

exoctk.contam_visibility.ephemeris_old2x.**unit_limit**(*x*)
Forces value to be in [-1, 1].

Parameters

x: float, int
The value to adjust.

Returns

float
The adjusted value.

1.4 exoctk.contam_visibility.f_visibilityPeriods module

Series of functions to compute the visibility periods for a given (RA,DEC) with in some cases the possibility to select a PA value.

Functions derived from the code of Wayne Kinzel provided by Jeff Valenti Extract from the e-mail of Wayne Kinzel: As before, the code is not officially tested, nor is it an official STScI product. Users should be warned that the apparent position of the Sun changes ~+/-0.2 degrees epending upon where JWST is in its orbit. So do not rely strongly on these results if the target is within ~0.2 degrees of |ecliptic latitude| 45 degrees or 85 degrees. For example if a target is at 84.9 degrees latitude and the tool says it is CVZ, it may not be with the operational orbit.

exoctk.contam_visibility.f_visibilityPeriods.**f_computeDurationOfVisibilityPeriodWithPA**(*ephemeris*, *mjdmin*, *mjdmax*, *ra*, *dec*, *pa*, *mjdc*)

Computes the duration of a specific visibility period associated to a given (RA,DEC), a given PA and given date

flag = 0 visibility period fully in the search interval flag = -1 start of the visibility period truncated by

the start of the search interval

flag = -2 end of the visibility period truncated by
the end of the search interval

flag = +1 the search interval is fully included in
the visibility period

Parameters

ephemeris: Ephemeris
The input ephemeris object.

mjdmin: float
The beginning of the search interval (modified Julian date). It must be covered by the ephemeris.

mjdmax: float
The end of the search interval (modified Julian date). It must be covered by the ephemeris.

ra: float
The input RA coordinate (equatorial coordinate, in rad).

dec: float
The input DEC coordinate (equatorial coordinate, in rad).

pa: float
The position angle.

Returns

tuple
The lists of visibility period starts and ends with flags.

exoctk.contam_visibility.f_visibilityPeriods.**f_computeVisibilityPeriods**(*ephemeris*, *mjdmin*, *mjdmax*, *ra*, *dec*)

Returns two lists containing the start end end of each visibility period and a list containing a status flag

flag = 0 visibility period fully in the search interval flag = -1 start of the visibility period truncated by

the start of the search interval

flag = -2 end of the visibility period truncated by
the end of the search interval

flag = +1 the search interval is fully included in
the visibility period

Parameters

ephemeris: Ephemeris
The input ephemeris object.

mjdmin: float
The beginning of the search interval (modified Julian date). It must be covered by the ephemeris.

mjdmax: float
The end of the search interval (modified Julian date). It must be covered by the ephemeris.

ra: float
The input RA coordinate (equatorial coordinate, in rad).

dec: float
The input DEC coordinate (equatorial coordinate, in rad).

Returns

tuple
The lists of visibility period starts and ends with flags.

exoctk.contam_visibility.f_visibilityPeriods.**f_computeVisibilityPeriodsWithPA**(*ephemeris*, *mjdmin*, *mjdmax*, *ra*, *dec*, *pa*)

Returns two lists containing the start end end of each visibility period and a list containing a status flag

flag = 0 visibility period fully in the search interval flag = -1 start of the visibility period truncated by

the start of the search interval

flag = -2 end of the visibility period truncated by
the end of the search interval

flag = +1 the search interval is fully included in
the visibility period

Parameters

ephemeris: Ephemeris
The input ephemeris object.

mjdmin: float
The beginning of the search interval (modified Julian date). It must be covered by the ephemeris.

mjdmax: float
The end of the search interval (modified Julian date). It must be covered by the ephemeris.

ra: float
The input RA coordinate (equatorial coordinate, in rad).

dec: float
The input DEC coordinate (equatorial coordinate, in rad).

pa: float
The position angle.

Returns

tuple
The lists of visibility period starts and ends with flags

1.5 exoctk.contam_visibility.math_extensionsx module

This module provides simple extensions to the Python mathematical library. 2018/06/26 Made PEP8 compliant and added sind() and cosd() - Joe Filippazzo Version 1 August 23, 2010 RLH - Added OBLIQUITY. Version 0 August 6, 2010 RLH - Created

class exoctk.contam_visibility.math_extensionsx.**Circle**(*radius*)
Bases: object

Class to represent a circle.

Initialize a circle with a specified radius.

Parameters

radius: float
The radius of the circle.

area()
Returns the area of the circle.

class exoctk.contam_visibility.math_extensionsx.**ContinuousHistogram**(*boundaries*, *highest_inclusive=False*)
Bases: exoctk.contam_visibility.math_extensionsx.Histogram

Class to represent a histogram with continuous values.

Initializes a continuous histogram.

Default behavior with highest_inclusive = False:
Bin 0 is defined by x <= boundaries[0]. For i > 0, bin i is defined by boundaries[i-1] < x <= boundaries[i]. Bin n+1 is defined by x > boundaries[n-1].

Behavior with highest_exclusive = True:
Bins below n are defined in the same way as above. Bin n is defined by boundaries[n-2] < x < boundaries[n-1]. Bin n+1 is defined by x >= boundaries[n-1].

Parameters

boundaries: sequence
List of numbers that separate the bins, in increasing order.

highest_inclusive: bool
True if highest bin includes the last boundary, False (default) otherwise.

`retrieve_boundaries()`
Returns the list of boundaries of a continuous histogram.

`store_items(value, count=1)`
Stores a value in the continuous histogram.

Parameters

value: float
The value to store.

count: int
Number of items with that value to store (default 1).

`class exoctk.contam_visibility.math_extensionsx.DiscreteBin(bin_value)`
Bases: `exoctk.contam_visibility.math_extensionsx.HistogramBin`

Class to represent a bin with a fixed value.

Constructor for a fixed-value bin

Parameters

bin_value: float
The value of the bin.

`ismatch(value)`
Returns True if the value matches the bin, False otherwise

Parameters

value: float
The value to compare.

Returns

bool
True if matches, else False.

`class exoctk.contam_visibility.math_extensionsx.DiscreteHistogram(values)`
Bases: `exoctk.contam_visibility.math_extensionsx.Histogram`

Class to represent a histogram with discrete values.

Initializes a histogram with discrete values.

Parameters

values: sequence
List of the discrete values.

`retrieve_count_by_value`(*value*)
Returns the count matching a certain value. If not found, return None

Parameters

value: float
The value to retrieve.

`retrieve_values`()
Returns the list of bin values of a discrete histogram

`store_items`(*value*, *count=1*)
Stores a value in the discrete histogram if it matches one of the bin values.

Count = number of items with that value to store (default 1).

Returns True if a match was found and the value could be stored, False otherwise

Parameters

value: float
The value to store.

count: int
Number of items with that value to store (default 1).

Returns

found
[bool] Whether or not a match was found.

class exoctk.contam_visibility.math_extensionsx.**Histogram**
Bases: `object`

Class to represent a histogram.

`normalize`(*total=None*)
Takes a histogram and returns a new histogram that normalizes all its values.

Parameters

total: int
Number of items to divide each bin by for the normalization. If not supplied, it defaults to the total in the histogram.

Returns

new_histogram
[Histogram] The new histogram.

num_items()
Returns the total number of items stored in the histogram

retrieve_count(*bin_index*)
Returns the number of items stored in a given bin of the histogram.

Parameters

bin_index: int
The index to use (starts with 1).

Returns

int
The number of items in the bin.

class exoctk.contam_visibility.math_extensionsx.**HistogramBin**
Bases: object

Class to represent a bin within a histogram.

store_items(*num_items=1*)
Stores a given number of items in the bin.

Parameters

num_items: int
Number of items to store (default 1).

class exoctk.contam_visibility.math_extensionsx.**LinearEquation**(*coeff0*, *coeff1*)
Bases: exoctk.contam_visibility.math_extensionsx.Polynomial

Subclass of Polynomial for linear equations. This implementation is three times faster, so Polynomial should be reserved for higher orders.

Constructor for a linear equation to provide a more 'natural' interface without using a list.

Parameters

coeff0: float
Additive constant.

coeff1: float
Multiplicative coefficient.

apply(*value*)
Applies a linear equation to an input value.

This is intended to be faster than the more general method with Polynomial.

Parameters

value: float
The evaluand.

Returns

float
The result of the evaluated equation.

class exoctk.contam_visibility.math_extensionsx.**PoissonDistribution**(*mean*, *max_boundary*)
Bases: exoctk.contam_visibility.math_extensionsx.DiscreteHistogram

Class to represent a Poisson distribution.

Constructor function for the Poisson distribution.

Parameters

mean: float
Mean parameter for the Poisson distribution.

max_boundary: float
The largest parameter for which the probability is to be computed. All values larger than max_boundary will be lumped into the highest bin.

cumulative_probability(*value*)
Returns the probability that a random variable will have a value no greater than the one specified.

Parameters

value: float
The value between 0 and the max_boundary of the distribution.

Returns

float
The probability.

generate_distribution()
Populates a Poisson distribution up to the maximum bin.

probability(*k*)
Computes the probability that the Poisson distribution takes on the value k.

Value must be a nonnegative integer.

Parameters

k: float
The value to compute.

Returns

float
The probability.

retrieve_count_by_value(*value*)
Returns the number of items in the histogram that have the designated value.

Value must be an integer between 0 and max_boundary.

Parameters

value: float
The value to retrieve.

Returns

result
[int] The number of items with the given value.

`retrieve_values()`
Returns the list of bin values for the Poisson distribution.

class exoctk.contam_visibility.math_extensionsx.**Polynomial**(*coefficients*)
Bases: object

Class to represent a polynomial.

Constructor for a polynomial. Coefficients = a list of coefficients, starting with order 0 and increasing.

Parameters

coefficients: sequence
The list of coefficients, starting with order 0 and increasing.

`apply(value)`
Returns the result of applying a polynomial to an input value

Parameters

value: float
The evaluand.

Returns

float
The result of the evaluated equation.

class exoctk.contam_visibility.math_extensionsx.**RangeBin**(*min_value=None*, *max_value=None*, *lower_inclusive=False*, *upper_inclusive=True*)
Bases: exoctk.contam_visibility.math_extensionsx.HistogramBin

Class to represent a bin with a range.

Constructor for a range bin.

Parameters

min_value: float
Minimum value for the bin.

max_value: float
Maximum value for the bin.

lower_inclusive: bool
True if min_value is inclusive, else False (default).

upper_inclusive: bool
True if max_value is inclusive, else False (default).

`describe_limits(precision=2)`
Returns a printed representation of the limits of the bin.

Parameters

precision: int
Number of digits to print after the decimal point.

Returns

str
The limits of the bin.

`ismatch`(*value*)
Indicates whether the bin matches the value.

Parameters

value: float
The value to compare.

Returns

bool
True if matches, else False.

`istoo_high`(*value*)
Returns True if the specified value is too high for the bin. Assumes the bin has an upper limit.

Parameters

value: float
The value to compare.

Returns

bool
True if too high, else False.

class exoctk.contam_visibility.math_extensionsx.**Rectangle**(*length*, *width*)
Bases: `object`

Class to represent a rectangle.

Initialize a rectangle with a specified length and width.

Parameters

length: float
The length of the rectangle.

width: float
The width of the rectangle.

`area`()
Returns the area of the rectangle.

`motion_tolerant_area`(*motion_length*, *motion_angle*)
Returns the area within a rectangle that can tolerate a motion in a known direction while remaining within the rectangle.

Parameters

motion_length: float
Distance of motion (same units as rectangle length and width).

motion_angle: float
Angle in radians between the direction of motion and long direction of rectangle.

Returns

float
The area.

class exoctk.contam_visibility.math_extensionsx.**Square**(*side*)
Bases: exoctk.contam_visibility.math_extensionsx.Rectangle

Class to represent a square.

Initialize a square with a specified side length.

Parameters

side: float
The length of the square side.

inner_area(*excluded_width*)
Returns the area of the square after removing a strip of specified width along each edge.

Parameters

excluded_width: float
The width of the strip to remove.

Returns

float
The area.

class exoctk.contam_visibility.math_extensionsx.**StatisticalList**(*data=None*)
Bases: list

Numeric list class with statistical attributes.

Initializes a statistical list.

Parameters

data: sequence
List of inputs to list.

compute_rms()
Computes the rms value of a statistical list.

compute_statistics(*min_value=None*, *max_value=None*, *max_bins=None*)
Computes statistics for a StatisticalList object; must contain at least one element.

Parameters

min_value: float
Minimum value for cutoff of histogram (defaults to minimum in list).

max_value: float
Maximum value for cutoff of histogram (defaults to maximum in list).

max_bins: int
Maximum number of bins in histogram.

`compute_variance()`
Computes the variance of a statistical list.

`exoctk.contam_visibility.math_extensionsx.`**`acos2`**`(`*`val`*`)`
Safe version of acos that handles invalid arguments in the same way as asin2

Parameters

val: float
The evaluand.

Returns

float
The arccos of the value.

`exoctk.contam_visibility.math_extensionsx.`**`asin2`**`(`*`val`*`)`
Safe version of asin that handles invalid arguments.

Arguments greater than 1 are truncated to 1; arguments less than -1 are set to -1.

Parameters

val: float
The evaluand.

Returns

float
The arcsin of the value.

`exoctk.contam_visibility.math_extensionsx.`**`atan2d`**`(`*`x`*`)`
Return the arctan in degrees

Parameters

x: float
The evaluand.

Returns

float
The arctan of x in degrees.

`exoctk.contam_visibility.math_extensionsx.`**`average_histograms`**`(`*`histograms`*`)`
Takes a list of histogram objects and simply averages all the bin values.

All histograms in the list must be identical except for the count.

Parameters

histograms: sequence
A lst of Histogram objects to combine.

Returns

new_histogram
[Histogram] The averaged histogram.

exoctk.contam_visibility.math_extensionsx.**avg**(*l*)
Returns the average of a list of numbers

Parameters

l: sequence
The list of numbers.

Returns

float
The average.

exoctk.contam_visibility.math_extensionsx.**avg2**(*num1*, *num2*)
Returns the average of two numbers

Parameters

num1: float
The first number.

num2: float
The second number.

Returns

float
The average.

exoctk.contam_visibility.math_extensionsx.**combine_histograms**(*histograms*)
Takes a list of histograms and returns a new Histogram object that sums the values in each bin.

All histograms in the list must be identical except for the count.

Parameters

histograms: sequence
A lst of Histogram objects to combine.

Returns

Histogram
The combined histogram.

exoctk.contam_visibility.math_extensionsx.**conditional_probability**(*p_joint*, *p_B*)
Returns probability of event A given event B.

Parameters

p_joint: float
P(A,B).

p_B: float
Probability of event B.

Returns

float
The probability.

exoctk.contam_visibility.math_extensionsx.**cosd**(*x*)
Return the cos in degrees

Parameters

x: float
The evaluand.

Returns

float
The cos of x in degrees.

exoctk.contam_visibility.math_extensionsx.**factorial**(*num*)
Returns the factorial of a nonnegative integer. This function is provided in math module starting with Python 2.6, but implement anyway for compatibility with older systems.

Parameters

num: int
The number to factorialize.

Returns

int
The factorial.

exoctk.contam_visibility.math_extensionsx.**output_as_percentage**(*num*, *fractional_digits=1*)
Output a percentage neatly.

Parameters

num: float
The number to make into a percentage.

fractional_digits: int
Number of digits to output as fractions of a percent. If not supplied, there is no reduction in precision.

Returns

str
The percentage.

exoctk.contam_visibility.math_extensionsx.**percent_str**(*num*, *fractional_digits=1*)
Output a number as a percentage.

Parameters

num: float
The number to make into a percentage.

fractional_digits: int
Number of digits to output as fractions of a percent. If not supplied, there is no reduction in precision.

Returns

str
The percentage.

exoctk.contam_visibility.math_extensionsx.**really_greater_than**(*x*, *y*)
Safe greater-than function that returns true if and only if x is "significantly" greater than y

Parameters

x: float
The first number.

y: float
The second number.

Returns

bool
True if x is greater, else False.

exoctk.contam_visibility.math_extensionsx.**really_less_than**(*x*, *y*)
Safe less-than function that returns true if and only if x is "significantly" less than y.

Parameters

x: float
The first number.

y: float
The second number.

Returns

bool
True if x is less, else False.

exoctk.contam_visibility.math_extensionsx.**sind**(*x*)
Return the sin in degrees.

Parameters

x: float
The evaluand.

Returns

float
The sin of x in degrees.

exoctk.contam_visibility.math_extensionsx.**stdev**(*l*)
Standard deviation of a list of numbers that represent sample values

Parameters

l: sequence
The list to take the standard deviation of.

Returns

float
The standard deviation.

exoctk.contam_visibility.math_extensionsx.**variance**(*l*)
Variance of a list of numbers that represent sample values

Parameters

l: sequence
The list to take the variance of.

Returns

float
The variance.

1.6 exoctk.contam_visibility.quaternionx module

Version 4 September 9, 2010 WMK Flipped sign of the angle in the QX, QY, QZ, QJX, QJY, QJZ, set_values, set_as_QX, . . . functions to be consistent with the corrected multiplication. Also updated the doc strings.

Version 3 September 8, 2010 RLH Backed out change to cnvrt in version 2.

Version 2 September 3, 2010 RLH Fixed sign error in quaternion multiplication functions. Added quaternion __str__ method. Modified cnvrt method to return a CelestialVector

4/9/2010 WMK Redefined the __init__ inputs Changed from a Vector internal representation to 3 scalers Fixed an error in cvt_att_Q_to_angles, was assuming an att2inertial Quaternion! Streamlined some of the functions

Version 1.0 August 3, 2010 Got rid of degrees trig functions.

Combined this and rotationsx.py module to avoid circular imports and made it PEP Compliant Joe Filippazzo - 2018/06/26

class exoctk.contam_visibility.quaternionx.**Attitude**(*ra=0.0*, *dec=0.0*, *pa=0.0*, *frame='eq'*, *degrees=True*)
Bases: exoctk.contam_visibility.quaternionx.CelestialVector

Defines an Observatory attitude by adding a position angle.

Constructor for an Attitude.

pa = position_angle in degrees(default) or radians if degrees=False is specified. Other arguments are the same as with CelestialVector

Parameters

ra: float
The right ascension.

dec: float
The declination.

pa: float
The position angle.

frame: str
The frame to use.

degrees: bool
Use degrees.

class exoctk.contam_visibility.quaternionx.**CelestialVector**(*ra=0.0*, *dec=0.0*, *frame='eq'*, *degrees=True*)

Bases: exoctk.contam_visibility.quaternionx.Vector

Class to encapsulate a unit vector on the celestial sphere.

Constructor for a celestial vector.

There are two spherical coordinates, a longitudinal coordinate (called right ascension), and a latitudinal coordinate (called declination). The RA is defined as the counterclockwise angle from a reference direction on the equatorial plane; it ranges from 0-360 degrees. The DEC is the angle between the vector and the equatorial plane; it ranges from -90 to 90 degrees. Angles are specified in degrees but represented internally as radians.

The frame attribute indicates the coordinate frame of the vector, which may be 'eq' (equatorial, default), 'ec' (ecliptic), or 'gal' (galactic). In equatorial coordinates, the equatorial plane is the celestial equator (extension of the Earth's equator) and the reference axis is the vernal equinox. In ecliptic coordiantes, the equatorial plane is the ecliptic (the Earth's orbital plane) and the reference axis is usually defined relative to the Sun. In galactic coordinates, the equatorial plane is the plane of the Galaxy.

The degrees attribute should be True if the RA, DEC inputs are in degrees. Otherwise radians is assumed.

The coordinates "ra" and "dec" may be used in all three systems. Other names for coordinates in different frames may be defined for clarity.

A CelestialVector is also an ordinary unit vector, with Cartesian coordinates defined relative to the equatorial plane.

Parameters

ra: float
The right ascension.

dec: float
The declination.

frame: str
The frame to use.

degrees: bool
Use degrees.

position_angle(*v*)
Returns the position angle of v at the self vector, in radians.

v is an arbitrary vector that should be a CelestialVector object. The position angle is the angle between the North vector on the plane orthogonal to the self vector and the projection of v onto that plane, defined counterclockwise. See "V3-axis Position Angle", John Isaacs, May 2003 for further discussion.

Parameters

v: Vector
The vector to measure against.

Returns

pa
[float] The position angle between the two vectors.

`rotate_about_axis`(*angle*, *axis*)
This rotates a vector about an axis by the specified angle by using a rotation matrix. A new vector is returned.

Axis must be 'x', 'y', or 'z'. The x-rotation rotates the y-axis toward the z-axis. The y-rotation rotates the z-axis toward the x-axis. The z-rotation rotates the x-axis toward the y-axis.

Parameters

angle: float
The angle of rotation.

axis: str
The axis to rotate about, ['x', 'y', 'z'].

Returns

result
[Vector] The rotated vector.

`rotate_about_eigenaxis`(*angle*, *eigenaxis*)
Rotates a vector about arbitrary eigenaxis.

eigenaxis = Vector object (axis about which to rotate). angle = angle to rotate by in radians. Rotation is counterclockwise looking outward from origin along eigenaxis. Function uses rotation matrix from Rodrigues formula.

Note: This function is more general than rotate_about_axis above and could be used in its place. However, rotate_about_axis is faster and clearer when the rotation axis is one of the Cartesian axes.

Parameters

angle: float
The angle of rotation.

eigenaxis: Vector
The eigenaxis to rotate about.

Returns

result
[Vector] The rotated vector.

`rotate_by_posang`(*pa*)
Returns the vector that results from rotating the self vector counterclockwise from the North projection onto the plane orthogonal to that vector by the specified position angle (in radians). See "V3-axis Position Angle", John Isaacs, May 2003 for further discussion.

Parameters

pa: float
The position angle.

Returns

result
[Vector] The rotated vector.

`rotate_using_quaternion`(*angle*, *eigenaxis*)
Rotates a vector about arbitrary eigenaxis using quaternion.

This is an alternative formulation for rotate_about_eigenaxis. Interface is the same as rotate_about_eigenaxis.

Parameters

angle: float
The angle of rotation.

eigenaxis: Vector
The eigenaxis to rotate about.

Returns

Vector
The rotated vector.

`set_eq`(*ra*, *dec*, *degrees=False*)
Modifies a celestial vector with a new RA and DEC.

degrees = True if units are degrees. Default is radians.

Parameters

ra: float
The right ascension.

dec: float
The declination.

degrees: bool
Use degrees.

`transform_frame`(*new_frame*)
Transforms coordinates between celestial and ecliptic frames and returns result as a new CelestialVector. If new coordinate frame is the same as the old, a copy of the vector is returned.

Parameters

new_frame: str
Convert to new frame.

Returns

result
[Vector] The transformed vector.

update_cartesian(*x=None*, *y=None*, *z=None*)
Modifies a celestial vector by specifying new Cartesian coordinates.

Any subset of the Cartesian coordinates may be specifed.

Parameters

x: float
The extent in x.

y: float
The extent in y.

z: float
The extent in z.

class exoctk.contam_visibility.quaternionx.**GalacticPole**(*latitude*, *longitude*, *ascending_node*)
Bases: object

Represents coordinates of galactic pole.

Initializes the coordinates of the galactic pole.

Parameters

latitude: float
Latitude of pole, in degrees.

longitude: float
Longitude of pole, in degrees.

ascending_node: float
Ascending node of pole, in degrees.

class exoctk.contam_visibility.quaternionx.**Matrix**(*rows*)
Bases: list

Class to encapsulate matrix data and methods.

A matrix is simply a list of lists that correspond to rows of the matrix. This is just intended to handle simple multiplication and vector rotations. For anything more advanced or computationally intensive, Python library routines should be used.

Constructor for a matrix.

This accepts a list of rows. It is assumed the rows are all of the same length.

Parameters

rows: sequence
The rows of the matrix.

column(*col_index*)
Returns a specified column of the matrix as a numeric list.

Parameters

col_index: int
The column index.

Returns

list
The column values.

element(*row_index*, *col_index*)
Returns an element of the matrix indexed by row and column.

Indices begin with 0.

Parameters

row_index: int
The row index.

col_index: int
The column index.

Returns

float
The matrix value.

get_cols()
Returns list of all columns in a matrix.

num_cols()
Returns the number of columns in the matrix.

num_rows()
Returns the number of rows in the matrix.

row(*row_index*)
Returns a specified row of the matrix.

Parameters

row_index: int
The row index.

Returns

list
The row values.

class exoctk.contam_visibility.quaternionx.**NumericList**(*iterable=()*, */*)
Bases: list

List class that supports multiplication. Only valid for numbers.

exoctk.contam_visibility.quaternionx.**QX**(*angle*)
Creates rotation quaternion about X axis, rotates a vector about this axis.

Parameters

angle: float
The angle to rotate by.

exoctk.contam_visibility.quaternionx.**QY**(*angle*)
Creates rotation quaternion about Y axis, rotates a vector about this axis.

Parameters

angle: float
The angle to rotate by.

exoctk.contam_visibility.quaternionx.**QZ**(*angle*)
Creates rotation quaternion about Z axis, rotates a vector about this axis

Parameters

angle: float
The angle to rotate by.

exoctk.contam_visibility.quaternionx.**Qmake_a_point**(*V*)
Creates a pure Q, i.e. defines a pointing not a rotation

Parameters

V: Vector
The vector.

Returns

Quaternion
The point as a quaternion.

exoctk.contam_visibility.quaternionx.**Qmake_aperture2inertial**(*coord1*, *coord2*, *APA*, *xoff*, *yoff*, *s*, *YapPA*, *V3ref*, *V2ref*)
Creates a rotation Q, going from the target in aperture frame to body.

Parameters

coord1: float
The first coordinate.

coord2: float
The second coordinate.

APA: float
The apature position.

xoff: float
The x offset.

yoff: float
The y offset.

s: float
The multiplicative factor.

V2ref: float
The V2 position.

V3ref: float
The V3 position.

Returns

Quaternion
The rotation quaternion.

exoctk.contam_visibility.quaternionx.**Qmake_body2inertial**(*coord1*, *coord2*, *V3pa*)
Creates a rotation Q, going from the body frame to inertial.

Parameters

coord1: float
The first coordinate.

coord2: float
The second coordinate.

V3pa: float
The V3 position.

Returns

Quaternion
The rotation quaternion

exoctk.contam_visibility.quaternionx.**Qmake_v2v3_2body**(*v2*, *v3*)
Creates a rotation Q, going from v2 and v3 in the body frame to inertial.

Parameters

v2: float
The V2 position.

V3: float
The V3 position.

Returns

Quaternion
The rotation quaternion.

exoctk.contam_visibility.quaternionx.**Qmake_v2v3_2inertial**(*coord1*, *coord2*, *V3pa*, *v2*, *v3*)
Creates a rotation Q, going from v2 and v3 in the body frame to inertial

Parameters

coord1: float
The first coordinate.

coord2: float
The second coordinate.

V3pa: float
The V3 position.

v2: float
The V2 position.

v3: float
The V3 position.

Returns

Quaternion
The rotation quaternion.

class exoctk.contam_visibility.quaternionx.**Quaternion**(*V*, *q4*)
Bases: object

This representation is used by Wertz and Markley

Quaternion constructor.

Parameters

V: Vector
The vector to construct the quaternion with.

q4: Vector
The fourth vector.

cnvrt(*V*)
Rotates a vector from the starting frame to the ending frame defined by the Q.

Parameters

V: Vector
The vector to rotate.

Returns

Vector
The rotated Vector.

conjugate()
Returns a copy of the conjugated Q

inv_cnvrt(*V*)
Rotates a vector from the ending frame to the starting frame defined by the Q.

Parameters

V: Vector
The vector to invert.

Returns

Vector
The inverted Vector.

length()
Returns length of the Q

normalize()
Returns a copy of the Q normalized

set_as_QX(*angle*)
Sets quaterion in place like QX function.

Parameters

angle: float
The angle of rotation

set_as_QY(*angle*)
Sets quaterion in place like QY function

Parameters

angle: float
The angle of rotation

set_as_QZ(*angle*)
Sets quaterion in place like QZ function.

Parameters

angle: float
The angle of rotation.

set_as_conjugate()
Assigns conjugate values in place.

set_as_mult(*QQ1*, *QQ2*)
Sets self as QQ1*QQ2 in place for quaternion multiplication.

Parameters

QQ1: Quaternion
The first quaternion.

QQ2: Quaternion
The second quaternion.

set_as_point(*V*)
Set V as a point.

Parameters

V: Vector
The vector to set as a point.

set_equal(*Q*)
Assigns values from other Q to this one.

Parameters

Q: Quaternion
The quaternion value to set.

set_values(*V*, *angle*)
Sets quaterion values using a direction vector and a rotation of the coordinate frame about it.

Parameters

V: Vector
The direction Vector.

angle: float
The angle of rotation.

class exoctk.contam_visibility.quaternionx.**Vector**(*x=0.0*, *y=0.0*, *z=0.0*)
Bases: object

Class to encapsulate vector data and operations.

Constructor for a three-dimensional vector.

Note that two-dimensional vectors can be constructed by omitting one of the coordinates, which will default to 0.

Parameters

x: float
The x coordinate.

y: float
The y coordinate.

z: float
The z coordinate.

angle(*V2*)
Returns angle between the two vectors in degrees.

Parameters

V2: Vector
The vector to measure.

Returns

float
The angle between the two vectors.

create_matrix()
Converts a Vector into a single-column matrix.

cross(*V1*, *V2*)
returns cross product of two vectors

Parameters

V1: Vector
The vector to cross.

V2: Vector
The vector to cross.

Returns

Vector
The resultant vector.

display()
Print the values

dot(*V2*)
returns dot product between two vectors.

Parameters

V2: Vector
The vector to dot.

Returns

Vector
The resultant vector.

length()
Returns magnitude of the vector

normalize()
Returns copy of the normalized vector

rx()
The magnitude of x

ry()
The magnitude of y

rz()
The magnitude of z

set_eq(*x=None*, *y=None*, *z=None*)
Assigns new value to vector.

Arguments are now optional to permit this to be used with 2D vectors or to modify any subset of coordinates.

Parameters

x: float
The x coordinate.

y: float
The y coordinate.

z: float
The z coordinate.

set_xyz(*ra*, *dec*)
Creates a unit vector from spherical coordinates

Parameters

ra: float
The right ascension.

dec: float
The declination.

exoctk.contam_visibility.quaternionx.**angle**(*V1*, *V2*)
returns angle between two vectors in degrees, non class member.

Parameters

V1: Vector
The first vector.

V2: Vector
The second vector.

Returns

float
The angle between the vectors.

exoctk.contam_visibility.quaternionx.**cross**(*v1*, *v2*)
Returns cross product between two vectors, non class member.

Parameters

v1: Vector
The first vector.

v2: Vector
The second vector.

Returns

float
The cross product of the vectors.

exoctk.contam_visibility.quaternionx.**cvt_body2inertial_Q_to_c1c2pa_tuple**(*Q*)
Creates a angle tuple from Q, assuming body frame to inertial Q and 321 rotation sequence.

Parameters

Q: Quaternion
The quaternion.

Returns

coord1
[float] The first coordinate.

coord2
[float] The second coordinate.

pa
[float] The poosition angle.

exoctk.contam_visibility.quaternionx.**cvt_c1c2_using_body2inertial_Q_to_v2v3pa_tuple**(*Q*, *coord1*, *coord2*)

Given Q and a position, returns v2, v3, V3PA tuple

Parameters

Q: Quaternion
The quaternion.

coord1: float
The first coordinate.

coord2: float
The second coordinate

Returns

coord1
[float] The first coordinate.

coord2
[float] The second coordinate.

pa
[float] The poosition angle.

exoctk.contam_visibility.quaternionx.**cvt_pt_Q_to_V**(*Q*)

Converts a pure (pointing) Q to a unit position Vector

Parameters

Q: Quaternion
The quaternion to convert to a Vector.

Returns

Vector
The point as a vector.

exoctk.contam_visibility.quaternionx.**cvt_v2v3_using_body2inertial_Q_to_c1c2pa_tuple**(*Q*, *v2*, *v3*)

Given Q and v2, v3 gives pos on sky and V3 PA.

Parameters

Q: Quaternion
The quaternion.

v2: float
The V2 position.

v3: float
The V3 position.

Returns

tuple
The coordinates and position angle

exoctk.contam_visibility.quaternionx.**dec_separation**(*v1*, *v2*)
Returns difference in declination between two CelestialVectors.

Parameters

v1: Vector
The first vector.

v2: Vector
The second vector.

Returns

float
The separation between Dec values

exoctk.contam_visibility.quaternionx.**dot**(*v1*, *v2*)
returns dot product between two vectors, non class member.

Parameters

v1: Vector
The first vector.

v2: Vector
The second vector.

Returns

float
The dot product of the vectors.

exoctk.contam_visibility.quaternionx.**make_celestial_vector**(*v*)
Takes a Vector object and creates an equivalent CelestialVector.

Input vector v must be a unit vector.

Parameters

v: Vector
The vector to convert.

Returns

result
[Vector] The updated vector.

exoctk.contam_visibility.quaternionx.**pos_V_to_ra_dec**(*V*)
Returns tuple of spherical angles from unit direction Vector.

Parameters

V: Vector
The vector to analyze.

Returns

ra
[float] The ra of the vector.

dec
[float] The dec of the vector.

exoctk.contam_visibility.quaternionx.**projection**(*v*, *axis*)
Returns projection of vector v on plane normal to axis.

First take cross-product of v and the axis and normalize it. Then cross the axis with the result and return a CelestialVector. See http://www.euclideanspace.com/maths/geometry/elements/plane/ lineOnPlane/index.htm.

Parameters

v: Vector
The vector to convert.

axis: str
The axis to project onto.

Returns

Vector
The updated vector.

exoctk.contam_visibility.quaternionx.**ra_delta**(*v1*, *v2*)
Returns difference in right ascension between two CelestialVectors.

Parameters

v1: Vector
The first vector.

v2: Vector
The second vector.

Returns

delta_ra
[float] The difference in RA.

exoctk.contam_visibility.quaternionx.**ra_separation**(*v1*, *v2*)
Returns separation in right ascension between two CelestialVectors. This is accurate only if the difference in declination is small.

|sep| = DELTA-RA cos DEC

Parameters

v1: Vector
The first vector.

v2: Vector
The second vector.

Returns

float
The separation between RA values.

exoctk.contam_visibility.quaternionx.**separation**(*v1*, *v2*, *norm=False*)
Returns angle between two unit vectors in radians.

The angle between two normalized vectors is the arc-cosine of the dot product. Unless the norm attribute is set to True, it is assumed the vectors are already normalized (for performance).

Parameters

v1: Vector
The first vector.

v2: Vector
The second vector.

norm: bool
Normalize the vectors.

Returns

separation
[float] The separation of the vectors.

exoctk.contam_visibility.quaternionx.**vel_ab**(*U*, *Vel*)
Takes a unit vector and a velocity vector(km/s) and returns a unit vector modidifed by the velocity abberation.

Parameters

U: Vector
The unit vector.

Vel: Vector
The velocity vector to multiply.

Returns

Vector
The modified vector.

1.7 exoctk.contam_visibility.resolve module

exoctk.contam_visibility.resolve.**resolve_target**(*targetName*)

1.8 exoctk.contam_visibility.sossContamFig module

exoctk.contam_visibility.sossContamFig.**contam**(*cube*, *instrument*, *targetName='noName'*, *paRange=[0, 360]*, *badPAs=array([], dtype=float64)*, *tmpDir=''*, *fig=''*, *to_html=True*)

1.9 exoctk.contam_visibility.sossFieldSim module

exoctk.contam_visibility.sossFieldSim.**fieldSim**(*ra*, *dec*, *instrument*, *binComp=''*)

Wraps `sossFieldSim`, `gtsFieldSim`, and `lrsFieldSim` together. Produces a field simulation for a target using any instrument (NIRISS, NIRCam, or MIRI).

Parameters

ra
[float] The RA of the target.

dec
[float] The Dec of the target.

instrument
[str] The instrument the contamination is being calculated for. Can either be (case-sensitive): 'NIRISS', 'NIRCam F322W2', 'NIRCam F444W', 'MIRI'

binComp
[sequence] The parameters of a binary companion.

Returns

simuCube
[np.ndarray] The simulated data cube. Index 0 and 1 (axis=0) show the trace of the target for orders 1 and 2 (respectively). Index 2-362 show the trace of the target at every position angle (PA) of the instrument.

exoctk.contam_visibility.sossFieldSim.**gtsFieldSim**(*ra*, *dec*, *filter*, *binComp=''*)

Produce a Grism Time Series field simulation for a target. Parameters ———- ra : float

The RA of the target.

dec
[float] The Dec of the target.

filter
[str] The NIRCam filter being used. Can either be: 'F444W' or 'F322W2' (case-sensitive)

binComp
[sequence] The parameters of a binary companion.

Returns

simuCube
[np.ndarray] The simulated data cube. Index 0 and 1 (axis=0) show the trace of the target for orders 1 and 2 (respectively). Index 2-362 show the trace of the target at every position angle (PA) of the instrument.

exoctk.contam_visibility.sossFieldSim.**lrsFieldSim**(*ra*, *dec*, *binComp=''*)
Produce a Grism Time Series field simulation for a target. Parameters ———- ra : float

The RA of the target.

dec
[float] The Dec of the target.

binComp
[sequence] The parameters of a binary companion.

Returns

simuCube
[np.ndarray] The simulated data cube. Index 0 and 1 (axis=0) show the trace of the target for orders 1 and 2 (respectively). Index 2-362 show the trace of the target at every position angle (PA) of the instrument.

exoctk.contam_visibility.sossFieldSim.**sossFieldSim**(*ra*, *dec*, *binComp=''*, *dimX=256*)
Produce a SOSS field simulation for a target.

Parameters

ra: float
The RA of the target.

dec: float
The Dec of the target.

binComp: sequence
The parameters of a binary companion.

dimX: int
The subarray size.

Returns

simuCub
[np.ndarray] The simulated data cube.

1.10 exoctk.contam_visibility.time_extensionsx module

Module containing library functions for time manipulation. Standard for time representation in this project is fractional days. Dates are represented as modified Julian dates (mjd). An mjd gives the number of days since midnight on November 17, 1858.

class exoctk.contam_visibility.time_extensionsx.**FlexibleInterval**(*est*, *lst*, *let*)
Bases: exoctk.contam_visibility.time_extensionsx.Interval

Class to represent an interval with flexibility on when it can start and end.

Constructor for a FlexibileInterval.

Parameters

est: float
Earliest start time (mjd).

lst: float
Latest start time (mjd).

let: float
Latest end time (mjd).

end_time()
Returns the end of the FlexibleInterval.

flexibility()
Returns the flexibility of the FlexibleInterval, in fractional days.

maximum_duration()
Returns the maximum duration of the FlexibleInterval, in fractional days.

start_time()
Returns the start of the FlexibleInterval.

class exoctk.contam_visibility.time_extensionsx.**Interval**(*start*, *end*)
Bases: object

Class to represent a simple temporal interval.

Constructor for an interval.

Parameters

start: float
The start time.

end: float
The end time.

duration()
Returns the duration of an interval in fractional days.

end_time()
Returns the end of the interval.

start_time()
Returns the start of the interval.

temporal_relationship(*time*)
Returns the temporal relationship between an interval and an absolute time.

Returns 'before' if the interval ends at or before the time, 'after' if the interval begins at or after the time, 'includes' if the time occurs during the interval.

Parameters

time: float
The time.

Returns

rel
[str] The temporal relationship.

exoctk.contam_visibility.time_extensionsx.**compute_mjd**(*year*, *day_of_year*, *hour*, *minute*, *second*)
Computes a modified Julian date from a date specified as a year, day of year, hour, minute, and second. Arguments should be integers.

Parameters

year: int
The year.

day_of_year: int
The day.

hour: int
The hour.

minute: int
The minute.

second: int
The second.

Returns

float
The modified julian day.

exoctk.contam_visibility.time_extensionsx.**days_in_year**(*year*)
Returns the number of days in a year.

Parameters

year: int
The year to search.

Returns

days
[int] The number of days that year.

exoctk.contam_visibility.time_extensionsx.**days_to_seconds**(*days*)
Takes a time in fractional days and converts it into integer seconds.

Parameters

days: float
The number of days as a float.

Returns

int
The number of seconds in as many days.

exoctk.contam_visibility.time_extensionsx.**display_date**(*mjd*)
Returns a string representation of the date represented by a modified Julian date.

Parameters

mjd: float
The modified julian day.

Returns

str
The MJD as a string.

exoctk.contam_visibility.time_extensionsx.**display_time**(*time*, *force_hours=False*)
Returns a string representation of a time specified in fractional days.

Parameters

time: float
The time as a float.

force_hours: bool
Force the hour calculation.

Returns

str
The time as a string.

exoctk.contam_visibility.time_extensionsx.**integer_days**(*time*)
Takes a time in fractional days and returns integer component.

Parameters

time: float
The float time.

Returns

int
The integer time.

exoctk.contam_visibility.time_extensionsx.**is_leap_year**(*year*)
Returns True if the year is a leap year, False otherwise.

Parameters

year: int
The year to check.

Returns

bool
Is the year a leap year?

exoctk.contam_visibility.time_extensionsx.**jd_to_mjd**(*jd*)
Converts a Julian date to a modified Julian date.

Parameters

jd: float
The true Julian day.

Returns

float
The modified Julian day.

exoctk.contam_visibility.time_extensionsx.**leap_years**(*year1*, *year2*)
Returns the number of leap years between year1 and year2, non-inclusive.

year1 and year2 must be integers, with year2 > year1

Parameters

year1: int
The start year.

year2: int
The end year.

Returns

int
The number of leap years between year1 and year2.

exoctk.contam_visibility.time_extensionsx.**mjd_from_string**(*time_string*)
Takes a string of the form yyyy.ddd:hh:mm:ss and returns an mjd.

Parameters

time_string: str
The MJD as a string.

Returns

float
The modified julian day.

exoctk.contam_visibility.time_extensionsx.**mjd_to_jd**(*mjd*)
Converts a modified Julian date to a true Julian date.

Parameters

mjd: float
The modified julian day.

Returns

float
The true Julian day.

exoctk.contam_visibility.time_extensionsx.**round_to_second**(*time*)
Rounds a time in days to the nearest second.

Parameters

time: int
The number of days as a float.

Returns

float
The number of seconds in as many days.

exoctk.contam_visibility.time_extensionsx.**seconds_into_day**(*time*)
Takes a time in fractional days and returns number of seconds since the start of the current day.

Parameters

time: float
The time as a float.

Returns

int
The day's duration in seconds.

exoctk.contam_visibility.time_extensionsx.**seconds_to_days**(*seconds*)
Takes a time in integer seconds and converts it into fractional days.

Parameters

seconds: int
The number of seconds.

Returns

float
The number of days as a float.

exoctk.contam_visibility.time_extensionsx.**time_from_string**(*time_string*)
Takes a string of the form ddd:hh:mm:ss and converts it to fractional days. All subfields above seconds are optional and may be omitted if the subfield and all higher-order ones are zero.

Parameters

time_string: str
The time as a string.

Returns

float
The fractional days.

1.11 exoctk.contam_visibility.visibilityPA module

Produces a graph of the visibility & accessible position angles for a given RA & DEC, and prints out corresponding information, including the ranges of accessible and inaccessible PAs.

Usage: python visibilityPA.py RA DEC [targetName]
if targetName is specified, then the figure is saved

-Created by David Lafreniere, March 2016 -makes use of (and hacks) several scripts created by Pierre Ferruit

that are part of the JWST Python tools JWSTpylib and JWSTpytools

exoctk.contam_visibility.visibilityPA.**checkVisPA**(*ra*, *dec*, *targetName=None*, *ephFileName=None*, *fig=None*)

Check the visibility at a range of position angles.

Parameters

ra: str
The RA of the target in hh:mm:ss.s or dd:mm:ss.s or representing a float

dec: str
The Dec of the target in hh:mm:ss.s or dd:mm:ss.s or representing a float

targetName: str
The target name

ephFileName: str
The filename of the ephemeris file

fig: bokeh.plotting.figure
The figure to plot to

Returns

paGood
[float] The good position angle.

paBad
[float] The bad position angle.

gd
[matplotlib.dates object] The greogrian date.

fig
[bokeh.plotting.figure object] The plotted figure.

exoctk.contam_visibility.visibilityPA.**convert_ddmmss_to_float**(*astring*)

Convert sexigesimal to decimal degrees

Parameters

astring: str
The sexigesimal coordinate.

Returns

hour_or_deg
[float] The converted coordinate.

exoctk.contam_visibility.visibilityPA.**fill_between**(*fig*, *xdata*, *pamin*, *pamax*, ***kwargs*)

exoctk.contam_visibility.visibilityPA.**using_gtvt**(*ra*, *dec*, *instrument*, *targetName='noName'*, *ephFileName=None*, *output='bokeh'*)

Plot the visibility (at a range of position angles) against time.

Parameters

ra
[str] The RA of the target (in degrees) hh:mm:ss.s or dd:mm:ss.s or representing a float

dec
[str] The Dec of the target (in degrees) hh:mm:ss.s or dd:mm:ss.s or representing a float

instrument
[str] Name of the instrument. Can either be (case-sensitive): 'NIRISS', 'NIRCam', 'MIRI', 'FGS', or 'NIRSpec'

ephFileName
[str] The filename of the ephemeris file.

output
[str] Switches on plotting with Bokeh. Parameter value must be 'bokeh'.

Returns

paGood
[float] The good position angle.

paBad
[float] The bad position angle.

gd
[matplotlib.dates object] The gregorian date.

fig
[bokeh.plotting.figure object] The plotted figure.

1.12 Module contents

Package to calculate target visibility and contamination

Groups and Integrations Calculator

The groups_integrations tool is a JWST observation planning tool designed with exoplanet observations in mind. Given a potential observation (which requires transit time, and an estimate of model and magnitude for the host star, and specifics of instrument setup) it's simple to get an optimized groups and integrations plan for the observation. The example notebook also outlines cases for batch demos – testing many transits/sources in a given instrument setup, or figuring out which instrument setup is best for a given transit.

The Groups and Integrations Calculator runs with pre-sampled pandeia data in the background – so it can have the power of those carefully built instrument models, but still run 100 times faster.

CHAPTER

TWO

GROUPS & INTEGRATIONS CALCULATOR

Contents

This is a quick demo for the Groups and Integrations Calculator (i.e. the `exoctk.groups_integrations` subpackage). This demo can run with the `exoctk-3.6` conda environment and a proper installation of the `exoctk` software package and the `EXOCTK_DATA` data package. For instructions on how to download and install these things, see the installation instructions.

The Groups and Integrations Calculator was written as an exoplanet-focused alternative to the JWST ETC (also known as `pandeia`). It's power comes from:

1. Using pre-sampled data and interpolating for a speedy calculation.
2. Having the power to provide you with the optimal configuration for your observation – instead of just letting you guess and check configurations.

This notebook has a few sections for clarity:

- Imports and Setup
- Input Parameter Space
- Building an Input Dictionary
- Running the Calculator
- Exploring the Outputs
- Two Examples for Batch Runs

2.1 Imports and Setup

In addition to installing `exoctk`, you'll need the pre-sampled data that the groups and integrations calculator runs on, in a `json` format. If you followed the installation instructions linked at the top of this notebook, you should have this under the environment variable `EXOCTK_DATA`, and we'll just take it from your `path` – otherwise you'll need to specify the path in this cell.

Also – **DON'T WORRY** if you get a big scary warning about the `matplotlib` backend. We need that in there to run server side when we host the website – but it won't affect any of this demo of your general `exoctk` usage. If it's truly too hideous run this cell twice and warning should vanish.

```
# Check if you have the environment variable
import os
EXOCTK_DATA = os.environ.get('EXOCTK_DATA')
if EXOCTK_DATA == '':
    GROUPS_INTEGRATIONS_DIR = '/path/to/your/local/copy/integrations_groups_data.json'
else:
    GROUPS_INTEGRATIONS_DIR = GROUPS_INTEGRATIONS_DIR = os.path.join(EXOCTK_DATA, 'groups_integrations/
↪groups_integrations.json')
```

```
# Imports
import json
from exoctk.groups_integrations.groups_integrations import perform_calculation
```

2.2 Input Parameter Space

We are often adding functionality and modes to this – so instead of providing a static list of what models, instrument modes, etc., are presently feasible, here's a little demo on how to walk through the data file yourself and see what parameters are possible.

```
# Open the file
with open(GROUPS_INTEGRATIONS_DIR) as f:
    parameter_space = json.load(f)

# Print the TA and Science observation modes
modes = [('science', 'sci_sat'), ('TA', 'ta_sat')]
for mode in modes:
    print('\n')
    print('Available modes for {}:\n'.format(mode[0]))
    instruments = list(parameter_space[mode[1]].keys())
    print('Instruments: {}\n'.format(instruments))
    for instrument in instruments:
        filters = list(parameter_space[mode[1]][instrument].keys())
        print('Filters for {}: {}'.format(instrument, filters))
        subarrays = list(parameter_space[mode[1]][instrument][filters[0]].keys())
        print('Subarrays for {}: {}\n'.format(instrument, subarrays))

# Print the available Phoenix models
print('\n')
print('Phoenix model keys :')
print('--------------------')
print(list(parameter_space['ta_sat'][instruments[-1]][filters[0]][subarrays[0]].keys()))

print('\n')
```

(continues on next page)

(continued from previous page)

```
print('Magnitude sampling :')
print('--------------------')
print(parameter_space['mags'])
```

```
Available modes for science:

Instruments: ['nirspec', 'niriss', 'miri', 'nircam']

Filters for nirspec: ['f070lp_g140m', 'f070lp_g140h', 'f170lp_g235h', 'f170lp_g235m', 'f290lp_g395m',
↪'f100lp_g140h', 'f290lp_g395h', 'f100lp_g140m', 'clear_prism']
Subarrays for nirspec: ['sub512', 'sub2048', 'sub1024a', 'sub1024b']

Filters for niriss: ['soss']
Subarrays for niriss: ['substrip256', 'substrip96']

Filters for miri: ['lrs']
Subarrays for miri: ['slitlessprism']

Filters for nircam: ['f322w2', 'f444w', 'f277w', 'f356w']
Subarrays for nircam: ['subgrism128', 'subgrism64', 'full', 'subgrism256']


Available modes for TA:

Instruments: ['nirspec', 'niriss', 'miri', 'nircam']

Filters for nirspec: ['f140x', 'f110w', 'clear']
Subarrays for nirspec: ['full', 'sub32', 'sub2048']

Filters for niriss: ['f480m']
Subarrays for niriss: ['subtasoss']

Filters for miri: ['f560w', 'f100w', 'f1500w']
Subarrays for miri: ['slitlessprism']

Filters for nircam: ['f335m']
Subarrays for nircam: ['sub32tats']


Phoenix model keys :
--------------------
['g5v', 'a1v', 'a3v', 'k5iii', 'm0i', 'm0v', 'g5iii', 'f5v', 'k0v', 'g5i', 'k2v', 'frame_time', 'm2i',
↪'k0i', 'g0iii', 'f5i', 'm0iii', 'a0v', 'g8v', 'a0i', 'mag', 'k7v', 'g2v', 'm5v', 'g0i', 'a5i', 'k5v',
↪'f8v', 'f0i', 'f0v', 'f2v', 'k5i', 'a5v', 'k0iii', 'g0v', 'm2v']


Magnitude sampling :
--------------------
[4.5, 6.5, 8.5, 10.5, 12.5]
```

2.3 Building an Input Dictionary

Running the groups and integrations calculator requires a dictionary of inputs. This section will go through an example input dictionary and what the limits on the parameters are.

```
# Initialize the dictionary
parameters = {}

# Source parameters
parameters['mag'] = 10 # 4.5 <= float <= 12.5
parameters['band'] = 'k' # only K band vega mag for now
parameters['mod'] = 'g2v' # Phoenix model per last section

# Observation specifics
parameters['obs_time'] = 5 # positive float, in hours
parameters['n_group'] = 'optimize' # 'optimize', or positive integer

# Detector setup -- within the modes of the last section
parameters['ins'] = 'nircam'
# For science observation
parameters['filt'] = 'f444w'
parameters['subarray'] = 'subgrism256'
# And target acquisition
parameters['filt_ta'] = 'f335m'
parameters['subarray_ta'] = 'sub32tats'

# Saturation level
parameters['sat_mode'] = 'well' # 'well', for full well fraction, or 'counts'
parameters['sat_max'] = .95 # < 1 for fullwell, and a positive integer always

# And feed in the data file
parameters['infile'] = GROUPS_INTEGRATIONS_DIR
input_dict = parameters.copy()
```

2.4 Running the Calculator

Now, running the calculator is relatively straightforward. We leaned on the `pandeia` function convention – so feeding our inputs into `perform_calculation` returns a dictionary of input parameters (a stripped down `pandiea` scene) as well as the results of the calculation. (Note that `perform_calculation` updates the `parameters` dictionary – so that object will be changed once you run the function.)

```
# Bookeeping for new/old parameters
inputs = list(parameters.keys())

# Perform the calculation
results = perform_calculation(parameters)
for key in results:
    if key in inputs:
        if key == 'infile':
            # hackers
            print('The input of infile was REDACTED!')
        else:
            print('The input of {} was {}.'.format(key, results[key]))
    else:
        print('The result of {} was {}'.format(key, results[key]))
```

```
The input of mag was 10.
The input of band was k.
The input of mod was g2v.
The input of obs_time was 5.
The input of n_group was 149.
The input of ins was nircam.
The input of filt was f444w.
The input of subarray was subgrism256.
The input of filt_ta was f335m.
The input of subarray_ta was sub32tats.
The input of sat_mode was well.
The input of sat_max was 55195.0.
The input of infile was REDACTED!
The result of n_col was 256
The result of n_row was 256
The result of n_amp was 1
The result of n_reset was 1
The result of n_frame was 1
The result of n_skip was 0
The result of t_frame was 1.347
The result of t_int was 200.657
The result of t_ramp was 200.657
The result of n_int was 90
The result of t_exp was 5.016
The result of t_duration was 5.05
The result of obs_eff was 0.993
The result of ta_t_frame was 0.01496
The result of min_ta_groups was 33
The result of max_ta_groups was 3
The result of t_duration_ta_min was 0.50864
The result of t_duration_ta_max was 0.05984
The result of max_sat_prediction was 55171.429
The result of max_sat_ta was 4613.154
The result of min_sat_ta was 50744.699
```

2.5 Exploring the Outputs

If you aren't quite familiar with the intricacies of a JWST observation, we'll unpack these results briefly.

Every JWST observation has a number of groups and integrations. Groups are how many frames you can pack into an integration, and generally this is goverened by how quickly your target will saturate on the detector. With every integration, there is overhead time added into the observation, and less time observing your actual transit. So, once the calculator has figured out the maximum possible groups before the detector is saturated, it will return that number of groups, how many integrations of that pattern it takes to fill up your whole transit time, and some additional helpful parameters like what's the maximum saturation you might reach during this observation, how long the actual scheme will take (since there will be a slightly different rounding when everything is added up), how efficient your observation is, etc.

For target acquisition, the efficiency is less in question than the ability of the detector to hit the minimum SNR required. This provides a reccomendation, so you know the minimum and maxmimum possible groups you can use, and can make an informed decision.

```
# So let's make a nice printed summary
print('The total time for science + TA observation scheme is {}-{} hours.'.format(
    results['t_duration']+results['t_duration_ta_max'], results['t_duration']+results['t_duration_ta_min
↪'']))
```

(continues on next page)

(continued from previous page)

```
print('You need {} groups and {} integrations for the science observation.'.format(
    results['n_group'], results['n_int']))
print('You need between {} and {} groups for target acquisition.'.format(
    results['max_ta_groups'], results['min_ta_groups']))
print('We estimate your science observation will reach at most {} counts -- how close were we to your␣
↪cutoff of {}?'.format(
    results['max_sat_prediction'], results['sat_max']))
print('With this observation scheme {}% of the observation will be science data.'.format(results['obs_
↪eff']*100))
```

```
The total time for science + TA observation scheme is 5.10984-5.55864 hours.
You need 149 groups and 90 integrations for the science observation.
You need between 3 and 33 groups for target acquisition.
We estimate your science observation will reach at most 55171.429 counts -- how close were we to your␣
↪cutoff of 55195.0?
With this observation scheme 99.3% of the observation will be science data.
```

2.6 Two Examples for Batch Runs

So far we've shown you how to run this one off – just like you would in the web tool. Here are two examples for running many calculations. Because the calculator is so light and only has a single parameter, it won't be particularly computationally expensive or logistically difficult to parallelize.

- Checking many potential observations in *one* instrument/mode/filter.
- Checking one transit in *many* instruments/modes/filters.

2.7 Checking many potential observations in *one* instrument/mode/filter

```
# Some imports/set up for ease of this part of the demo
from multiprocessing import Pool
p = Pool(4) # Feel free to set it higher but this will place nice with most laptops.

from astropy.io import ascii
import numpy as np
```

```
# Say you have a table of sources you want to read in.
# sources = ascii.read('path/to/source_table.csv')
# Since we don't we'll just make one up with reasonable transit objects
sources = {'mod': ['k0v', 'k5v', 'g5v', 'f5i', 'g0iii', 'f0i', 'k0iii', 'g2v', 'm0v', 'k5iii', 'm0i',
↪'g0v', 'g8v', 'f0v', 'g0i'],
           'obs_time': [3 + n*.15 for n in range(15)],
           'mag': [9 + n*.15 for n in range(15)]}

# Now use this to create input dictionaries
input_sources = []
for index, elem in enumerate(sources['mod']):
    input_source = input_dict.copy()
    input_source['mod'] = elem
    input_source['obs_time'] = sources['obs_time'][index]
```

(continues on next page)

(continued from previous page)

```
    input_source['mag'] = sources['mag'][index]
    input_sources.append(input_source)

# And run it in parallel
single_mode_results = p.map(perform_calculation, input_sources)
```

```
# And explore the output
obs_eff = [result['obs_eff'] for result in single_mode_results]
indeces = np.where(obs_eff == np.max(obs_eff))[0]
bests = [single_mode_results[index] for index in indeces]
for best in bests:
    print('One of the best sources is {}, {}, {} at {} efficiency.'.format(best['mod'], best['obs_time
↪'], best['mag'], best['obs_eff']))
    print('(This means {} groups and {} integrations.)'.format(best['n_group'], best['n_int']))
```

```
One of the best sources is g0i, 5.1, 11.1 at 0.998 efficiency.
(This means 406 groups and 34 integrations.)
```

2.8 Checking one transit in *many* instruments/modes/filters

```
# Let's take the LEAST efficient observation from the last example
# We'll see if it plays more nicely with another instrument, filter, or subarray.
worst = single_mode_results[np.where(obs_eff == np.min(obs_eff))[0][0]]
print("We're starting at a baseline of {} efficiency.".format(worst['obs_eff']))
print('Source : {}, {}, {} mode.'.format(worst['mod'], worst['mag'], worst['obs_time']))
print('Mode : {}, {}, {}'.format(worst['ins'], worst['filt'], worst['subarray']))
for key in ['mod', 'mag', 'obs_time']:
    input_dict[key] = worst[key]

# We'll call back to the parameter_space dictionary we walked through to look at available modes
modes = []
for ins in parameter_space['sci_sat'].keys():
    for filt in parameter_space['sci_sat'][ins].keys():
        for sub in parameter_space['sci_sat'][ins][filt].keys():
            input_mode = input_dict.copy()

            # Alter the science setup
            input_mode['ins'] = ins
            input_mode['filt'] = filt
            input_mode['subarray'] = sub

            # And we care less about TA so pick the default for each instrument
            input_mode['filt_ta'] = list(parameter_space['ta_sat'][ins].keys())[0]
            input_mode['subarray_ta'] = list(parameter_space['ta_sat'][ins][input_mode['filt_ta']].
↪keys())[0]
            modes.append(input_mode)

# And run it in parallel
single_source_results = p.map(perform_calculation, modes)
```

```
We're starting at a baseline of 0.984 efficiency.
Source : k0v, 9.0, 3.0 mode.
Mode : nircam, f444w, subgrism256
```

```
# And, again explore the output
obs_eff = [result['obs_eff'] for result in single_source_results]
indeces = np.where(obs_eff == np.max(obs_eff))[0]
bests = [single_source_results[index] for index in indeces]
for best in bests:
    print('The best mode is {}, {}, {} at {} efficiency.'.format(best['ins'], best['filt'], best[
↪'subarray'], best['obs_eff']))
    print('(This means {} groups and {} integrations.)'.format(best['n_group'], best['n_int']))
```

```
The best mode is nircam, f444w, subgrism64 at 0.996 efficiency.
(This means 234 groups and 135 integrations.)
```

CHAPTER

THREE

EXOCTK.GROUPS_INTEGRATIONS PACKAGE

3.1 Submodules

3.2 exoctk.groups_integrations.groups_integrations module

This is a module for calcuating groups and integrations with JWST on the fly. the main function (perform_calculation) takes a dictionary of inputs (modeled around how the web tool takes inputs) which must include : obs_time, n_group, mag, mod, band, filt, filt_ta, ins, subarray, subarray_ta, sat_mode, sat_max, and infile. It produces and dictionary of outputs that includes all of the original information as well as groups, integrations, saturation levels, and observation time estimates for target acquisition and science observations with JWST.

3.2.1 Authors

Jules Fowler, April 2017

3.2.2 Use

This is mostly a module to be used by ExoCTKWeb – but the main function can be run standalone.

exoctk.groups_integrations.groups_integrations.**calc_groups_from_exp_time**(*max_exptime_per_int*, *t_frame*)

Given the maximum saturation time, calculates the number of frames per group.

Parameters

max_exptime_per_int
[float] The maximum number of seconds an integration can last before it's oversaturated.

t_frame
[float] The time per frame.

Returns

groups
[int] The required number of groups.

exoctk.groups_integrations.groups_integrations.**calc_n_int**(*transit_time*, *n_group*, *n_reset*, *t_frame*, *n_frame*)

Calculates number of integrations required.

Parameters

transit_time
[float] The time of your planeting transit (in hours.)

n_group
[int] Groups per integration.

n_reset
[int] Reset frames per integration.

t_frame
[float] The frame time (in seconds).

n_frame
[int] Frames per group – always 1 except maybe brown dwarves.

Returns

n_ints
[float] The required number of integraitions.

exoctk.groups_integrations.groups_integrations.**calc_obs_efficiency**(*t_exp*, *t_duration*)
Calculates the observation efficiency.

Parameters

t_exp
[float] Exposure time (in seconds).

t_duration
[float] Duration time (in seconds).

Returns

obs_eff
[float] Observation efficiency.

exoctk.groups_integrations.groups_integrations.**calc_t_duration**(*n_group*, *n_int*, *n_reset*, *t_frame*, *n_frame=1*)
Calculates duration time (or exposure duration as told by APT.)

Parameters

n_group
[int] Groups per integration.

n_int
[int] Integrations per exposure.

n_reset
[int] Reset frames per integration.

t_frame
[float] Frame time (in seconds).

n_frame
[int, optional] Frames per group – always one except brown dwarves.

Returns

t_duration
[float] Duration time (in seconds).

exoctk.groups_integrations.groups_integrations.**calc_t_exp**(*n_int*, *t_ramp*)
Calculates exposure time (or photon collection duration as told by APT.)

Parameters

n_int
[int] Integrations per exposure.

t_ramp
[float] Ramp time (in seconds).

Returns

t_exp
[float] Exposure time (in seconds).

exoctk.groups_integrations.groups_integrations.**calc_t_frame**(*n_col*, *n_row*, *n_amp*, *ins*)
Calculates the frame time for a given ins/readmode/subarray.

Parameters

n_col
[int] Number of columns.

n_row
[int] Number of rows.

n_amp
[int] Amplifiers reading data.

ins
[str] The instrument key.

Returns:

t_frame
[float] The frame time (in seconds).

exoctk.groups_integrations.groups_integrations.**calc_t_int**(*n_group*, *t_frame*, *n_frame*, *n_skip*)
Calculates the integration time.

Parameters

n_group
[int] Groups per integration.

t_frame
[float] Frame time (in seconds).

n_frame
[int] Frames per group – always 1 except maybe brown dwarves.

n_skip
[int] Skips per integration – always 0 except maybe brown dwarves.

Returns

t_int
[float] Integration time (in seconds).

exoctk.groups_integrations.groups_integrations.**calc_t_ramp**(*t_int*, *n_reset*, *t_frame*)
Calculates the ramp time – or the integration time plus overhead for resets.

Parameters

t_int
[float] Integration time (in seconds).

n_reset
[int] Rest frames per integration

t_frame
[float] Frame time (in seconds).

Returns

t_ramp
[float] Ramp time (in seconds).

exoctk.groups_integrations.groups_integrations.**convert_sat**(*sat_max*, *sat_mode*, *ins*, *infile*, *ta=False*)
Converts full well fraction to a saturation in counts OR provides the max fullwell for TA mode.

Parameters

sat_max
[float] Either a full well fraction or counts.

sat_mode
[str] 'well' or 'counts'.

ins
[str] The instrument.

infile
[str] The path to the data file.

ta
[bool, optional] Whether or not it's TA mode.

Returns

sat_max
[float] The fullwell to use in counts.

exoctk.groups_integrations.groups_integrations.**interpolate_from_dat**(*mag*, *ins*, *filt*, *sub*, *mod*, *band*, *t_frame*, *sat_lvl*, *infile*, *ta=False*)
Interpolates the precalculated pandeia data to estimate the saturation limit.

Parameters

mag
[float] The magnitude of the source. (Takes between 4.5-12.5)

ins
[str] The instrument, allowable "miri", "niriss", "nirspec", "nircam".

filt
[str] Filter.

sub
[str] Subarray.

mod
[str] Phoenix model key.

band
[str] Magnitude band – unsed rn.

t_frame
[float] Frame time.

sat_lvl
[float] The maximum fullwell saturation we'll allow.

infile
[str] The data file to use.

ta
[bool, optional] Whether or not we're running this for TA.

Returns

n_group
[int] The number of groups that won't oversaturate the detector.

max_sat
[int] The maximum saturation level reached by that number of groups.

exoctk.groups_integrations.groups_integrations.**map_to_ta_modes**(*ins*, *max_group*, *min_group*)
Turns the min/max groups into the closest allowable TA group mode.

Parameters

ins
[str] Instrument.

max_group
[int] The maximum number of groups without oversaturating.

min_group
[int] The groups needed to hit the target SNR.

Returns

min_ta_groups
[int] The min possible groups to hit target SNR.

max_ta_groups
[int] The max possible groups before saturation.

exoctk.groups_integrations.groups_integrations.**min_groups**(*mag*, *ins*, *filt*, *sub*, *mod*, *band*, *infile*)
Estimates the minimum number of groups to reach target acq sat requirements.

Parameters

mag
[float] Magnitude of star.

ins
[str] Instrument.

filt
[str] Filter.

sub
[str] Subarray.

mod
[str] Phoenix model key.

band
[str, currently unused?] The band – right now only k sooo?

infile
[str] The file with the pandeia data.

Returns

min_groups
[int] The minimum number of groups to reach target snr.

exoctk.groups_integrations.groups_integrations.**perform_calculation**(*params*, *n_frame=1*, *n_skip=0*)
Calculates all of the outputs and puts them in a dictionary for easy access.

Parameters

params
[dict] Dictionary of all the needed parameters. Must include: obs_time, n_group, mag, mod, band, filt, filt_ta, ins, subarray, subarray_ta, sat_mode, sat_max, infile

n_frame :int, optional
The number of frames – almost always 1.

n_skip: int, optional
Number of skips – almost always 0

Returns

params
[dict, str] Dictionary of outputs and inputs. If the calculation throws an error it will return a string error message instead.

exoctk.groups_integrations.groups_integrations.**set_params_from_ins**(*ins*, *subarray*)
Sets/collects the running parameters from the instrument.

Parameters

ins
[str] Instrument, options are nircam, niriss, nirpec, and miri.

subarray
[str] Subarray mode.

Returns

rows
[int] The number of pixels per row.

cols
[int] The number of columns per row.

exoctk.groups_integrations.groups_integrations.**set_t_frame**(*infile*, *ins*, *sub*, *ta=False*)
Assign the appropriate frame time based on the ins and subarray. For now, modes are implied.

Parameters

infile: str
The path to the data file.

ins
[str] The instrument : 'miri', 'niriss', 'nirspec', or 'nircam'.

sub
[str] The subarray – too lazy to write out the options here.

ta
[bool] Whether this is for TA or not.

Returns

t_frame
[float] The frame time for this ins/sub combo.

3.3 Module contents

Lightcurve Fitting Tool

The light-curve fitting tool fits large numbers of spectroscopic light curves simultaneously while sharing model parameters across wavelengths and visits. It includes multiple uncertainty estimation algorithms and a comprehensive library of physical and systematic model components that are fully customizable.

CHAPTER

FOUR

EXOCTK.LIGHTCURVE_FITTING PACKAGE

4.1 Submodules

4.2 exoctk.lightcurve_fitting.fitters module

Functions used to fit models to light curve data

Author: Joe Filippazzo Email: jfilippazzo@stsci.edu

exoctk.lightcurve_fitting.fitters.**lmfitter**(*time*, *data*, *model*, *uncertainty=None*, *verbose=True*, ***kwargs*)

Use lmfit

Parameters

data: sequence
The observational data

model: ExoCTK.lightcurve_fitting.models.Model
The model to fit

uncertainty: np.ndarray (optional)
The uncertainty on the (same shape) data

method: str
The name of the method to use

name: str
A name for the best fit model

verbose: bool
Print some stuff

Returns

lmfit.Model.fit.fit_report
The results of the fit

4.3 exoctk.lightcurve_fitting.lightcurve module

Base and child classes to handle light curve fitting

Author: Joe Filippazzo Email: jfilippazzo@stsci.edu

class exoctk.lightcurve_fitting.lightcurve.**LightCurve**(*time*, *flux*, *unc=None*, *parameters=None*, *units='MJD'*, *name='My Light Curve'*)

Bases: exoctk.lightcurve_fitting.models.Model

A class to store the actual light curve

Parameters

time: sequence
The time axis in days, [MJD or BJD]

flux: sequence
The flux in electrons (not ADU)

unc: sequence
The uncertainty on the flux

parameters: str, object (optional)
The orbital parameters of the star/planet system, may be a path to a JSON file or a parameter object

units: str
The time units

name: str
A name for the object

fit(*model*, *fitter='lmfit'*, ***kwargs*)
Fit the model to the lightcurve

Parameters

model: ExoCTK.lightcurve_fitter.models.Model
The model to fit to the data

fitter: str
The name of the fitter to use

plot(*fits=True*, *draw=True*)
Plot the light curve with all available fits

Parameters

fits: bool
Plot the fit models

draw: bool
Show the figure, else return it

Returns

bokeh.plotting.figure
The figure

reset()
Reset the results

class exoctk.lightcurve_fitting.lightcurve.**LightCurveFitter**(*time*, *flux*, *model*)
Bases: object

Fit the model to the flux cube

Parameters

time:
1D or 2D time axes

flux:
2D flux

master_slicer(*value*, *param_name='wavelength'*)

run()
Run the model fits

4.4 exoctk.lightcurve_fitting.models module

Base and child classes to handle models used to fit light curves

Author: Joe Filippazzo Email: jfilippazzo@stsci.edu

class exoctk.lightcurve_fitting.models.**CompositeModel**(*models*, ***kwargs*)
Bases: exoctk.lightcurve_fitting.models.Model

A class to create composite models

Initialize the composite model

Parameters

models: sequence
The list of models

eval(***kwargs*)
Evaluate the model components

class exoctk.lightcurve_fitting.models.**Model**(***kwargs*)
Bases: object

Create a model instance

property flux
A getter for the flux

interp(*new_time*)
Interpolate the flux to a new time axis

Parameters

new_time: sequence, astropy.units.quantity.Quantity
The time array

property parameters
A getter for the parameters

plot(*time*, *components=False*, *fig=None*, *draw=False*, *color='blue'*, ***kwargs*)
Plot the model

Parameters

time: array-like
The time axis to use

components: bool
Plot all model components

fig: bokeh.plotting.figure (optional)
The figure to plot on

Returns

bokeh.plotting.figure
The figure

property time
A getter for the time

property units
A getter for the units

class exoctk.lightcurve_fitting.models.**PolynomialModel**(***kwargs*)
Bases: exoctk.lightcurve_fitting.models.Model

Polynomial Model

Initialize the polynomial model

eval(***kwargs*)
Evaluate the function with the given values

class exoctk.lightcurve_fitting.models.**TransitModel**(***kwargs*)
Bases: exoctk.lightcurve_fitting.models.Model

Transit Model

Initialize the transit model

eval(***kwargs*)
Evaluate the function with the given values

4.5 exoctk.lightcurve_fitting.parameters module

Base and child classes to handle orbital parameters

Author: Joe Filippazzo Email: jfilippazzo@stsci.edu

class exoctk.lightcurve_fitting.parameters.**Parameter**(*name*, *value*, *ptype='free'*, *mn=None*, *mx=None*)
Bases: object

A generic parameter class

Instantiate a Parameter with a name and value at least

Parameters

name: str
The name of the parameter

value: float, int, str, list, tuple
The value of the parameter

ptype: str
Parameter type, ['free','fixed','independent']

mn: float, int, str, list, tuple (optioal)
The minimum value

mx: float, int, str, list, tuple (optioal)
The maximim value

`property ptype`
Getter for the ptype

`property values`
Return all values for this parameter

`class exoctk.lightcurve_fitting.parameters.Parameters(param_file=None, **kwargs)`
Bases: `object`

A class to hold the Parameter instances

Initialize the parameter object

Parameters

param_file: str
A text file of the parameters to parse

4.6 Module contents

Package to fit models to light curve data

Limb Darkening Calculator

The limb_darkening tool calculates limb-darkening coefficients for a specified stellar model, plotting results versus μ and wavelength. It uses high spectral resolution stellar atmospheric models, which are a necessity given JWST's expected precision.

CHAPTER

FIVE

CALCULATE LIMB DARKENING COEFFICIENTS

To calculate the limb darkening coefficients, we need a model grid.

In our first example, we use the Phoenix ACES models but any grid can be loaded into a modelgrid.ModelGrid() object if the spectra are stored as FITS files.

Two model grids are available in the EXOCTK_DATA directory and have corresponding child classes for convenience. The Phoenix ACES models and the Kurucz ATLAS9 models can be loaded with the modelgrid.ACES() and modelgrid.ATLAS9() classes respectively.

We can also use the resolution argument to resample the model spectra. This greatly speeds up the caluclations.

```
model_grid = modelgrid.ACES(resolution=700)
print(model_grid.data)
```

``1056 models loaded from /Users/jfilippazzo/Documents/STScI/ExoCTK/EXOCTK_DATA/modelgrid/ACES/

Teff logg … filename

—— —- … ———————————————————- 5800.0 3.0 … lte05800-3.00-0.0.PHOENIX-ACES-AGSS-COND-SPECINT-2011.fits 7600.0 5.0 … lte07600-5.00+0.5.PHOENIX-ACES-AGSS-COND-SPECINT-2011.fits 4100.0 5.0 … lte04100-5.00-0.0.PHOENIX-ACES-AGSS-COND-SPECINT-2011.fits 6900.0 4.0 … lte06900-4.00-0.5.PHOENIX-ACES-AGSS-COND-SPECINT-2011.fits 4400.0 3.0 … lte04400-3.00+0.5.PHOENIX-ACES-AGSS-COND-SPECINT-2011.fits 2300.0 5.0 … lte02300-5.00-0.5.PHOENIX-ACES-AGSS-COND-SPECINT-2011.fits 4200.0 4.0 … lte04200-4.00-0.5.PHOENIX-ACES-AGSS-COND-SPECINT-2011.fits 4700.0 5.0 … lte04700-5.00-0.0.PHOENIX-ACES-AGSS-COND-SPECINT-2011.fits 6900.0 3.0 … lte06900-3.00+0.5.PHOENIX-ACES-AGSS-COND-SPECINT-2011.fits 2600.0 4.0 … lte02600-4.00-0.0.PHOENIX-ACES-AGSS-COND-SPECINT-2011.fits

… … … …

5200.0 5.0 … lte05200-5.00-0.5.PHOENIX-ACES-AGSS-COND-SPECINT-2011.fits 3500.0 3.0 … lte03500-3.00+0.5.PHOENIX-ACES-AGSS-COND-SPECINT-2011.fits 3300.0 4.0 … lte03300-4.00-0.5.PHOENIX-ACES-AGSS-COND-SPECINT-2011.fits 5100.0 4.0 … lte05100-4.00-0.0.PHOENIX-ACES-AGSS-COND-SPECINT-2011.fits 5400.0 5.0 … lte05400-5.00-0.5.PHOENIX-ACES-AGSS-COND-SPECINT-2011.fits 3300.0 3.0 … lte03300-3.00+0.5.PHOENIX-ACES-AGSS-COND-SPECINT-2011.fits 3500.0 4.0 … lte03500-4.00-0.5.PHOENIX-ACES-AGSS-COND-SPECINT-2011.fits 3600.0 5.0 … lte03600-5.00-0.0.PHOENIX-ACES-AGSS-COND-SPECINT-2011.fits 5700.0 4.0 … lte05700-4.00-0.0.PHOENIX-ACES-AGSS-COND-SPECINT-2011.fits 6600.0 3.0 … lte06600-3.00-0.5.PHOENIX-ACES-AGSS-COND-SPECINT-2011.fits 6000.0 4.0 … lte06000-4.00+0.5.PHOENIX-ACES-AGSS-COND-SPECINT-2011.fits Length = 1056 rows``

Now let's customize it to our desired effective temperature, surface gravity, metallicity, and wavelength ranges by running the customize() method on our grid.

```
model_grid.customize(Teff_rng=(2500,2600), logg_rng=(5,5.5), FeH_rng=(-0.5,0.5))
```

```
12/1056 spectra in parameter range Teff: (2500, 2600) , logg: (5, 5.5) , FeH: (-0.5, 0.5) , wavelength: (<Quantity 0. um>, <Quantity 40. um>) Loading flux into table... 100.00 percent complete!
```

Now we can caluclate the limb darkening coefficients using the limb_darkening_fit.LDC() class.

```
ld = lf.LDC(model_grid)
```

We just need to specify the desired effective temperature, surface gravity, metallicity, and the function(s) to fit to the limb darkening profile (including 'uniform', 'linear', 'quadratic', 'square-root', 'logarithmic', 'exponential', and 'nonlinear').

We can do this with for a single model on the grid:

..code-block:: python

teff, logg, FeH = 2500, 5, 0 ld.calculate(teff, logg, FeH, 'quadratic', name='on-grid', color='blue')

```
Bandpass trimmed to 0.05 um - 2.5999 um 1 bins of 100 pixels each.
```

Or a single model off the grid, where the spectral intensity model is directly interpolated before the limb darkening coefficients are calculated. This takes a few seconds when calculating:

..code-block:: python

teff, logg, FeH = 2511, 5.22, 0.04 ld.calculate(teff, logg, FeH, 'quadratic', name='off-grid', color='red')

`` Interpolating grid point [2511/5.22/0.04]... Run time in seconds: 11.166051149368286 Bandpass trimmed to 0.05 um - 2.5999 um 1 bins of 100 pixels each.``

Now we can see the results table using the following command:

..code-block:: python

print(ld.results)

`` name Teff logg FeH profile filter ... color c1 e1 c2 e2 ——— —— —- —- ——— ——- ... —— —— —— —— ——

on-grid 2500.0 5.0 0.0 quadratic Top Hat ... blue 0.218 0.024 0.391 0.033

off-grid 2511.0 5.22 0.04 quadratic Top Hat ... red 0.224 0.025 0.398 0.033``

5.1 Using a Photometric Bandpass

Above we caluclated the limb darkening in a particular wavelength range set when we ran the `customize()` method on our `core.ModelGrid()` object.

Additionally, we can calculate the limb darkening through a particular photometric bandpass.

First we have to create a `svo_filters.svo.Filter()` object which we can then pass to the calculate method. Let's use 2MASS H-band for this example.

..code-block:: python

H_band = svo.Filter('2MASS.H') H_band.plot()

Now we can tell `LDC.calculate()` to apply the filter to the spectral intensity models before calculating the limb darkening coefficients using the bandpass argument. We'll compare the results of using the bandpass (purple line) to the results where we just used the wavelength window of 1.4-1.9 mu$m$ (green line).

..code-block:: python

ld = lf.LDC(model_grid) teff, logg, FeH = 2511, 5.22, 0.04 ld.calculate(teff, logg, FeH, '4-parameter', name='Top Hat', color='green') ld.calculate(teff, logg, FeH, '4-parameter', bandpass=H_band, name='H band', color='purple') ld.plot(show=True)

```
Interpolating grid point [2511/5.22/0.04]... Run time in seconds:  12.76802396774292 Bandpass
trimmed to 0.05 um - 2.5999 um 1 bins of 100 pixels each. Interpolating grid point [2511/5.22/
0.04]... Run time in seconds:  12.711306095123291
```

5.2 Using a Grism

Grisms are also supported. We can use the whole grism wavelength range (as if it was a bandpass) or truncate the grism to consider arbitrary wavelength ranges by setting the `wave_min` and `wave_max` arguments.

..code-block:: python

G141 = svo.Filter('WFC3_IR.G141', wave_min=1.61*q.um, wave_max=1.65*q.um) G141.plot()

```
Bandpass trimmed to 1.11 um - 1.65 um 15 bins of 431 pixels each.
```

Now we can caluclate the LDCs for each of the 15 wavelength bins of the G141 grism we just created, where the first column in the table is the bin center. This is not very useful to plot but... why not?

..code-block:: python

teff, logg, FeH = 2511, 5.22, 0.04 ld.calculate(teff, logg, FeH, '4-parameter', bandpass=G141) print(ld.results)

``Interpolating grid point [2511/5.22/0.04]... Run time in seconds: 12.591181993484497

name Teff logg FeH profile ... e2 c3 e3 c4 e4

——— —— —— —— ——— ... —— —— —— —— —— 1.12 um 2511.0 5.22 0.04 4-parameter ... 0.011 -0.599 0.011 0.193 0.004 1.15 um 2511.0 5.22 0.04 4-parameter ... 0.016 0.454 0.017 -0.071 0.006 1.19 um 2511.0 5.22 0.04 4-parameter ... 0.01 0.458 0.011 -0.086 0.004 1.22 um 2511.0 5.22 0.04 4-parameter ... 0.01 0.7 0.011 -0.168 0.004 1.25 um 2511.0 5.22 0.04 4-parameter ... 0.008 0.321 0.009 -0.052 0.003 1.28 um 2511.0 5.22 0.04 4-parameter ... 0.019 0.832 0.02 -0.213 0.008 1.32 um 2511.0 5.22 0.04 4-parameter ... 0.006 0.766 0.007 -0.179 0.003 1.35 um 2511.0 5.22 0.04 4-parameter ... 0.024 -0.365 0.026 0.138 0.01 1.39 um 2511.0 5.22 0.04 4-parameter ... 0.048 -1.159 0.051 0.379 0.019 1.43 um 2511.0 5.22 0.04 4-parameter ... 0.012 -0.775 0.013 0.209 0.005 1.46 um 2511.0 5.22 0.04 4-parameter ... 0.033 -0.893 0.035 0.273 0.013

1.5 um 2511.0 5.22 0.04 4-parameter ... 0.037 -0.776 0.039 0.26 0.015

1.54 um 2511.0 5.22 0.04 4-parameter ... 0.05 -0.623 0.053 0.235 0.02 1.59 um 2511.0 5.22 0.04 4-parameter ... 0.01 0.308 0.011 -0.049 0.004 1.63 um 2511.0 5.22 0.04 4-parameter ... 0.005 0.57 0.005 -0.131 0.002``

CHAPTER

SIX

EXOCTK.LIMB_DARKENING PACKAGE

6.1 Submodules

6.2 exoctk.limb_darkening.limb_darkening_fit module

A module to calculate limb darkening coefficients from a grid of model spectra

class exoctk.limb_darkening.limb_darkening_fit.**LDC**(*model_grid='ACES'*)

Bases: object

A class to hold all the LDCs you want to run

Initialize an LDC object

Parameters

model_grid: exoctk.modelgrid.ModelGrid
The grid of synthetic spectra from which the coefficients will be calculated

static bootstrap_errors(*mu_vals*, *func*, *coeffs*, *errors*, *n_samples=1000*)

Bootstrapping LDC errors

Parameters

mu_vals: sequence
The mu values

func: callable
The LD profile function

coeffs: sequence
The coefficients

errors: sequence
The errors on each coeff

n_samples: int
The number of samples

Returns

tuple
The lower and upper errors

calculate(*Teff*, *logg*, *FeH*, *profile*, *mu_min=0.05*, *ld_min=0.01*, *bandpass=None*, *name=None*, *color=None*, ***kwargs*)

Calculates the limb darkening coefficients for a given synthetic spectrum. If the model grid does not contain a spectrum of the given parameters, the grid is interpolated to those parameters.

Reference for limb-darkening laws: http://www.astro.ex.ac.uk/people/sing/David_Sing/Limb_Darkening.html

Parameters

Teff: int
The effective temperature of the model

logg: float
The logarithm of the surface gravity

FeH: float
The logarithm of the metallicity

profile: str
The name of the limb darkening profile function to use, including 'uniform', 'linear', 'quadratic', 'square-root', 'logarithmic', 'exponential', and '4-parameter'

mu_min: float
The minimum mu value to consider

ld_min: float
The minimum limb darkening value to consider

bandpass: svo_filters.svo.Filter() (optional)
The photometric filter through which the limb darkening is to be calculated

name: str (optional)
A name for the calculation

color: str (optional)
A color for the plotted result

plot(*fig=None*, *show=False*, ***kwargs*)

Plot the LDCs

Parameters

fig: matplotlib.pyplot.figure, bokeh.plotting.figure (optional)
An existing figure to plot on

show: bool
Show the figure

plot_tabs(*show=False*, ***kwargs*)

Plot the LDCs in a tabbed figure

Parameters

fig: matplotlib.pyplot.figure, bokeh.plotting.figure (optional)
An existing figure to plot on

show: bool
Show the figure

save(*filepath*)
Save the LDC results to file

Parameters

filepath: str
The complete filepath to save the results to

exoctk.limb_darkening.limb_darkening_fit.**ld_profile**(*name='quadratic'*, *latex=False*)
Define the function to fit the limb darkening profile

Reference:
https://www.cfa.harvard.edu/~lkreidberg/batman/ tutorial.html#limb-darkening-options

Parameters

name: str
The name of the limb darkening profile function to use, including 'uniform', 'linear', 'quadratic', 'square-root', 'logarithmic', 'exponential', '3-parameter', and '4-parameter'

latex: bool
Return the function as a LaTeX formatted string

Returns

function, str
The corresponding function for the given profile

6.3 Module contents

Package to generate limb darkening coefficients from a grid of model spectra

Atmospheric Retrievals

The `atmospheric_retrievals` subpackage within the `exoctk` package currently contains a module for performing retrievals via the PLATON package. This Jupyter notebook contains a demo of how to use the platon_wrapper module.

Users who wish to use the `atmospheric_retrievals` tools may do so by installing the `exoctk` package. Please see the installation instructions for further details.

CHAPTER

SEVEN

ATMOSPHERIC RETRIEVALS

Contents

Below, we demonstrate the use of ExoCTK's `platon_wrapper` module. As suggested by its name, this module is a wrapper around the `platon.retreiver.run_multinest` and `platon.retriever.run_emcee` methods, which uses multinested sampling and MCMC algorithms to retrieve atmospheric parameters, respectively. For further information about `platon`, see the project documentation, the API docs for the retriever module, or the GitHub repository.

Note that some of the examples provided below are minimal, bare-bones examples that are meant to show how users may use the software while not taking much computation time to complete. The parameters used and the corresponding results are not indiative of a true scientific use case. For more comprehensive and robust examples, see the examples.py module in the `exoctk.atmospheric_retrievals` subpackage.

The first section of this page provides a simple example of how to run the software on a local machine. The later sections describe how to perform retrievals using Amazon Web Services (AWS) Elastic Computing (EC2) instances.

This notebook assumes that the user has installed the `exoctk` package and its required libraries. For more information about the installation of `exoctk`, see the installation instructions.

7.1 A Simple Example

Below is a simple example of how to use the `PlatonWrapper` object to perform atmospheric retrievals. First, a few necessary imports:

```
import numpy as np
from platon.constants import R_sun, R_jup, M_jup
from exoctk.atmospheric_retrievals.platon_wrapper import PlatonWrapper
```

The `PlatonWrapper` object requires the user to supply a dictionary containing initial guesses of parameters that they wish to fit. Note that `Rs`, `Mp`, `Rp`, and `T` must be supplied, while the others are optional.

Also note that `Rs` are in units of solar radii, `Mp` are in units of Jupiter masses, and `Rp` is is units of Jupiter radii.

```
params = {
    'Rs': 1.19,  # Required
    'Mp': 0.73,  # Required
    'Rp': 1.4,  # Required
    'T': 1200.0,  # Required
    'logZ': 0,  # Optional
    'CO_ratio': 0.53,  # Optional
    'log_cloudtop_P': 4,  # Optional
    'log_scatt_factor': 0,  # Optional
    'scatt_slope': 4,  # Optional
    'error_multiple': 1,  # Optional
    'T_star': 6091}  # Optional
```

In order to perform the retrieval, users must instantiate a `PlatonWrapper` object and set the parameters.

```
pw = PlatonWrapper()
pw.set_parameters(params)
```

Users may define fitting priors via the `fit_info` attribute.

```
pw.fit_info.add_gaussian_fit_param('Mp', 0.04*M_jup)
pw.fit_info.add_uniform_fit_param('Rp', 0.9*(1.4 * R_jup), 1.1*(1.4 * R_jup))
pw.fit_info.add_uniform_fit_param('T', 300, 3000)
pw.fit_info.add_uniform_fit_param("logZ", -1, 3)
pw.fit_info.add_uniform_fit_param("log_cloudtop_P", -0.99, 5)
```

Prior to performing the retrieval, users must define `bins`, `depths`, and `errors` attributes. The `bins` atribute must be a list of lists, with each element being the lower and upper bounds of the wavelenth bin. The `depths` and `errors` attributes are both 1-dimensional `numpy` arrays.

```
wavelengths = 1e-6*np.array([1.119, 1.1387])
pw.bins = [[w-0.0095e-6, w+0.0095e-6] for w in wavelengths]
pw.depths = 1e-6 * np.array([14512.7, 14546.5])
pw.errors = 1e-6 * np.array([50.6, 35.5])
```

With everything defined, users can now perform the retrieval. Users may choose to use the the MCMC method (`emcee`) or the Multinested Sampling method (`multinest`).

MCMC Method

```
pw.retrieve('emcee')
pw.save_results()
pw.make_plot()
```

Multinested Sampling Method

```
pw.retrieve('multinest')
pw.save_results()
pw.make_plot()
```

Note that results are saved in a text file named `<method>_results.dat`, a corner plot is saved to `<method>_corner.png`, and a log file describing the execution of the software is saved to `YYYY-MM-DD-HH-MM.log`, which is a timestamp reflecting the creation time of the log file.

7.2 Using Amazon Web Services to Perform Atmospheric Retrievals

The following sections guide users on how to perform atmospheric retrievals using Amazon Web Services (AWS) Elastic Computing (EC2) instances.

7.2.1 What is Amazon Web Services?

Amazon Web Services provides on-demand cloud-based computing platforms, with a variety of services such as Elastic Compute Cloud (EC2), Cloud Storage (S3), Relational Database Service (RDS), and more. Learn more at https://aws.amazon.com/what-is-aws/

7.2.2 What is an EC2 Instance?

The Elastic Compute Cloud (EC2) service enables users to spin up virtual servers, with a variety of operating systems, storage space, memory, processors. Learn more at https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html

7.2.3 Why use AWS?

Atmospheric retrievals are often computationally expensive, both in the amount of time it takes to complete a retrieval, but also in the cost of purchasing and/or maintaining a suitable machine. Particularly, if users do not have access to a dedicated science machine or cluster, and instead must rely on personal laptops or desktops, atmospheric retrievals can become quite burdensome in day-to-day research work.

AWS provides a means to outsource this computational effort to machines that live in the cloud, and for low costs. With AWS, users can create a virtual machine (VM), perform atmospheric retrievals, and have the machine automatically shutdown upon completion. Depending on the type of VM, typical costs can range from anywhere between ~\$0.02/hour (i.e. a small, 1 CPU Linux machine) to ~\$3.00/hour (i.e. a heftier, multiple CPU, GPU-enabled Linux machine).

For example, a small trial run of an atmospheric retrieval for hd209458b using PLATON takes roughly 35 minutes at a total cost of \$0.01 using a small CPU EC2 instance, and took roughly 24 minutes at a total cost of \$1.22 using a GPU-enabled EC2 instance.

7.2.4 AWS-specific software in the `atmospheric_retrievals` subpackage

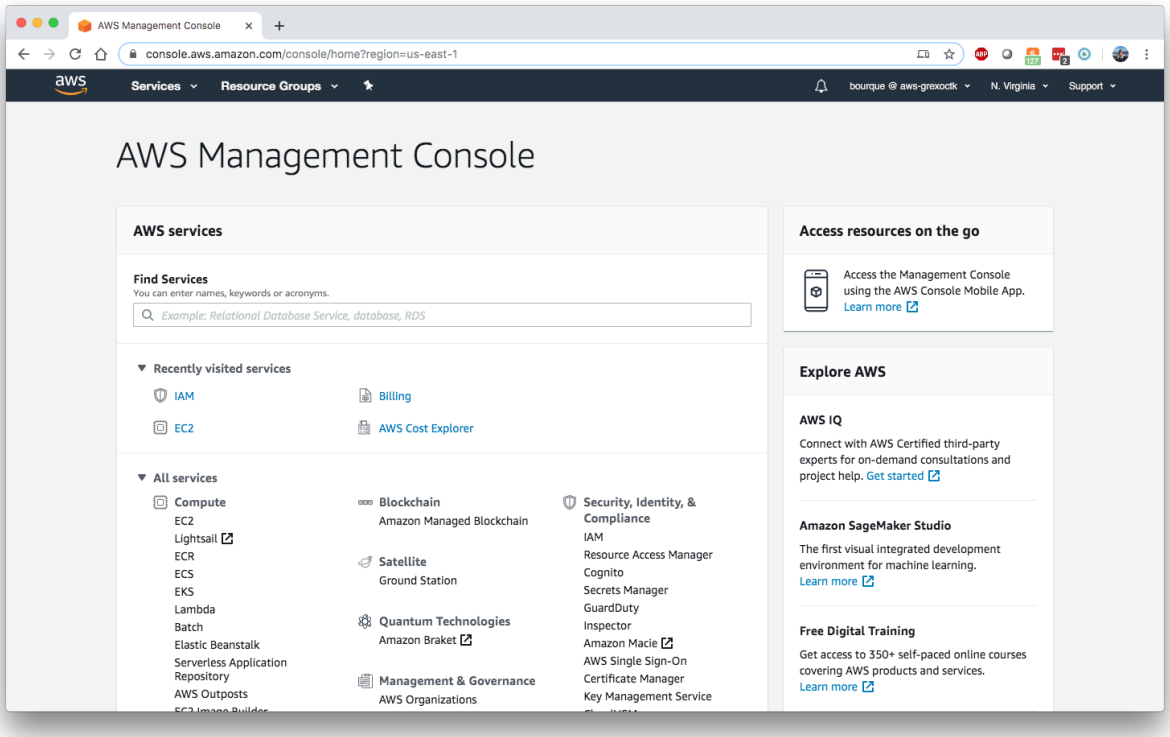
The `atmospheric_retrievals` subpackage provides software that enables users to use AWS EC2 instances to perform atmospheric retrievals. The relevant software modules/tools are:

- `aws_config.json` - a configuration file that contains a path the a public ssh key and a pointer to a particular EC2 instance
- `aws_tools.py` - Various functions to support AWS EC2 interactivity, such as starting/stopping an EC2 instance, transferring files to/from EC2 instances, and logging standard output from EC2 instances
- `build-exoctk-env-cpu.sh` - A bash script for creating an `exoctk` software environment on an EC2 instance
- `build-exoctk-env-gpu.sh` - A bash script for creating an `exoctk` software environment on a GPU-enabled EC2 instance
- `exoctk-env-init.sh` - A bash script that initializes an existing `exoctk` software environment on an EC2 instance

7.2.5 Setting up an AWS account

Users must first set up an AWS account and configure a ssh key pair in order to connect to the services.

1. Visit https://aws.amazon.com to create an account. Unfortunately, a credit card is required for sign up. There is no immediate fee for signing up; users will only incur costs when a service is used.
2. Once an account has been created, sign into the AWS console. Users should see a screen similar to this:



3. At the top of the page, under “Services”, select “IAM” to access the Identity and Access Management console.
4. On the left side of the page, select “Users”

5. Click the "Add user" button to create a new user. In the "User name" field, enter the username used for the AWS account. Select "Programmatic access" for the "Access type" option. Click on "Next: Permissions".
6. Select "Add user to group", and click "Create group". A "Create group" pane will open. In the "Group name" field, enter "admin". Check the box next to the first option, "AdministratorAccess", and click "Create group".
7. Click "Next: Tags". This step is optional, so users may then click "Next: Review", then "Create user".
8. When the user is created, users will be presented with a "Access Key ID" and "Secret Access Key". Take note of these, or download them to a csv file, as they will be used in the next step.
9. In a terminal, type `aws configure`. Users will be prompted to enter their Access Key ID and the Secret Access Key from the previous step. Also provide a Default region name (e.g. `us-east-1`, `us-west-1`, etc.) and for "output format" use `json`. For a list of available region names, see https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Concepts.RegionsAndAvailabilityZones.html
10. Executing these commands should result in the creation of a `aws/` directory, containing `config` and `credentials` files populated with the information that was provided.

7.2.6 Create a SSH key pair

In order to connect to an EC2 instance, users must next configure an SSH key pair:

1. In a terminal, type `ssh-keygen -t rsa -f`, where `<rsa_key_name>` is the name of the resulting ssh key files (users can name this whatever they would like). When prompted to enter a passphrase, leave it empty by hitting `enter`, and then `enter` again. Running this command should result in the creation of two files: (1) `<rsa_key_name>`, which is the private SSH key, and `<rsa_key_name>`.pub, which is the public SSH key.
2. In the browser, navigate to the AWS EC2 console (https://console.aws.amazon.com/ec2), select `Key Pairs` under `Network & Security` on the left hand side of the page.
3. Select `Import key pair`
4. In the `Name` field, enter a name you wish to use.
5. In the large field on the bottom, paste the contents of the `<rsa_key_name>`.pub file.
6. Select `Import key pair` to complete the process.

7.2.7 Launch an EC2 instance

To create and launch an EC2 instance:

1. Select "Instances" from the left-hand side of the AWS EC2 console
2. Select the "Launch Instance" button
3. Select an Amazon Machine Image (AMI) of your choosing. Note that there is a box on the left that allows users to only show free tier only eligible AMIs. For the purposes of the examples in this notebook, it is suggested to use `ami-0c322300a1dd5dc79` (Red Hat Enterprise Linux 8 (HVM), SSD Volume Type, 64-bit (x86)).
4. Select the Instance Type with the configuration of your choosing. For the purposes of the examples in this notebook, it is suggested to use `t2.small`. When satisfied, choose "Review and Launch"
5. On the "Review Instance Launch" page, users may review and/or change any settings prior to launching the EC2 instance. For the purposes of the examples in this notebook, it is suggested to "Edit storage" and increase the "Size" to 20 GiB to allow enough storage space to build the `exoctk` software environment.
6. When satisfied, click "Launch". The user will be prompted to select or create a key pair. Select the existing key pair that was created in the "Create a SSH key pair" section. Check the acknowledgement box, and select "Launch Instances"

7. If the EC2 instance was launched successfully, there will be a success message with a link to the newly-created EC2 instance.

Note: For users interested in using GPU-enabled EC2 instances, see the "Using a GPU-enabled EC2 instance" section at the end of this notebook. This warrants its own section because it requires a rather complex installation process.

7.2.8 Build the `exoctk` software environment on the EC2 instance

Once the newly-created EC2 instance has been in its "Running" state for a minute or two, users can log into the machine through the command line and install the necessary software dependencies needed for running the `atmospheric_retrievals` code.

To log into the EC2 instance from the command line, type:

```
ssh -i <path_to_private_key> ec2-user@<ec2_public_dns>
```

where `<path_to_private_key>` is the path to the private SSH key file (i.e. the `<rsa_key_name>` that was created in the "Create a SSH key file" section), and `<ec2_public_dns>` is the Public DNS of the EC2 instance, which is provided in the "Description" of the EC2 instance under the "Instances" panel in the AWS EC2 console. This public DNS should look something like `ec2-NN-NN-NNN-NN.compute-N.amazonaws.com`.

Users may be asked (yes/no) if they want to connect to the machine. Enter "yes".

Once logged in, users can build the `exoctk` software environment by either copy/pasting the commands from the `atmospheric_retrievals/build-exoctk-env-cpu.sh` file straight into the EC2 terminal, or by copying the `build-exoctk-env-cpu.sh` file directly to the EC2 instance and running it. To do the later option, from your local machine, type:

```
scp -i <path_to_private_key> build-exoctk-env-cpu.sh ec2-user@<ec2_public_dns>:/home/ec2-user/build-
↪exoctk-env-cpu.sh
ssh -i <path_to_private_key> ec2-user@<ec2_public_dns>
./build-exoctk-env-cpu.sh
```

Once completed, users may log out of the EC2 instance, as there will no longer be any command-line interaction needed.

7.2.9 Fill out the `aws_config.json` file

Within the `atmospheric_retrievals` subpackage, there exists an `aws_config.json` file. Fill in the values for the two fields: `ec2_id`, and `ssh_file`. The `ec2_id` should contain the name of EC2 template ID (which can be found under "Instance ID" in the description of the EC2 instance in the AWS EC2 console), and `ssh_file` should point to the location of the private SSH file described in the "Create a SSH key pair" section:

```
{
    "ec2_id" : "<ec2_instance_ID>",
    "ssh_file" : "<path_to_private_key>"
}
```

7.2.10 Run some code!

Now that we have configured everything to run on AWS, the next step is to simply perform a retrieval! Open a Python session or Jupyter notebook. To invoke the use of the AWS EC2 instance, simply use the `use_aws()` method before performing the retrieval. A short example is provided below.

```
import numpy as np
from platon.constants import R_sun, R_jup, M_jup
from exoctk.atmospheric_retrievals.aws_tools import get_config
from exoctk.atmospheric_retrievals.platon_wrapper import PlatonWrapper

params = {
    'Rs': 1.19,  # Required
    'Mp': 0.73,  # Required
    'Rp': 1.4,  # Required
    'T': 1200.0,  # Required
    'logZ': 0,  # Optional
    'CO_ratio': 0.53,  # Optional
    'log_cloudtop_P': 4,  # Optional
    'log_scatt_factor': 0,  # Optional
    'scatt_slope': 4,  # Optional
    'error_multiple': 1,  # Optional
    'T_star': 6091}  # Optional

pw = PlatonWrapper()
pw.set_parameters(params)

pw.fit_info.add_gaussian_fit_param('Mp', 0.04*M_jup)
pw.fit_info.add_uniform_fit_param('Rp', 0.9*(1.4 * R_jup), 1.1*(1.4 * R_jup))
pw.fit_info.add_uniform_fit_param('T', 300, 3000)
pw.fit_info.add_uniform_fit_param("logZ", -1, 3)
pw.fit_info.add_uniform_fit_param("log_cloudtop_P", -0.99, 5)

wavelengths = 1e-6*np.array([1.119, 1.1387])
pw.bins = [[w-0.0095e-6, w+0.0095e-6] for w in wavelengths]
pw.depths = 1e-6 * np.array([14512.7, 14546.5])
pw.errors = 1e-6 * np.array([50.6, 35.5])

ssh_file = get_config()['ssh_file']
ec2_id = get_config()['ec2_id']
pw.use_aws(ssh_file, ec2_id)

pw.retrieve('multinest')
pw.save_results()
pw.make_plot()
```

7.2.11 Output products

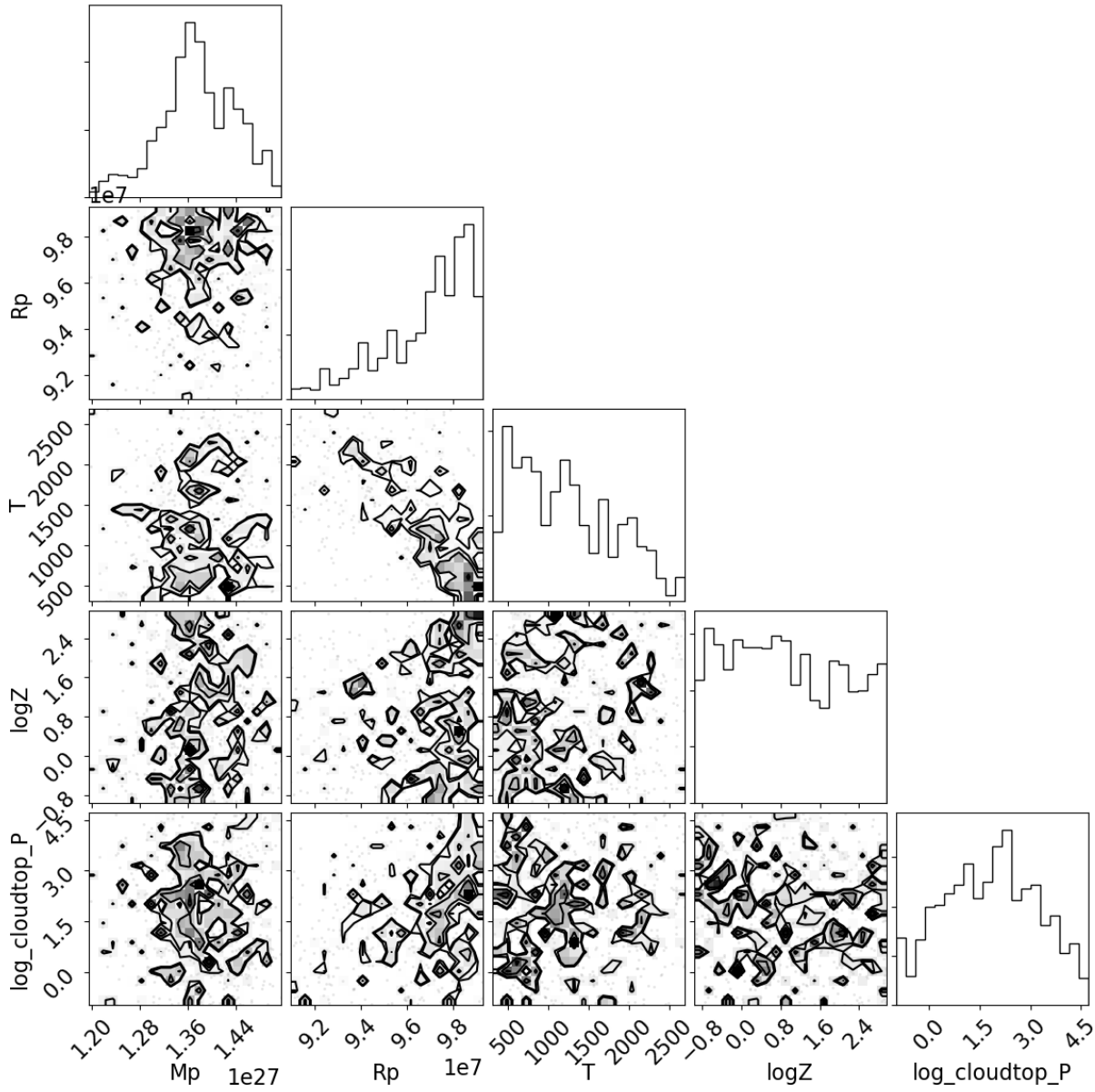
Executing the above code will result in a few output files:

- `YYYY-MM-DD-HH-MM.log` - A log file that captures information about the execution of the code, including software environment information, EC2 start/stop information, retrieval information and results, and total computation time.
- `multinest_results.dat/emcee_results.obj` - A data file containing the best fit results of the retrieval. Note that `emcee` results are saved as a Python object and saved to an object file.

- <method>_corner.png - A corner plot describing the quality of the best fit results of the retrieval, where <method> is the method used (i.e. multinest or emcee)

Here is an example of what these output products may look like:

Corner plot:



Results file:

Log file:

```
#Parameter Lower_error Median Upper_error Best_fit
Max_lnprob -42.226407335262884
Mp 5.625640866139536e+25 1.3721551087537816e+27 6.8701328789195775e+25 1.384787311926881e+27
Rp 2761008.9775913507 97274944.55719694 1380448.1193517298 98295437.66243619
T 606.854163819528 1154.7212486292049 776.7942785576104 945.046123723654
logZ 1.2285097671880447 0.8062276325837088 1.465410969404544 -0.5730007201094471
log_cloudtop_P 1.6469444397253428 1.9061989680281963 1.4065841099127987 3.564827608179593
```

```
12/16/2019 18:25:08 PM INFO: User: bourque
12/16/2019 18:25:08 PM INFO: Python Version: 3.6.7 |Anaconda, Inc.| (default, Oct 23 2018, 14:01:38) [GCC
4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)]
12/16/2019 18:25:08 PM INFO: Python Executable Path: /home/user/miniconda3/envs/exoctk-3.6/bin/python
12/16/2019 18:25:08 PM INFO: Setting parameters: {'Rs': 1.19, 'Mp': 0.73, 'Rp': 1.4, 'T': 1200.0, 'logZ': 0,
12/16/2019 18:26:28 PM INFO: Failed to import gnumpy; not using GPU
12/16/2019 18:26:28 PM INFO: Performing atmopsheric retrievals via multinest
12/16/2019 18:26:28 PM INFO: Iteration 1: ln_prob=-4.88e+04  Mp=0.76 M_jup  Rp=1.46 R_jup  T=2839 K
12/16/2019 18:26:28 PM INFO: Iteration 2: ln_prob=-1.41e+02  Mp=0.70 M_jup  Rp=1.32 R_jup  T=1614 K
12/16/2019 18:26:28 PM INFO: Iteration 3: ln_prob=-6.47e+02  Mp=0.69 M_jup  Rp=1.44 R_jup  T= 619 K
12/16/2019 18:26:28 PM INFO: Iteration 4: ln_prob=-5.51e+02  Mp=0.75 M_jup  Rp=1.33 R_jup  T=1585 K
12/16/2019 18:26:28 PM INFO: Iteration 5: ln_prob=-4.63e+02  Mp=0.69 M_jup  Rp=1.30 R_jup  T=1533 K
12/16/2019 18:26:28 PM INFO: Iteration 6: ln_prob=-2.21e+04  Mp=0.76 M_jup  Rp=1.52 R_jup  T=1999 K
12/16/2019 18:26:28 PM INFO: Iteration 7: ln_prob=-4.29e+01  Mp=0.73 M_jup  Rp=1.36 R_jup  T=1098 K
12/16/2019 18:26:28 PM INFO: Iteration 8: ln_prob=-1.78e+03  Mp=0.70 M_jup  Rp=1.45 R_jup  T=1057 K
12/16/2019 18:26:28 PM INFO: Iteration 9: ln_prob=-1.14e+03  Mp=0.81 M_jup  Rp=1.39 R_jup  T= 961 K
12/16/2019 18:26:28 PM INFO: Iteration 10: ln_prob=-6.31e+02  Mp=0.70 M_jup  Rp=1.32 R_jup  T=2641 K
12/16/2019 18:26:28 PM INFO: Iteration 11: ln_prob=-1.24e+02  Mp=0.77 M_jup  Rp=1.36 R_jup  T=1479 K
```

7.2.12 A final note: Make sure to terminate unused EC2 instances, or else Amazon will keep charging you!

Jeff Bezos does not need any more of your money. To terminate an EC2 instance, go to the EC2 console, then the "Instances" page. Select the instance of interest, click "Actions", "Instance State", then "Terminate".

7.2.13 Using a GPU-enabled EC2 instance

The `atmospheric_retrieval` subpackage supports the use of GPU-enabled EC2 instances. Users may create a GPU-enabled AMI/instance type configuration, such as AMI `ami-0c322300a1dd5dc79` with instance type `p3.2xlarge`, which contains multiple GPUs. However, the process for building a GPU-enabled `exoctk` software environment is more complex than that is described in the "Build the `exoctk` software environment on the ec2 instance" section, as it involves multiple machine reboots and thus cannot be easily installed by running a single bash script.

Below are instructions for installing/configuring the software environment needed for the GPU-enabled EC2 instance. These commands are also provided in the `atmospheric_retrievals/build-exoctk-env-gpu.sh` file.

1. First create a GPU-enabled EC2 instance, as described above and in the "Launch an EC2 instance" section.
2. Once the EC2 instance is created, navigate back to the AWS EC2 console and select "Instances" on the left side of the page to see a list of EC2 instances. Users should see the EC2 instance that was just created.
3. Allow the EC2 instance to enter the "Running" phase for a minute or two. Then, connect to the machine via the command line, as described in the "Build the `exoctk` software environment on the EC2 instance" section.
4. Once logged into the machine, users can now begin to run the necessary installation commands, provided below:

```
// Install NVIDIA GPU Driver
sudo yum -y update
sudo yum -y install wget nano elfutils-libelf-devel
sudo yum -y groupinstall "Development Tools"
sudo sed -i 's/crashkernel=auto"/crashkernel=auto nouveau.modeset=0"/g' /etc/default/grub
sudo grub2-mkconfig -o /boot/grub2/grub.cfg
sudo touch /etc/modprobe.d/blacklist.conf
```

(continues on next page)

(continued from previous page)

```
sudo chmod 777 /etc/modprobe.d/blacklist.conf
sudo echo 'blacklist nouveau' > /etc/modprobe.d/blacklist.conf
sudo mv /boot/initramfs-$(uname -r).img /boot/initramfs-$(uname -r)-nouveau.img
sudo dracut /boot/initramfs-$(uname -r).img $(uname -r)
sudo reboot
```

5. Allow EC2 to reboot, then when the EC2 is running again, log back into the instance.

```
sudo systemctl isolate multi-user.target
wget http://us.download.nvidia.com/XFree86/Linux-x86_64/430.40/NVIDIA-Linux-x86_64-430.40.run
sudo sh NVIDIA-Linux-x86_64-430.40.run
// Choose "No" when prompted to install 32-bit
sudo reboot
```

6. Again, allow EC2 to reboot, then when the EC2 is running again, log back into the instance.

```
// Install CUDA Toolkit
wget http://developer.download.nvidia.com/compute/cuda/10.1/Prod/local_installers/cuda_10.1.243_418.87.
↪00_linux.run
sudo sh cuda_10.1.243_418.87.00_linux.run
// Unselect
export PATH=$PATH:/usr/local/cuda-10.1/bin
export LD_LIBRARY_PATH=/usr/local/cuda-10.1/lib64

// Install Anaconda
curl -O https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
chmod 700 ./Miniconda3-latest-Linux-x86_64.sh
bash ./Miniconda3-latest-Linux-x86_64.sh -b -p $HOME/miniconda3

// Set important environment variables
export PATH=/home/ec2-user/miniconda3/bin:$PATH
export EXOCTK_DATA=''

// Create base CONDA
conda create --yes -n exoctk-3.6 python=3.6 git numpy flask pytest
conda init bash
source ~/.bashrc
conda activate exoctk-3.6

// Install ExoCTK package and conda environment
git clone https://github.com/ExoCTK/exoctk.git
cd exoctk/
conda env update -f env/environment-3.6.yml
conda init bash
source ~/.bashrc
conda activate exoctk-3.6
python setup.py develop
cd ../

// Install jwst_gtvt
rm -fr /home/ec2-user/miniconda3/envs/exoctk-3.6/lib/python3.6/site-packages/jwst_gtvt
git clone https://github.com/spacetelescope/jwst_gtvt.git
cd jwst_gtvt
git checkout cd6bc76f66f478eafbcc71834d3e735c73e03ed5
python setup.py develop
cd ../
```

(continues on next page)

(continued from previous page)

```
// Install additional libraries
pip install bibtexparser==1.1.0
pip install corner==2.0.1
pip install lmfit==0.9.13
pip install platon==3.1

// Install cudamat
git clone https://github.com/cudamat/cudamat.git
sudo sed -i "s/-O',/-O3',/g" /home/ec2-user/cudamat/setup.py
cd cudamat
python setup.py develop
cd ../

// Install gnumpy
git clone https://github.com/ExoCTK/gnumpy3.git
cd gnumpy3
python setup.py develop
cd ../

// Build skeleton EXOCTK_DATA directory
mkdir exoctk_data/
mkdir exoctk_data/exoctk_contam/
mkdir exoctk_data/exoctk_log/
mkdir exoctk_data/fortney/
mkdir exoctk_data/generic/
mkdir exoctk_data/groups_integrations/
mkdir exoctk_data/modelgrid/

// Print the complete environment
conda env export
```

Now that the EC2 instance is configured, select the EC2 instance in the AWS console and select "Actions > Stop" to stop the instance. Paste the EC2 instance ID into the `ec2_id` key in the `aws_config.json` file to point to the instance for processing.

At this point, users may now run code like the example in the "Run some code!" section, but now retrievals will be performed on the GPU-enabled EC2 instance.

CHAPTER

EIGHT

EXOCTK.ATMOSPHERIC_RETRIEVALS PACKAGE

8.1 Submodules

8.2 exoctk.atmospheric_retrievals.aws_tools module

This module contains various functions for interacting with AWS for `exoctk` atmospheric retrievals.

8.2.1 Authors

- Matthew Bourque

8.2.2 Use

This script is inteneded to be imported and used by other modules, for example:

from aws_tools import get_config get_config()

8.2.3 Dependencies

Dependent libraries include:

- boto3
- paramiko
- scp

Users must also have a `aws_config.json` file present within the `atmospheric_retrievals` subdirectory. This file must be of a valid JSON format and contain two key/value pairs, `ec2_id` and `ssh_file`, e.g.:

{ “ec2_id” : “lt-021de8b904bc2b728”, “ssh_file” : “~/.ssh/my_ssh_key.pem” }

where the `ec2_id` contains the ID for an EC2 launch template or an existing EC2 instance, and `ssh_file` points to the SSH public key used for logging into an AWS account.

`exoctk.atmospheric_retrievals.aws_tools.`**`build_environment`**(*instance*, *key*, *client*)

Builds an `exoctk` environment on the given AWS EC2 instance

Parameters

instance
[obj] A `boto3` AWS EC2 instance object.

key
[obj] A `paramiko.rsakey.RSAKey` object.

client
[obj] A `paramiko.client.SSHClient` object.

`exoctk.atmospheric_retrievals.aws_tools.`**`get_config`**`()`
Return a dictionary that holds the contents of the `aws_config.json` config file.

Returns

settings
[dict] A dictionary that holds the contents of the config file.

`exoctk.atmospheric_retrievals.aws_tools.`**`log_output`**`(`*`output`*`)`
Logs the given output of the EC2 instance.

Parameters

output
[str] The standard output of the EC2 instance

`exoctk.atmospheric_retrievals.aws_tools.`**`start_ec2`**`(`*`ssh_file`*`, `*`ec2_id`*`)`
Create a new EC2 instance or start an existing EC2 instance.

A new EC2 instance will be created if the supplied `ec2_id` is an EC2 template ID. An existing EC2 instance will be started if the supplied `ec2_id` is an ID for an existing EC2 instance.

Parameters

ssh_file
[str] Relative path to SSH public key to be used by AWS (e.g. `~/.ssh/exoctk.pem`).

ec2_id
[str] The AWS EC2 template id (e.g. `lt-021de8b904bc2b728`) or instance ID (e.g. `i-0d0c8ca4ab324b260`).

Returns

instance
[obj] A `boto3` AWS EC2 instance object.

key
[obj] A `paramiko.rsakey.RSAKey` object.

client
[obj] A `paramiko.client.SSHClient` object.

`exoctk.atmospheric_retrievals.aws_tools.`**`stop_ec2`**`(`*`ec2_id`*`, `*`instance`*`)`
Terminates or stops the given AWS EC2 instance.

The instance is terminated if the supplied `ec2_id` is a EC2 template ID. The instance is stopped if the supplied `ec2_id` is an ID for a particular EC2 instance.

Parameters

ec2_id
[str] The AWS EC2 template id (e.g. `lt-021de8b904bc2b728`) or instance ID (e.g. `i-0d0c8ca4ab324b260`).

instance
[obj] A `boto3` AWS EC2 instance object.

`exoctk.atmospheric_retrievals.aws_tools.`**`transfer_from_ec2`**(*instance*, *key*, *client*, *filename*)
Copy files from EC2 user back to the user

Parameters

instance
[obj] A `boto3` AWS EC2 instance object.

key
[obj] A `paramiko.rsakey.RSAKey` object.

client
[obj] A `paramiko.client.SSHClient` object.

filename
[str] The path to the file to transfer

`exoctk.atmospheric_retrievals.aws_tools.`**`transfer_to_ec2`**(*instance*, *key*, *client*, *filename*)
Copy parameter file from user to EC2 instance

Parameters

instance
[obj] A `boto3` AWS EC2 instance object.

key
[obj] A `paramiko.rsakey.RSAKey` object.

client
[obj] A `paramiko.client.SSHClient` object.

filename
[str] The path to the file to transfer

8.3 exoctk.atmospheric_retrievals.examples module

This module provides examples of how to use the `platon_wrapper` module in ExoCTK's `atmopsheric_retrieval` subpackage. Each example can be run using either the `multinest` method or the `emcee` method. See the docstring of each example function for further details.

8.3.1 Authors

- Matthew Bourque

8.3.2 Use

To run all examples, one can execute this script via the command line as such:

```
>>> python examples.py
```

To run individual examples, one can import the example functions and pass as a parameter which method to run. Available examples include:

from examples import example, example_aws_short, example_aws_long example('emcee') example('multinest') example_aws_short('emcee') example_aws_short('multinest') example_aws_long('emcee') example_aws_long('multinest')

8.3.3 Dependencies

Dependent libraries include:

- `exoctk`
- `numpy`
- `pandas`
- `platon`

To run the examples that use AWS, users must also have a `aws_config.json` file present within the `atmospheric_retrievals` subdirectory. This file must be of a valid JSON format and contain two key/value pairs, `ec2_id` and `ssh_file`, e.g.:

{ "ec2_id" : "lt-021de8b904bc2b728", "ssh_file" : "~/.ssh/my_ssh_key.pem" }

where the `ec2_id` contains the ID for an EC2 launch template or an existing EC2 instance, and `ssh_file` points to the SSH public key used for logging into an AWS account.

Note that if the `ec2_id` points to a launch template (i.e. the string starts with `lt-`), a new EC2 instance will be created and launched. However, if the `ec2_id` points to an existing EC2 instance (i.e. the string starts with `i-`), the existing EC2 instance will be started and used.

8.3.4 References

Example data was pulled from the "Transmission Spectra" tab from corresponding ExoMAST pages, available at https://exo.mast.stsci.edu/

`exoctk.atmospheric_retrievals.examples.`**`example`**(*method*)
Performs a short example run of the retrievals using local machine.

Parameters

method
[str] The method to use to perform the atmopsheric retrieval; can either be `multinest` or `emcee`

`exoctk.atmospheric_retrievals.examples.`**`example_aws_long`**(*method*)
Performs an longer example run of the retrievals using AWS.

Parameters

method
[str] The method to use to perform the atmopsheric retrieval; can either be `multinest` or `emcee`

`exoctk.atmospheric_retrievals.examples.`**`example_aws_short`**(*method*)
Performs an short example run of the retrievals using AWS.

Parameters

method
[str] The method to use to perform the atmopsheric retrieval; can either be `multinest` or `emcee`

`exoctk.atmospheric_retrievals.examples.`**`get_example_data`**(*object_name*)
Return `bins`, `depths`, and `errors` for the given `object_name`. Data is read in from a `csv` file with a filename corresponding to `object_name`.

Parameters

object_name
[str] The object of interest (e.g. `hd209458b`)

Returns

bins
[np.array] A 2xN `numpy` array of wavelength bins, of the form `[[wavelength_bin_min, wavelength_bin_max], ...]`

depths
[np.array] A 1D `numpy` array of depth values

errors: np.array
A 1D `numpy` array of depth error values.

8.4 exoctk.atmospheric_retrievals.platon_wrapper module

A wrapper around the `platon` atmospheric retrieval tool.

This module serves as a wrapper around the atmospheric retrieval software for `platon`. It provides methods for performing retrievals through multinested sampling and MCMC methods. For more information about `platon`, please see `https://platon.readthedocs.io`. For examples of how to use this software, see the `examples.py` module.

8.4.1 Authors

- Matthew Bourque

8.4.2 Use

Users can perform the atmospheric retrieval by instantiating a `PlatonWrapper` object and passing fit parameters within the python environment. An example of this is provided below. For more examples of how to use this software, including ways to use AWS for performing computations, see the `examples.py` module, or the `atmospheric_retrievals_demo.ipynb` notebook under the `exoctk/notebooks/` directory.

```
import numpy as np
from platon.constants import R_sun, R_jup, M_jup
from exoctk.atmospheric_retrievals.platon_wrapper import PlatonWrapper

# Build dictionary of parameters you wish to fit
params = {
    'Rs': 1.19,  # Required
    'Mp': 0.73,  # Required
    'Rp': 1.4,  # Required
    'T': 1200.0,  # Required
    'logZ': 0,  # Optional
    'CO_ratio': 0.53,  # Optional
    'log_cloudtop_P': 4,  # Optional
    'log_scatt_factor': 0,  # Optional
    'scatt_slope': 4,  # Optional
    'error_multiple': 1,  # Optional
    'T_star': 6091}  # Optional

# Initialize PlatonWrapper object and set the parameters
pw = PlatonWrapper()
pw.set_parameters(params)

# Add any additional fit parameters
R_guess = 1.4 * R_jup
T_guess = 1200
pw.fit_info.add_gaussian_fit_param('Rs', 0.02*R_sun)
pw.fit_info.add_gaussian_fit_param('Mp', 0.04*M_jup)
pw.fit_info.add_uniform_fit_param('Rp', 0.9*R_guess, 1.1*R_guess)
pw.fit_info.add_uniform_fit_param('T', 0.5*T_guess, 1.5*T_guess)
pw.fit_info.add_uniform_fit_param("log_scatt_factor", 0, 1)
pw.fit_info.add_uniform_fit_param("logZ", -1, 3)
pw.fit_info.add_uniform_fit_param("log_cloudtop_P", -0.99, 5)
pw.fit_info.add_uniform_fit_param("error_multiple", 0.5, 5)

# Define bins, depths, and errors
pw.wavelengths = 1e-6*np.array([1.119, 1.138, 1.157, 1.175, 1.194, 1.213, 1.232, 1.251, 1.
↪270, 1.288, 1.307, 1.326, 1.345, 1.364, 1.383, 1.401, 1.420, 1.439, 1.458, 1.477, 1.496, 1.
↪515, 1.533, 1.552, 1.571, 1.590, 1.609, 1.628])
pw.bins = [[w-0.0095e-6, w+0.0095e-6] for w in pw.wavelengths]
pw.depths = 1e-6 * np.array([14512.7, 14546.5, 14566.3, 14523.1, 14528.7, 14549.9, 14571.8,␣
↪14538.6, 14522.2, 14538.4, 14535.9, 14604.5, 14685.0, 14779.0, 14752.1, 14788.8, 14705.2,␣
↪14701.7, 14677.7, 14695.1, 14722.3, 14641.4, 14676.8, 14666.2, 14642.5, 14594.1, 14530.1,␣
↪14642.1])
pw.errors = 1e-6 * np.array([50.6, 35.5, 35.2, 34.6, 34.1, 33.7, 33.5, 33.6, 33.8, 33.7, 33.
↪4, 33.4, 33.5, 33.9, 34.4, 34.5, 34.7, 35.0, 35.4, 35.9, 36.4, 36.6, 37.1, 37.8, 38.6, 39.
↪2, 39.9, 40.8])

# Perform the retrieval by your favorite method
pw.retrieve('multinest')  # OR
```

(continues on next page)

(continued from previous page)

```
pw.retrieve_('emcee')

# Save the results to an output file
pw.save_results()

# Save a plot of the results
pw.make_plot()
```

8.4.3 Dependencies

- corner
- exoctk
- matplotlib
- platon

class exoctk.atmospheric_retrievals.platon_wrapper.**PlatonWrapper**

Bases: object

Class object for running the platon atmospheric retrieval software.

Initialize the class object.

make_plot()

Create a corner plot that shows the results of the retrieval.

retrieve(*method*)

Perform the atmopsheric retrieval via the given method

Parameters

method

[str] The method by which to perform atmospheric retrievals. Can either be emcee or multinest.

save_results()

Save the results of the retrieval to an output file.

set_parameters(*params*)

Set necessary parameters to perform the retrieval.

Required parameters include Rs, Mp, Rp, and T. Optional parameters include logZ, CO_ratio, log_cloudtop_P, log_scatt_factor, scatt_slope, error_multiple, and T_star.

Parameters

params

[str or dict] Either a path to a params file to use, or a dictionary of parameters and their values for running the software. See "Use" documentation for further details.

use_aws(*ssh_file*, *ec2_id*)

Sets appropriate parameters in order to perform processing using an AWS EC2 instance.

Parameters

ssh_file
[str] The path to a public SSH key used to connect to the EC2 instance.

ec2_id
[str] A template id that points to a pre-built EC2 instance.

8.5 Module contents

Phase Constraint Calculator

The Phase Constraint Calculator provides a simple interface for calculating JWST observation start windows in phase-space for both, transits and eclipse observations. This allows the user to quickly calculate minimum and maximum phase values that serve as inputs for the APT special requirements section when planning your observations.

CHAPTER

NINE

PHASE CONSTRAINT CALCULATOR TUTORIAL

Due to uncertainties on the exact scheduling time of JWST observations (due, e.g., to previous observations, unforeseen scenarios, etc.), it is recommended that users of the observatory consider some flexibility in the start of their observations to give leeway of an hour to the observatory scheduling system (although observers can choose to narrow this window, there is a penalty in the charged time to the program).

The time window can be defined in the time-domain in Astronomer's Proposal Tool (APT) under the APT Special Requirements, for periodic phenomena like transiting exoplanets this would be cumbersome to include as one would have to define a time-window for every possible transit/eclipse event on the current observing Cycle. Fortunately, APT also allows users to define this window in phase-space (i.e., in units of fractions of the orbital period), where the zero-phase can be arbitrarily defined.

The **ExoCTK's phase-constraint package** was developed in order to perform the calculations of these windows in phase-space for any transiting exoplanet out there in a quick-and-easy way. Therefore this package greatly simplifies the work of observation planning when it comes to transiting exoplanet observations.

9.1 Quick Start

Let's suppose we want to obtain the 1-hour-window in phase space to schedule an observation of the primary transit of WASP-18b. To obtain this with **ExoCTK's phase-constraint** package, one would simply do:

```
import exoctk.phase_constraint_overlap.phase_constraint_overlap as pc
min_phase, max_phase = pc.phase_overlap_constraint('WASP-18b', window_size = 1.)
```

Which would produce an output such as this:

```
Retrieved period is 0.94124. Retrieved t0 is 58374.669900000095. Retrieved transit/eclipse
duration is:  2.14368 hrs; implied pre mid-transit/eclipse on-target time:  2.89368 hrs.
Performing calculations with Period: 0.94124, t0: 58374.669900000095, ecc: None, omega: None
degs, inc: None degs. MINIMUM PHASE: 0.8276351762922669, MAXIMUM PHASE: 0.8719030215460457
```

Two lines. That's all it took! In addition to the phase-constraints (the minimum and maximum phases), the phase_overlap_constraint call also returns the parameters it used to calculate the phase-constraint, along with some ephemerides of the planet, e.g., the period $\mathcal{P} = 0.94124$ days and time-of-transit center $\sqcup_0 = 58374.6699$ in Modified Julian Date (MJD, i.e., $\mathcal{J}D - 2400000.5$). But how did this magic happen? What do these numbers actually mean? Keep reading to understand how the phase-constraint calculator actually works.

9.2 Primary Eclipses: Using the Phase-Constraint Calculator

In the example above, the phase-constraint calculator returned the minimum and maximum phases for the exoplanet under study given only the planet name and the size of the window we were aiming to.

9.2.1 How Did It Do That?

In the background, the phase-constraint package automatically queries the exoplanet properties from exo.MAST given only the planet's name. Using this, it retrieves the properties of interest (period, $\mathcal{P}$, and total transit duration, $\mathcal{T}_{14}$, in this case) and, by default, assumes the observer wants to start the observations at the very least a time:

$\mathcal{T}_{pre} = 0.75 + \text{MAX}(1, T_{14}/2) + T_{14}/2$ hours

prior to mid-transit in this case. This time, by the way, is not arbitrary. Overall, the recommended (e.g., see this JWST observation planning step-by-step tutorial) time to spend on a target for a transit/eclipse observation is the above time $\mathcal{T}_{pre}$ prior to the mid-transit time, and $\mathcal{T}_{post} = T_{14}/2 + MAX(1, T_{14}/2) + T_W$ hours post mid-transit where $\mathcal{T}_W$ is the phase-constraint window (one hour in our example above). Using the retrieved properties for WASP-18b shown above, we can understand how the calculation was done in the background. The transit duration is $\mathcal{T}_{14} = 2.14368$ hours; the period is $\mathcal{P} = 0.94124 = 22.58976$ hours. The time $\mathcal{T}_{pre}$ is, thus, $\mathcal{T}_{pre} \approx 2.89368$, which in phase-space units is

$\mathcal{T}_{pre}/P \approx 0.128097$.

APT assumes the transit event is always located at phase 1 (or zero, whichever is more comfortable). Thus:

Maximum phase = $\infty - T_{pre}/P \approx 0.871903$,

which is exactly the maximum phase retrieved by the calculation. The minimum phase is simply one hour earlier in phase space. This gives:

Minimum phase = $\infty - (T_{pre} + 1)/P \approx 0.827635$,

again, exactly the minimum phase quoted above.

9.2.2 Modifying Phase-Constraint Parameters

The phase-constraint calculator allows to ingest a number of variables into the calculation in order to give control to the user in terms of the calculations they want to make. For instance, the pre-transit duration discussed above, $\mathcal{T}_{pre}$, can be changed by the user. This is done using the `pretransit_duration` variable. Suppose we wanted to retrieve the phase-constraint that corresponds to a pre-transit duration of 4 hours instead. We can simply do:

```
minp, maxp = pc.phase_overlap_constraint('WASP-18b', window_size = 1., pretransit_duration = 4.)
```

```
Retrieved period is 0.94124. Retrieved t0 is 58374.669900000095. Performing calculations with
Period: 0.94124, t0: 58374.669900000095, ecc: None, omega: None degs, inc: None degs. MINIMUM
PHASE: 0.7786607737311064, MAXIMUM PHASE: 0.8229286189848852
```

Of course, that is not the only parameter we can change. In fact, every transit parameter of interest can be ingested to the phase_overlap_constraint function, in whose case the user-defined properties will override the exo.MAST ones. Let's use, for instance, the ephemerides found for WASP-18b by Shporer et al. (2019) - $\mathcal{P} = 0.941452419$, $\sqcup_0 = 2458002.354726$

```
minp, maxp = pc.phase_overlap_constraint('WASP-18b', window_size = 1., period = 0.941452419, t0 =␣
↪2458002.354726)
```

```
Retrieved transit/eclipse duration is: 2.14368 hrs; implied pre mid-transit/eclipse on-target
time: 2.89368 hrs. Performing calculations with Period: 0.941452419, t0: 2458002.354726, ecc:
None, omega: None degs, inc: None degs. MINIMUM PHASE: 0.8276740668009621, MAXIMUM PHASE:
0.8719319239435721
```

Note how they are only slightly differnt than the ones retrieved from exo.MAST! One important detail in the above calculation, is that the time-of-transit center is of no use in phase-space because, by definition, for APT this is at phase equals 1. This means one could put any place holder value for $\sqcup_0$, and the calculation would result in the exact same values:

```
minp, maxp = pc.phase_overlap_constraint('WASP-18b', window_size = 1., period = 0.941452419, t0 = -1)
```

```
Retrieved transit/eclipse duration is: 2.14368 hrs; implied pre mid-transit/eclipse on-target
time: 2.89368 hrs. Performing calculations with Period: 0.941452419, t0: -1, ecc: None, omega:
None degs, inc: None degs. MINIMUM PHASE: 0.8276740668009621, MAXIMUM PHASE: 0.8719319239435721
```

Why does the phase-constraint overlap receives the time-of-transit center at all in the calculation? This will become clearer in the next section.

9.3 Secondary Eclipses: Using the Phase-Constraint Calculator

9.3.1 Phase-Constraints for Secondary Eclipses

The ExoCTK phase-constraint calculator can also obtain phase-constraints for secondary eclipses. This is indicated by the secondary flag in the `phase_overlap_constraint` function, which by default is `False`. Setting it to `True` in the WASP-18b case gives:

```
minp, maxp = pc.phase_overlap_constraint('WASP-18b', window_size = 1., period = 0.941452419, secondary␣
↪= True)
```

```
Retrieved transit/eclipse duration is: 2.122865968966563 hrs; implied pre mid-transit/eclipse
on-target time: 2.872865968966563 hrs. Performing calculations with Period: 0.941452419, t0:
None, ecc: 0.01, omega: 257.27 degs, inc: 85.68 degs. MINIMUM PHASE: 0.3271883452721046,
MAXIMUM PHASE: 0.3714462024147147
```

Note that, given the small eccentricity and inclination of WASP-18b's orbit, in this case the maximum phase is almost equal to the value one would obtain assuming a circular orbit for this exoplanet, which would locate the maximum phase at $0.5 - (T_{pre})/P \approx 0.3719$ (i.e., with the secondary eclipse centered at phase 0.5). The difference is of seconds — likely not critical for most JWST observations.

One important detail to remember before moving on: when ingesting the phase-constraints given above on APT, remember that we are still defining the zero-phase to be at the time of primary transit. This means that the phases given above only make sense to target eclipses in your observations if your "Zero Phase" in APT is set to the time of primary transit. This just makes it easier for the user: no need to compute times of secondary eclipses! (this is done in the background by the package). If you still want to know the time of secondary eclipse for some reason, keep reading. We got you covered!

9.3.2 Finding Secondary Eclipse Times

To find the phase-constraints for secondary eclipses, in the background the ExoCTK phase-constraint package solves the proper minimization of the conjunction problem numerically (equation (5) in Winn 2010), and thus finds the time of secondary eclipse (in phase-space) to perform the calculation using the orbital elements retrieved from exo.MAST (for secondary eclipses, in addition to the period $\mathcal{P}$, you need the inclination, $\rangle$, the eccentricity, $\rceil$, and the argument of periastron passage, ω — all of which can also be user-defined). This gives another functionality to the package: a secondary eclipse time calculator.

To retrieve the time of secondary eclipse, you can use the `get_secondary_time` flag in the `phase_overlap_constraint` function which, in addition to the minimum and maximum phases, returns the time of secondary eclipse just after the time of primary transit. Let's try this out for WASP-18b again:

```
minp, maxp, tsec = pc.phase_overlap_constraint('WASP-18b', window_size = 1., secondary = True, get_
↪secondary_time = True)
```

```
Retrieved period is 0.94124. Retrieved t0 is 58374.669900000095. Retrieved transit/eclipse
duration is:  2.122865968966563 hrs; implied pre mid-transit/eclipse on-target time:  2.
872865968966563 hrs. Performing calculations with Period:  0.94124, t0:  58374.669900000095,
ecc:  0.01, omega:  257.27 degs, inc:  85.68 degs. MINIMUM PHASE: 0.32714966265626544, MAXIMUM
PHASE: 0.37141750791004413, TSEC: 58375.13919576395
```

```
print('Secondary eclipse time:',tsec)
```

```
Secondary eclipse time:  58375.13919576395
```

As can be seen, the secondary eclipse time matches beautifully with our expectations for a non-eccentric orbit, which would give a secondary eclipse time of $\sqcup_0 + P/2 \approx 58375.14052$ MJD — only a 5-second difference between the two results.

9.3.3 Exploring Challenges for Secondary-Eclipse Times: HD 80606b, GJ 436b and HAT-P-2b

In order to showcase the power of the ExoCTK phase-constraint tool for secondary eclipse times and phase-constraints, we present here the results using our tool for more challenging systems in terms of predicting the location of their secondary eclipses. In order to compare with the literature values, however, we will be computing the phases at which secondary eclipses occur and not the times. This makes it easier to compare across datasets obtained at different epochs.

We start with HD 80606b, which is know to be very eccentric ($\rceil = 0.93$). A quick hack, if one is aiming at calculating the phase at which secondary eclipses occur is to let `window_size = 0.` and `pretransit_duration = 0` (Of course, never input this in APT!). This will force the minimum and maximum phases to return the phase at which secondary eclipse occur (because one is forcing the window to be of zero width, and for the observations to start exactly at the time of secodary eclipse). Let's see how well our phase-constraint tool does in this challenguing system:

```
minp, maxp = pc.phase_overlap_constraint('HD80606 b', window_size = 0., pretransit_duration = 0.,␣
↪secondary = True)
```

```
Retrieved period is 111.4367. Retrieved t0 is 55210.14280000003. Performing calculations with
Period:  111.4367, t0:  55210.14280000003, ecc:  0.93, omega:  301.03 degs, inc:  89.29 degs.
MINIMUM PHASE: 0.9455607255787186, MAXIMUM PHASE: 0.9455607255787186
```

This matches pretty well with the phase at which secondary eclipse happens in the literature (0.947; Laughlin, et al 2009)! Note we are using more updated planetary parameters than the ones from Laughlin et al., 2009, which explains the slight discrepancy in phase-space.

Next, let's try GJ 436b — a mildly eccentic system ($\rceil = 0.138$):

```
minp, maxp = pc.phase_overlap_constraint('GJ 436b', window_size = 0., pretransit_duration = 0.,␣
↪secondary = True)
```

```
Retrieved period is 2.64388312. Retrieved t0 is 54864.5839999998. Performing calculations with
Period: 2.64388312, t0: 54864.5839999998, ecc: 0.13827, omega: 351.0 degs, inc: 86.774 degs.
MINIMUM PHASE: 0.5868253469349103, MAXIMUM PHASE: 0.5868253469349103
```

Woah! Excellent agreement with Stevenson et al. (2010), where the secondary eclipse phase is at $/.5868 + / - 0.0003$. Finally, let's give the tool a shot with HAT-P-2b ($\rceil = 0.517$):

```
minp, maxp = pc.phase_overlap_constraint('HAT-P-2b', window_size = 0., pretransit_duration = 0.,␣
↪secondary = True)
```

```
Retrieved period is 5.6335158. Retrieved t0 is 55288.349100000225. Performing calculations with
Period: 5.6335158, t0: 55288.349100000225, ecc: 0.5172, omega: 188.01 degs, inc: 86.16 degs.
MINIMUM PHASE: 0.1876234349401976, MAXIMUM PHASE: 0.1876234349401976
```

Once again: beautiful agreement with de Wit et al. (2017), where the secondary eclipse phase happens at 0.187.

CHAPTER

TEN

EXOCTK.PHASE_CONSTRAINT_OVERLAP PACKAGE

10.1 Submodules

10.2 exoctk.phase_constraint_overlap.phase_constraint_overlap module

Phase contraint overlap tool. This tool calculates the minimum and maximum phase of the primary or secondary transit (by default, primary) based on parameters provided by the user.

Authors:
Catherine Martlin, 2018 Mees Fix, 2018 Nestor Espinoza, 2020

Usage:
calculate_constraint <target_name> [–t0=<t0>] [–period=<p>] [–pre_duration=<pre_duration>] [–transit_duration=<trans_dur>] [–window_size=<win_size>] [–secondary] [–eccentricity=<ecc>] [–omega=<omega>] [–inclination=<inc>] [–winn_approx] [–get_secondary_time]

Arguments:
<target_name> Name of target

Options:
-h –help Show this screen. –version Show version. –t0=<t0> The starting time of the transit in BJD or HJD. Only useful if user wants to have the time-of-secondary eclipse returned. –period=<p> The period of the transit in days. –pre_duration=<pre_duration> The duration of observations *before* transit/eclipse mid-time in hours. –transit_duration=<trans_dur> The duration of the transit in hours. –window_size=<win_size> The window size of the transit in hours [default: 1.0] –secondary If active, calculate phases for secondary eclipses (user needs to supply eccentricity, omega and inclination). –eccentricity=<ecc> The eccentricity of the orbit (needed for secondary eclipse constraints). –omega=<omega> The argument of periastron passage (needed for secondary eclipse constraints). –inclination=<inc> The inclination of the orbit (needed for secondary eclipse constraints). –winn_approx If active, instead of running the whole Kepler equation calculation, time of secondary eclipse is calculated using eq. (6) in Winn (2010; https://arxiv.org/abs/1001.2010v5) –get_secondary_time If active, calculation also returns time-of-secondary eclipse. Needs t0 as input.

exoctk.phase_constraint_overlap.phase_constraint_overlap.**calculate_phase**(*period*, *pre_duration*, *window_size*, *t0=None*, *ecc=None*, *omega=None*, *inc=None*, *secondary=False*, *winn_approx=False*, *get_secondary_time=False*)

Function to calculate the min and max phase.

Parameters

period
[float] The period of the transit in days.

pre_duration
[float] The duration of observations *before* transit/eclipse mid-time in hours.

window_size
[float] The window size of transit in hours. Default is 1 hour.

t0
[float] The time of (primary) transit center (only needed if get_secondary_time is True).

ecc
[float] The eccentricity of the orbit (only needed for secondary eclipses).

omega
[float] The argument of periastron passage, in degrees (only needed for secondary eclipses).

inc
[float] The inclination of the orbit, in degrees (only needed for secondary eclipses).

secondary
[boolean] If True, calculation will be done for secondary eclipses.

winn_approx
[boolean] If True, secondary eclipse calculation will use the Winn (2010) approximation to estimate time of secondary eclipse — (only valid for not very eccentric and inclined orbits).

get_secondary_time
[boolean] If True, return time of secondary eclipse along with the phase constraints.

Returns

minphase
[float] The minimum phase constraint.

maxphase
[float] The maximum phase constraint.

exoctk.phase_constraint_overlap.phase_constraint_overlap.**calculate_pre_duration**(*transitDur*)
Function to calculate the pre-transit hours to be spent on target as recommended by the Tdwell equation:

0.75 + Max(1hr,T14/2) (before transit) + T14 + Max(1hr, T14/2) (after transit) + 1hr (timing window)

The output is, thus, 0.75 + Max(1hr,T14/2) (before transit) + T14/2.

Parameters

transitDur
[float] The duration of the transit/eclipse in hours.

Returns

pretransit_duration
[float] The duration of the observation prior to transit/eclipse mid-time in hours.

exoctk.phase_constraint_overlap.phase_constraint_overlap.**calculate_tsec**(*period*, *ecc*, *omega*, *inc*, *t0=None*, *tperi=None*, *winn_approximation=False*)

Function to calculate the time of secondary eclipse.

This uses Halley's method (Newton-Raphson, but using second derivatives) to first find the true anomaly (f) at which secondary eclipse occurs, then uses this to get the eccentric anomaly (E) at secondary eclipse, which gives the mean anomaly (M) at secondary eclipse using Kepler's equation. This finally leads to the time of secondary eclipse using the definition of the mean anomaly (M = n*(t - tau) — here tau is the time of pericenter passage, n = 2*pi/period the mean motion).

Time inputs can be either the time of periastron passage directly or the time of transit center. If the latter, the true anomaly for primary transit will be calculated using Halley's method as well, and this will be used to get the time of periastron passage.

Parameters

period
[float] The period of the transit in days.

ecc
[float] Eccentricity of the orbit

omega
[float] Argument of periastron passage (in radians)

inc
[string] Inclination of the orbit (in radians)

t0
[float] The transit time in BJD or HJD (will be used to get time of periastron passage).

tperi
[float] The time of periastron passage in BJD or HJD (needed if t0 is not supplied).

winn_approximation
[boolean] If True, the approximation in Winn (2010) is used — (only valid for not very eccentric and inclined orbits).

Returns

tsec
[float] The time of secondary eclipse

exoctk.phase_constraint_overlap.phase_constraint_overlap.**drsky**(*x*, *ecc*, *omega*, *inc*)

Function whose roots we wish to find to obtain time of secondary (and primary) eclipse(s)

When one takes the derivative of equation (5) in Winn (2010; https://arxiv.org/abs/1001.2010v5), and equates that to zero (to find the minimum/maximum of said function), one gets to an equation of the form g(x) = 0. This function (drsky) is g(x), where x is the true anomaly.

Parameters

x
[float] True anomaly

ecc
[float] Eccentricity of the orbit

omega
[float] Argument of periastron passage (in radians)

inc
[float] Inclination of the orbit (in radians)

Returns

drsky
[float] Function evaluated at x, ecc, omega, inc

exoctk.phase_constraint_overlap.phase_constraint_overlap.**drsky_2prime**(*x*, *ecc*, *omega*, *inc*)
Second derivative of function drsky. This is the second derivative with respect to f of the drsky function.

Parameters

x
[float] True anomaly

ecc
[float] Eccentricity of the orbit

omega
[float] Argument of periastron passage (in radians)

inc
[float] Inclination of the orbit (in radians)

Returns

drsky_2prime
[float] Function evaluated at x, ecc, omega, inc

exoctk.phase_constraint_overlap.phase_constraint_overlap.**drsky_prime**(*x*, *ecc*, *omega*, *inc*)
Derivative of function drsky. This is the first derivative with respect to f of the drsky function.

Parameters

x
[float] True anomaly

ecc
[float] Eccentricity of the orbit

omega
[float] Argument of periastron passage (in radians)

inc
[float] Inclination of the orbit (in radians)

Returns

drsky_prime
[float] Function evaluated at x, ecc, omega, inc

exoctk.phase_constraint_overlap.phase_constraint_overlap.**getE**(*f*, *ecc*)
Function that returns the eccentric anomaly

Note normally this is defined in terms of cosines (see, e.g., Section 2.4 in Murray and Dermott), but numerically this is troublesome because the arccosine doesn't handle negative numbers by definition (equation 2.43). That's why the arctan version is better as signs are preserved (derivation is also in the same section, equation 2.46).

Parameters

f
[float] True anomaly

ecc
[float] Eccentricity

Returns

E
[float] Eccentric anomaly

exoctk.phase_constraint_overlap.phase_constraint_overlap.**getLTT**(*a*, *c*, *ecc*, *omega*, *inc*, *f*)
Function that calculates the Light Travel Time (LTT) for eclipses and transit

This function returns the light travel time of an eclipse or transit given the orbital parameters, the semi-major axis of the orbit and the speed of light. Consistent units must be used for the latter two parameters. The returned time is in the units of time given by the speed of light input parameter.

Parameters

a
[float] Semi-major axis of the orbit. Can be in any unit, as long as it is consistent with c.

c
[float] Speed of light. Can be in any unit, as long as it is consistent with a. The time-unit for the speed of light will define the returned time-unit for the light-travel time.

ecc
[float] Eccentricity of the orbit.

omega
[float] Argument of periastron passage (in radians).

inc
[string] Inclination of the orbit (in radians)

f
[float] True anomaly at the time of transit and/or eclipse (in radians).

Returns

ltt
[float] The light travel time, defined here as the time it takes for a photon to go from the planet at transit/eclipse to the plane in the sky where the star is located.

exoctk.phase_constraint_overlap.phase_constraint_overlap.**getM**(*E*, *ecc*)
Function that returns the mean anomaly using Kepler's equation

Parameters

E
[float] Eccentric anomaly

ecc: float
Eccentricity

Returns

M
[float] Mean anomaly

exoctk.phase_constraint_overlap.phase_constraint_overlap.**phase_overlap_constraint**(*target_name*, *period=None*, *t0=None*, *pretransit_duration=None*, *transit_dur=None*, *window_size=None*, *secondary=False*, *ecc=None*, *omega=None*, *inc=None*, *winn_approx=False*, *get_secondary_time=False*)

The main function to calculate the phase overlap constraints. We will update to allow a user to just plug in the target_name and get the other variables.

Parameters

target_name
[string] The name of the target transiting planet.

period
[float] The period of the transit in days.

t0
[float] The transit mid-time in BJD or HJD (only useful if time-of-secondary eclipse wants to be returned).

pretransit_duration
[float] The duration of the observations *before* transit/eclipse in hours.

transit_dur
[float] The duration of the transit/eclipse in hours.

window_size
[float] The window size of transit in hours. Default is 1 hour.

secondary
[boolean] If True, phase constraint will be the one for the secondary eclipse. Default is primary (i.e., transits).

ecc
[float] Eccentricity of the orbit. Needed only if secondary is True.

omega
[float] Argument of periastron of the orbit in degrees. Needed only if secondary is True.

inc
[float] Inclination of the orbit in degrees. Needed only if secondary is true.

winn_approx
[boolean] If True, instead of running the whole Kepler equation calculation, time of secondary eclipse is calculated using eq. (6) in Winn (2010; https://arxiv.org/abs/1001.2010v5)

get_secondary_time
[boolean] If True, this function also returns the time-of-mid secondary eclipse.

Returns

minphase
[float] The minimum phase constraint.

maxphase
[float] The maximum phase constraint.

tsec
[float] (optional) If get_secondary_time is True, the time of secondary eclipse in the same units as input t0.

10.3 Module contents

Part II

Installation Instructions and Notebook Availability

To install the ExoCTK package one can follow the instructions listed in our README available here on GitHub.

There are also several Jupyter Notebooks available for users to aid in learning how to use the various tools included in the ExoCTK package which can be found within the 'Notebooks' folder following download of the ExoCTK repository or for viewing online here.

Part III

Newsletter Subscription

If you'd like to stay up-to-date with our releases and updates we suggest subscribing to our newsletter. One can do so by following the instructions below:

Subscribe by email - you are not require to log in to the ListServ interface.

(1) Send an email from your mailbox to `exoctk-news-subscribe-request@maillist.stsci.edu`

(2) Subject and body should be blank. If you use an email signature, remove that as well and then send the message.

(3) You will receive a confirmation email - be sure to follow the instructions to ensure you are properly subscribed.

PYTHON MODULE INDEX

e

INDEX

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

V