CASCADe Documentation

Release 0.9 beta

Jeroen Bouwman

Feb 02, 2019

CONTENTS

1 CASCADe API documentation 3

2 Indices and tables 23

Python Module Index 25

Index 27

At present several thousand transiting exoplanet systems have been discovered. For relatively few systems, however, a spectro-photometric characterization of the planetary atmospheres could be performed due to the tiny photometric signatures of the atmospheres and the large systematic noise introduced by the used instruments or the earth atmosphere. Several methods have been developed to deal with instrument and atmospheric noise. These methods include high precision calibration and modeling of the instruments, modeling of the noise using methods like principle component analysis or Gaussian processes and the simultaneous observations of many reference stars. Though significant progress has been made, most of these methods have drawbacks as they either have to make too many assumptions or do not fully utilize all information available in the data to negate the noise terms.

The CASCADe project implements a novel "data driven" method, pioneered by Schoelkopf et al (2015) utilizing the causal connections within a data set, and uses this to calibrate the spectral timeseries data of single transiting systems. The current code has been tested successfully to spectroscopic data obtained with the Spitzer and HST observatories.

CHAPTER

ONE

CASCADE API DOCUMENTATION

1.1 Howto Install CASCADe

Clone a repository

To start working locally on an existing remote repository, clone it with the command *git clone <repository path>*. By cloning a repository, you'll download a copy of its files into your local computer, preserving the Git connection with the remote repository.

You can either clone it via HTTPS or [SSH](../ssh/README.md). If you chose to clone it via HTTPS, you'll have to enter your credentials every time you pull and push. With SSH, you enter your credentials once and can pull and push straightaway.

You can find both paths (HTTPS and SSH) by navigating to your project's landing page and clicking **Clone**. GitLab will prompt you with both paths, from which you can copy and paste in your command line.

As an example, consider a repository path:

- HTTPS: *https://gitlab.com/gitlab-org/gitlab-ce.git*
- SSH: '' git@gitlab.com:gitlab-org/gitlab-ce.git ''

To get started, open a terminal window in the directory you wish to clone the repository files into, and run one of the following commands.

Clone via HTTPS:

```
`bash git clone https://gitlab.com/gitlab-org/gitlab-ce.git `
```

Clone via SSH:

```
`bash git clone git@gitlab.com:gitlab-org/gitlab-ce.git `
```

Both commands will download a copy of the files in a folder named after the project's name.

You can then navigate to the directory and start working on it locally.

Go to the master branch to pull the latest changes from there

```
`bash git checkout master `
```

Download the latest changes in the project

This is for you to work on an up-to-date copy (it is important to do this every time you start working on a project), while you set up tracking branches. You pull from remote repositories to get all the changes made by users since the last time you cloned or pulled the project. Later, you can push your local commits to the remote repositories.

`bash git pull REMOTE NAME-OF-BRANCH `

When you first clone a repository, REMOTE is typically "origin". This is where the repository came from, and it indicates the SSH or HTTPS URL of the repository on the remote server. NAME-OF-BRANCH is usually "master", but it may be any existing branch.

1.2 Using Cascade

to run the code, first load all needed modules:

```
import cascade
```

Then, create transit spectroscopy object

```
tso = cascade.TSO.TSOSuite()
```

To reset all previous divined or initialized parameters

```
tso.execute("reset")
```

Initialize the TSO object using ini files which define the data, model parameters and behavior of the causal pixel model implemented in CASCADe.

```
path = cascade.initialize.default_initialization_path
tso = cascade.TSO.TSOSuite("initialize", "cascade_cpm.ini",
                           "cascade_object.ini",
                           "cascade_data_spectral_images.ini", path=path)
```

Load the observational data

```
tso.execute("load_data")
```

Subtract the background

```
tso.execute("subtract_background")
```

Sigma clip data

```
tso.execute("sigma_clip_data")
```

Determine the position of source from the spectroscopic data set

```
tso.execute("determine_source_position")
```

Set the extraction area within which the signal of the exoplanet will be determined

```
tso.execute("set_extraction_mask")
```

Extract the spectrum of the Star + planet in an optimal way

```
tso.execute("optimal_extraction")
```

Setup the matrix of regressors used to model the noise

```
tso.execute("select_regressors")
```

Define the eclipse model

```
tso.execute("define_eclipse_model")
```

Derive the calibrated time series and fit for the planetary signal

```
tso.execute("calibrate_timeseries")
```

Extract the planetary signal

```
tso.execute("extract_spectrum")
```

Correct the extracted planetary signal for non uniform subtraction of average eclipse/transit signal

```
tso.execute("correct_extracted_spectrum")
```

Save the planetary signal

```
tso.execute("save_results")
```

Plot results (planetary spectrum, residual etc.)

```
tso.execute("plot_results")
```

1.3 Modules of the CASCADe package

1.3.1 The cascade.TSO module

The TSO module is the main module of the CASCADe package. The classes defined in this module define the time series object and all routines acting upon the TSO instance to extract the spectrum of the transiting exoplanet.

class TSOSuite(**init_files*, *path=None*)

Bases: object

Transit Spectroscopy Object Suite class.

This is the main class containing the ligth curve data of and transiting exoplanet and all functionality to calibrate and analyse the light curves and to extractthe spectrum of the transiting exoplanet.

Parameters **init_files** (*'list' of 'str'*) – List containing all the initialization files needed to run the CASCADe code.

Raises ValueError – Raised when commands not recognized as valid

---

Examples

To make instance of TSOSuite class

```
>>> tso = cascade.TSO.TSOSuite()
```

---

execute(*command*, **init_files*, *path=None*)
Check if command is valid and excecute if True

Parameters

- **command** – Command to be excecuted. If valid the method corresponding to the command will be excecuted
- ***init_files** – Single or multiple file names of the .ini files containing the parameters defining the observation and calibration settings.
- **path** – (optional) Filepath to the .ini files, standard value in None

Raises `ValueError` – error is raised if command is not valid

Examples

```
>>> tso.execute('reset')
```

---

initialize_TSO(**init_files*, *path=None*)
Initializes the TSO object by reading in a single or multiple .ini files

Parameters

- ***init_files** – Single or multiple file names of the .ini files containing the parameters defining the observation and calibration settings.
- **path** – (optional) Filepath to the .ini files, standard value in None

Variables **cascade_parameters** – cascade.initialize.initialize.configurator

Raises `FileNotFoundError` – Raises error if .ini file is not found

Examples

```
>>> tso.execute("initialize", init_flle_name)
```

---

reset_TSO()
Reset initialization of TSO object by removing all loaded parameters.

Examples

```
>>> tso.execute("reset")
```

---

load_data()
Load the transit time series observations from file, for the object, observatory, instrument and file location specified in the loaded initialization files

Variables **observation** – cascade.instruments.instruments.Observation

---

Examples

To load the observed data into the tso object:

```
>>> tso.execute("load_data")
```

subtract_background()
Subtract median background determined from data or background model from the science observations.

Variables **isBackgroundSubtracted** – True if background is subtracted

Raises AttributeError – In case no background data is defined

Examples

To subtract the background from the spectral images:

```
>>> tso.execute("subtract_background")
```

static sigma_clip_data_cosmic(*data*, *sigma*)
Sigma Clip in time.

Parameters

- **data** – input data to be cliped
- **sigma** – sigma value of sigmaclip

Returns *sigma_clip_mask* – updated mask

sigma_clip_data()
Perform sigma clip on science data to flag bad data.

create_cleaned_dataset()
Create a cleaned dataset to be used in regresion analysis.

define_eclipse_model()
This function defines the light curve model used to analize the transit or eclipse. We define both the actual trasit/eclipse signal as wel as an calibration signal.

determine_source_position()
This function determines the position of the source in the slit over time and the spectral trace.

We check if trace and position are already set, if not, determine them from the data by deriving the "center of light" of the dispersed light.

Parameters

- **spectral_trace** – The trace of the dispersed light on the detector normalized to its median position. In case the data are extracted spectra, the trace is zero.
- **position** – Postion of the source on the detector in the cross dispersion directon as a function of time, normalized to the median position.
- **median_position** – median source position.

Returns

- *position* – position of source in time

- *med_position* – median (in time) position of source

set_extraction_mask()
Set mask which defines the area of interest within which a transit signal will be determined. The mask is set along the spectral trace with a pixel width of nExtractionWidth

Returns *mask* – In case data are Spectra : 1D mask In case data are Spectral images or cubes: 2D mask

_create_edge_mask(*kernel*, *roi_mask_cube*)
Create an edge mask to mask all pixels for which the convolution kernel extends beyond the ROI :param kernel: :param roi_mask:

Returns *edge_mask*

_create_extraction_profile(*cleaned_data_with_roi_mask*, *extracted_spectra*, *kernel*, *mask_for_extraction*)
Create the normilzed source profile used for optimal extraction. The cleaned data is convolved with an appropriate kernel to smooth the profile and to increase the SNR. On the edges, where the kernel extends over the boundary, non convolved data is used to prevent edge effects

Parameters

- **cleaned_data_with_roi_mask** –
- **extracted_spectra** –
- **kernel** –
- **mask_for_extraction** –

Returns *extraction_profile*

_create_3dKernel(*sigma_time*)
Create a 3d Kernel from 2d Instrument specific Kernel to include time dimention

Parameters **sigma_time** –

Returns *3dKernel*

optimal_extraction()
Optimally extract spectrum using procedure of Horne 1986, PASP 98, 609

Returns *Optimally extracted 1d Spectra*

select_regressors()
Select pixels which will be used as regressors.

List of regressors, using the following list index: first index: [# nod] second index: [# valid pixel in extraction mask] third index: [0=pixel coord; 1=list of regressors] forth index: [0=coordinate wave direction;

1=coordinate spatial direction]

static get_design_matrix(*cleaned_data_in*, *original_mask_in*, *regressor_selection*, *nrebin*, *clip=False*, *clip_pctl_time=0.0*, *clip_pctl_regressors=0.0*)
Return the design matrix based on the data set itself

cleaned_data_in time series data with bad pixels corrected

original_mask_in data mask before cleaning

regressor_selection nrebin clip clip_pctl_time clip_pctl_regressors

Design Matirx

reshape_data(*data_in*)
Reshape the time series data to a uniform dimentional shape

return_all_design_matrices(*clip=False*, *clip_pctl_time=0.0*, *clip_pctl_regressors=0.0*)
Setup the regression matrix based on the sub set of the data slected to be used as calibrators.

list with design matrici with the following index convention: first index: [# nods] second index : [# of valid pixels within extraction mask] third index : [0]

calibrate_timeseries()
Calibrate the ligth curve data

extract_spectrum()
Extract the planetary spectrum from the calibrated ligth curve data

correct_extracted_spectrum()
Make correction for non-uniform subtraction of transit signal due to differences in the relative weighting of the regressors

save_results()
Save results

plot_results()
Plot the extracted planetary spectrum and scaled signal on the detector.

1.3.2 The cascade.cpm_model module

Routines used in causal pixel model

Version 1.0

@author: Jeroen Bouwman MPIA Heidelberg

solve_linear_equation(*design_matrix*, *data*, *weights=None*, *cv_method='gcv'*, *reg_par={'lam0': 1e-06, 'lam1': 100.0, 'nlam': 60}*, *feature_scaling='norm'*, *degrees_of_freedom=None*)
Solve linear system using SVD with TIKHONOV regularization

For details see:

PHD thesis by Diana Maria SIMA, "Regularization techniques in Model Fitting and Parameter estimation", KU Leuven 2006

Hogg et al 2010, "Data analysis recipies: Fitting a model to data" Rust & O'Leaary, "Residual periodograms for choosing regularization

parameters for ill-posed porblems"

Krakauer et al "Using generalized cross-validationto select parameters in inversions for regional carbon fluxes"

This routine solves the linear equation

A x = y

by finding optimal solution x_hat by minimizing

||y-A*x_hat||^2 + lambda * ||x_hat||^2

Input parameters:

design_matrx: Design matrix

data: Data

weights: Weights used in the linear least square minimization

cv_method:

Method used to find optimal regularization parameter which can be: "gvc" : Generizalize Cross Validation [RECOMMENDED!!!] "b95" : normalized cumulatative periodogram using 95% limit "B100": normalized cumulatative periodogram

reg_par: Parameter describing search grid to find optimal regularization parameter lambda:

lam0 : minimum lambda lam1 : maximum lambda nlam : number of grid points

feature_scaling: norm : normalise features using L2 norm None : no featue scaling

return_PCR(*design_matrix*, *n_components=None*, *variance_prior_scaling=1.0*)

Perform principal component regression with marginalization. To marginalize over the eigenlightcurves we need to solve x = (A.T V^(-1) A)^(-1) * (A.T V^(-1) y), where V = C + B.T Lambda B, with B matrix containing the eigenlightcurves and lambda the median squared amplitudes of the eigenlightcurves.

1.3.3 The cascade.data_model module

Created on Sun Jul 10 16:14:19 2016

@author: bouwman

class SpectralData(*wavelength=nan*, *wavelength_unit=None*, *data=nan*, *data_unit=None*, *uncertainty=nan*, *mask=False*, ***kwargs*)

Bases: `cascade.data_model.data_model.InstanceDescriptorMixin`

Class defining basic properties of spectral data INPUT, required: ————— wavelength

wavelength of data (can be frequencies)

wavelenth_unit The physical unit of the wavelength (uses astropy.units)

data spectral data

data_unit the physical unit of the data (uses astropy.units)

uncertainty uncertainty on spectral data

mask mask defining masked data

****kwargs** any auxilary data relevant to the spectral data (like position, detector temperature etc.) If unit is not explicitly given a unit atribute is added. Input argument can be instance of astropy quantity. Auxilary atributes are added to instance of the SpectralData class and not to the class itself. Only the required input stated above is always defined for all instances.

Instance of SpectralData class. All data are stored internally as numppy arrays. Outputted data are astropy Quantities unless no units (=None) are specified.

wavelength

wavelength_unit

data

uncertainty

data_unit

mask

class SpectralDataTimeSeries(*wavelength=nan*, *wavelength_unit=None*, *data=array([[nan]])*, *data_unit=None*, *uncertainty=array([[nan]])*, *mask=False*, *time=nan*, *time_unit=None*, ***kwargs*)

Bases: *cascade.data_model.data_model.SpectralData*

Class defining timeseries of spectral data. This class inherits from SpectralData. The data now has one additional dimension: time Input: —– time

time of observation

time_unit physical unit of time data

time

time_unit

1.3.4 The cascade.exoplanet_tools module

This Module defines the functionality to get catalog data on the targeted exoplanet and define the model ligth curve for the system. It also difines some usefull functionality for exoplanet atmosphere analysis.

Kmag = Unit("Kmag")
Definition of generic K band magnitude

Vmag = Unit("Vmag")
Definition of a generic Vband magnitude

masked_array_input(*func*)
Decorator function to check and handel masked Quantities

If one of the input arguments is wavelength or flux, the array can be a masked Quantity, masking out only 'bad' data. This decorator checks for masked arrays and upon finding the first masked array, passes the data and stores the mask to be used to create a masked Quantity after the function returns.

Parameters **func** (*method*) – Function to be decorated

KmagToJy (*magnitude: Unit("Kmag")*, *system='Johnson'*)
Convert Kband Magnitudes to Jy

Parameters

- **magnitude** (*'Kmag'*) – Input K band magnitude to be converted to Jy.
- **system** (*'str'*) – optional, either 'Johnson' or '2MASS', default is 'Johnson'

Returns **flux** (*'astropy.units.Quantity', u.Jy*) – Flux in Jy, converted from input Kband magnitude

Raises AssertionError – raises error if Photometric system not recognized

JytoKmag (*flux: Unit("Jy")*, *system='Johnson'*)
Convert flux in Jy to Kband Magnitudes

Parameters

- **flux** (*'astropy.units.Quantity', 'u.Jy or equivalent'*) – Input Flux to be converted K band magnitude.
- **system** (*'str'*) – optional, either 'Johnson' or '2MASS', default is 'Johnson'

Returns **magnitude** (*'astropy.units.Quantity', Kmag*) – Magnitude converted from input fkux value

Raises AssertionError – raises error if Photometric system not recognized

Planck (*wavelength: Unit("micron")*, *temperature: Unit("K")*)
This function calculates the emisison from a Black Body.

Parameters

- **wavelength** (*'astropy.units.Quantity'*) – Input wavelength in units of microns or equivalent
- **temperature** (*'astropy.units.Quantity'*) – Input temperature in units of Kelvin or equivalent

Returns **blackbody** (*'astropy.units.Quantity'*) – B_nu in cgs units [erg/s/cm2/Hz/sr]

SurfaceGravity (*MassPlanet: Unit("jupiterMass")*, *RadiusPlanet: Unit("jupiterRad")*)
Calculates surface gravity of planet

Parameters

- **MassPlanet** – Mass of planet in units of Jupiter mass or equivalent
- **RadiusPlanet** – Radius of planet in units of Jupiter radius or equivalent

Returns *sgrav* – Surface gravity in units of m s-2

ScaleHeight (*MeanMolecularMass: Unit("u")*, *SurfaceGravity: Unit("m / s2")*, *Temperature: Unit("K")*)
Calculate the scaleheigth of the planet

Parameters

- **MeanMolecularMass** (*'astropy.units.Quantity'*) – in units of mass of the hydrogen atom or equivalent

- **SurfaceGravity** (*'astropy.units.Quantity'*) – in units of m s-2 or equivalent
- **Temperature** (*'astropy.units.Quantity'*) – in units of K or equivalent

Returns **ScaleHeight** (*'astropy.units.Quantity'*) – scaleheigth in unit of km

TransitDepth (*RadiusPlanet: Unit("jupiterRad")*, *RadiusStar: Unit("solRad")*)

Calculates the depth of the planetary transit assuming one can neglect the emision from the night side of the planet.

Parameters

- **Planet** (*Radius*) – Planetary radius in Jovian radii or equivalent
- **Star** (*Radius*) – Stellar radius in Solar radii or equivalent

Returns *depth* – Relative transit depth (unit less)

EquilibriumTemperature (*StellarTemperature: Unit("K")*, *StellarRadius: Unit("solRad")*, *SemiMajorAxis: Unit("AU")*, *Albedo=0.3*, *epsilon=0.7*)

Calculate the Equlibrium Temperature of the Planet

Parameters

- **StellarTemperature** (*'astropy.units.Quantity'*) – Temperature of the central star in units of K or equivalent
- **StellarRadius** (*'astropy.units.Quantity'*) – Radius of the central star in units of Solar Radii or equivalent
- **Albedo** (*'float'*) – Albedo of the planet.
- **SemiMajorAxis** (*'astropy.units.Quantity'*) – The semi-major axis of platetary orbit in units of AU or equivalent
- **epsilon** (*'float'*) – Green house effect parameter

Returns **ET** (*'astropy.units.Quantity'*) – Equlibrium Temperature of the exoplanet

convert_spectrum_to_brighness_temperature (*wavelength: Unit("micron")*, *contrast: Unit("%")*, *StellarTemperature: Unit("K")*, *StellarRadius: Unit("solRad")*, *RadiusPlanet: Unit("jupiterRad")*, *error: Unit("%") = None*)

Function to convert the secondary eclipse spectrum to brightness temperature.

Parameters

- **wavelength** – Wavelength in u.micron or equivalent unit.
- **contrast** – Contrast between planet and star in u.percent.
- **StellarTemperature** – Temperature if the star in u.K or equivalent unit.
- **StellarRadius** – Radius of the star in u.R_sun or equivalent unit.
- **RadiusPlanet** – Radius of the planet in u.R_jupiter or equivalent unit.
- **error** – (optional) Error on contrast in u.percent (standart value = None).

Returns

- *brighness_temperature* – Eclipse spectrum in units of brightness temperature.
- *error_brighness_temperature* – (optional) Error on the spectrum in units of brightness temperature.

eclipse_to_transit (*eclipse*)
Converts eclipse spectrum to transit spectrum

Parameters **eclipse** – Transit depth values to be converted

Returns *transit* – transit depth values derived from input eclipse values

transit_to_eclipse (*transit*)
Converts transit spectrum to eclipse spectrum

Parameters **transit** – Transit depth values to be converted

Returns *eclipse* – eclipse depth values derived from input transit values

combine_spectra (*identifier_list=[]*, *path=''*)
Convienience function to combine multiple extracted spectra of the same source by calculating a weighted averige.

Parameters

- **identifier_list** (*'list' of 'str'*) – List of file identifiers of the individual spectra to be combined
- **path** (*'str'*) – path to the fits files

Returns **combined_spectrum** (*'array_like'*) – The combined spectrum based on the spectra specified in the input list

get_calalog (*catalog_name*, *update=True*)
Get exoplanet catalog data

Parameters

- **catalog_name** (*'str'*) – name of catalog to use
- **update** (*'bool'*) – Boolian indicating if local calalog file will be updated

Returns **files_downloaded** (*'list' of 'str'*) – list of downloaded catalog files

parse_database (*catalog_name*, *update=True*)
Read CSV files containing exoplanet catalog data

Parameters

- **catalog_name** (*'str'*) – name of catalog to use
- **update** (*'bool'*) – Boolian indicating if local calalog file will be updated

Returns **table_list** (*'list' of 'astropy.table.Table'*) – List containing astropy Tables with the parameters of the exoplanet systems in the database.

Note:

The following exoplanet databases can be used: The Transing exoplanet catalog (TEPCAT) The NASA exoplanet Archive The Exoplanet Orbit Database

---

Raises `ValueError` – Raises error if the input catalog is nor recognized

extract_exoplanet_data(*data_list*, *target_name_or_position*, *coord_unit=None*, *coordinate_frame='icrs'*, *search_radius=<Quantity 5. arcsec>*)

Extract the data record for a single target

Parameters

- **data_list** (*'list' of 'astropy.Table'*) – List containing table with exoplanet data
- **target_name_or_position** – Name of the target or coordinates of the target for which the record is extracted
- **coord_unit** – Unit of coordinates e.g (u.hourangle, u.deg)
- **coordinate_frame** – Frame of coordinate system e.g icrs

Returns **table_list** (*'list'*) – List containing data record of the specified planet

class lightcuve

Bases: `object`

Class defining lightcurve model used to model the observed transit/eclipse observations. Current valid lightcurve models: batman

Variables

- **lc** (*'array_like'*) – The lightcurve model
- **par** (*'ordered_dict'*) – The lightcurve parameters

Notes

Uses factory method to pick model/package used to calculate the lightcurve model.

Raises `ValueError` – Error is raised if no valid lightcurve model is defined

valid_models = {'batman'}

class batman_model

Bases: `object`

This class defines the lightcurve model used to analyse the observed transit/eclipse using the batman package by Laura Kreidberg[1].

Variables

- **lc** (*'array_like'*) – The values of the lightcurve model
- **par** (*'ordered_dict'*) – The model parameters difining the lightcurve model

---

References

---

[1] Kreidberg, L. 2015, PASP 127, 1161

static define_batman_model(*InputParameter*)
This function defines the light curve model used to analize the transit or eclipse. We use the batman package to calculate the light curves.We assume here a symmetric transit signal, that the secondary transit is at phase 0.5 and primary transit at 0.0.

Parameters **InputParameter** (*'dict'*) – Ordered dict containing all needed inut parameter to define model

Returns

- **tmodel** (*'array_like'*) – Orbital phase of planet used for lightcurve model
- **lcmode** (*'array_like'*) – Normalized values of the lightcurve model

ReturnParFromIni()
Get relevant parameters for lightcurve model from CASCADe intitialization files

Returns par (*'ordered_dict'*) – input model parameters for batman lightcurve model

ReturnParFromDB()
Get relevant parameters for lightcurve model from exoplanet database specified in CASCADe initialization file

Returns par (*'ordered_dict'*) – input model parameters for batman lightcurve model

Raises `ValueError` – Raises error in case the observation type is not recognized.

1.3.5 The cascade.initialize module

This Module defines the functionality to generate and read .ini files which are used to initialize CASCADe.

Created on Thu Mar 16 21:02:31 2017

@author: bouwman

generate_default_initialization()
Convenience function to generate example .ini file for CASCADe

class configurator(**file_names*)
Bases: `object`

isInitialized = False

reset()

1.3.6 The cascade.instruments module

CASCADe

Observatory and Instruments specific Module

@author: bouwman

class Observation
Bases: object

This class handles the selection of the correct observatory and instrument classes and loads the time series data to be analyzed

class HST
Bases: cascade.instruments.instruments.ObservatoryBase

This observatory class defines the instuments and data handling for the spectropgraphs of the Spitzer Space telescope

name

location

NAIF_ID

observatory_instruments

class HSTWFC3
Bases: cascade.instruments.instruments.InstrumentBase

This instrument class defines the properties of the WFC3 instrument of the Hubble Space Telescope

name

load_data()

get_instrument_setup()
Retrieve all relevant parameters defining the instrument and data setup

get_spectra(*is_background=False*)
read (uncalibrated) spectral timeseries, phase and wavelength

get_spectral_images(*is_background=False*)
read uncalibrated spectral images (flt data product)

_define_convolution_kernel()
Define the instrument specific convolution kernel which will be used in the correction procedure of bad pixels

_define_region_of_interest()
Defines region on detector which containes the intended target star.

_get_background_cal_data()
Get the calibration data from which the background in the science images can be determined. For further details see: http://www.stsci.edu/hst/wfc3/documents/ISRs/WFC3-2015-17.pdf

_fit_background(*science_data_in*)
Fits the background in the HST Grism data using the method described in: http://www.stsci.edu/hst/wfc3/documents/ISRs/WFC3-2015-17.pdf

_determine_relative_source_position(*spectral_image_cube*, *mask*)
Determine the shift of the spectra (source) relative to the first integration. Note that it is important for this to work properly to have identified bad pixels and to correct the values using an edge preserving correction, i.e. an correction which takes into account the dispersion direction and psf size (relative to pixel size) Input: ——

spectral_image_cube

mask

relative x and y position as a function of time.

_determine_source_position_from_cal_image(*calibration_image_cube*, *calibration_data_files*)
Determines the source position on the detector of the target source in the calibration image takes prior to the spectroscopic observations.

_read_grism_configuration_files()
Gets the relevant data from WFC3 configuration files

_read_reference_pixel_file()
Read the calibration file containig the definition of the reference pixel appropriate for a given sub array and or filer

static _search_ref_pixel_cal_file(*ptable*, *inst_aperture*, *inst_filter*)
Search the reference pixel calibration file for the reference pixel given the instrument aperture and filter. See also http://www.stsci.edu/hst/observatory/apertures/wfc3.html

_get_subarray_size(*calibration_data*, *spectral_data*)

_get_wavelength_calibration()
Return the wavelength calibration

get_spectral_trace()
Get spectral trace

static _WFC3Trace(*xc*, *yc*, *DYDX*, *xref=522*, *yref=522*, *xref_grism=522*, *yref_grism=522*, *subarray=256*, *subarray_grism=256*)
This function defines the spectral trace for the wfc3 grism modes. Details can be found in:

http://www.stsci.edu/hst/wfc3/documents/ISRs/WFC3-2016-15.pdf

and http://www.stsci.edu/hst/observatory/apertures/wfc3.html

static _WFC3Dispersion(*xc*, *yc*, *DYDX*, *DLDP*, *xref=522*, *yref=522*, *xref_grism=522*, *yref_grism=522*, *subarray=256*, *subarray_grism=256*)
Convert pixel coordinate to wavelength. Method and coefficient adopted from Kuntschner et al. (2009), Wilkins et al. (2014). See also http://www.stsci.edu/hst/wfc3/documents/ISRs/WFC3-2016-15.pdf

In case the direct image and spectral image are not taken with the same aperture, the centroid measurement is adjusted according to the table in: http://www.stsci.edu/hst/observatory/apertures/wfc3.html

xc: X coordinate of direct image centroid

yc: Y coordinate of direct image centroid

xref yref xref_grism yref_grism subarray subarray_grism

wavelength: return wavelength mapping of x coordinate in micron

class Spitzer
Bases: `cascade.instruments.instruments.ObservatoryBase`

This observatory class defines the instuments and data handling for the spectropgraphs of the Spitzer Space telescope

name

location

NAIF_ID

observatory_instruments

class SpitzerIRS

Bases: `cascade.instruments.instruments.InstrumentBase`

This instrument class defines the properties of the IRS instrument of the Spitzer Space Telescope

name

load_data()

get_instrument_setup()

Retrieve all relevant parameters defining the instrument and data setup

get_spectra(*is_background=False*)

read uncalibrated spectral timeseries, phase and wavelength

get_spectral_images(*is_background=False*)

read uncalibrated spectral images

Notes on FOV:

in the fits header the following relevant info is used: # FOVID 26 IRS_Short-Lo_1st_Order_1st_Position # FOVID 27 IRS_Short-Lo_1st_Order_2nd_Position # FOVID 28 IRS_Short-Lo_1st_Order_Center_Position # FOVID 29 IRS_Short-Lo_Module_Center # FOVID 32 IRS_Short-Lo_2nd_Order_1st_Position # FOVID 33 IRS_Short-Lo_2nd_Order_2nd_Position # FOVID 34 IRS_Short-Lo_2nd_Order_Center_Position # FOVID 40 IRS_Long-Lo_1st_Order_Center_Position # FOVID 46 IRS_Long-Lo_2nd_Order_Center_Position

Notes on timing:

FRAMTIME the total effective exposure time (ramp length) in seconds

_define_convolution_kernel()

Define the instrument specific convolution kernel which will be used in the correction procedure of bad pixels

_define_region_of_interest()

Defines region on detector which containes the intended target star.

_get_order_mask()

Gets the mask which defines the pixels used with a given spectral order

_get_wavelength_calibration()

Get wavelength calibration file

get_detector_cubes(*is_background=False*)

Get detector cube data

Notes on timing in header:

There are several integration-time-related keywords. Of greatest interest to the observer is the "effective integration time", which is the time on-chip between the first and last non-destructive reads for each pixel. It is called:

RAMPTIME = Total integration time for the current DCE.

The value of RAMPTIME gives the usable portion of the integration ramp, occurring between the beginning of the first read and the end of the last read. It excludes detector array pre-conditioning time. It may also be of interest to know the exposure time at other points along the ramp. The SUR sequence consists of the time taken at the beginning of a SUR sequence to condition the array (header keyword DEADTIME), the time taken to complete one read and one spin through the array (GRPTIME), and the non-destructive reads separated by uniform wait times. The wait consists of "clocking" through the array without reading or resetting. The time it takes to clock through the array once is given by the SAMPTIME keyword. So, for an N-read ramp:

RAMPTIME = 2x(N-1)xSAMPTIME

and DCE duration = DEADTIME + GRPTIME + RAMPTIME

Note that peak-up data is not obtained in SUR mode. It is obtained in Double Correlated Sampling (DCS) mode. In that case, RAMPTIME gives the time interval between the 2nd sample and the preceeding reset.

get_spectral_trace()
Get spectral trace

1.3.7 The cascade.utilities module

This Module defines some utility functions used in cascade

@author: bouwman

write_timeseries_to_fits(*data*, *path*)
Write spectral timeseries data object to fits files

find(*pattern*, *path*)
Return a list of all data files

spectres(*new_spec_wavs*, *old_spec_wavs*, *spec_fluxes*, *spec_errs=None*)
SpectRes: A fast spectral resampling function. Copyright (C) 2017 A. C. Carnall Function for resampling spectra (and optionally associated uncertainties) onto a new wavelength basis.

Parameters

- **new_spec_wavs** (*numpy.ndarray*) – Array containing the new wavelength sampling desired for the spectrum or spectra.
- **old_spec_wavs** (*numpy.ndarray*) – 1D array containing the current wavelength sampling of the spectrum or spectra.
- **spec_fluxes** (*numpy.ndarray*) – Array containing spectral fluxes at the wavelengths specified in old_spec_wavs, last dimension must correspond to the shape of old_spec_wavs. Extra dimensions before this may be used to include multiple spectra.

- **spec_errs** (*numpy.ndarray (optional)*) – Array of the same shape as spec_fluxes containing uncertainties associated with each spectral flux value.

Returns

- **resampled_fluxes** (*numpy.ndarray*) – Array of resampled flux values, first dimension is the same length as new_spec_wavs, other dimensions are the same as spec_fluxes
- **resampled_errs** (*numpy.ndarray*) – Array of uncertainties associated with fluxes in resampled_fluxes. Only returned if spec_errs was specified.

CHAPTER

TWO

INDICES AND TABLES

- genindex
- search

PYTHON MODULE INDEX

C

cascade.cpm_model.cpm_model, 9
cascade.data_model.data_model, 10
cascade.exoplanet_tools.exoplanet_tools, 11
cascade.initialize.initialize, 16
cascade.instruments.instruments, 16
cascade.TSO.TSO, 5
cascade.utilities.utilities, 20

INDEX

Symbols

_WFC3Dispersion() (*HSTWFC3 static method*), 18
_WFC3Trace() (*HSTWFC3 static method*), 18
_create_3dKernel() (*TSOSuite method*), 8
_create_edge_mask() (*TSOSuite method*), 8
_create_extraction_profile() (*TSOSuite method*), 8
_define_convolution_kernel() (*HSTWFC3 method*), 17
_define_convolution_kernel() (*SpitzerIRS method*), 19
_define_region_of_interest() (*HSTWFC3 method*), 17
_define_region_of_interest() (*SpitzerIRS method*), 19
_determine_relative_source_position() (*HSTWFC3 method*), 17
_determine_source_position_from_cal_image() (*HSTWFC3 method*), 18
_fit_background() (*HSTWFC3 method*), 17
_get_background_cal_data() (*HSTWFC3 method*), 17
_get_order_mask() (*SpitzerIRS method*), 19
_get_subarray_size() (*HSTWFC3 method*), 18
_get_wavelength_calibration() (*HSTWFC3 method*), 18
_get_wavelength_calibration() (*SpitzerIRS method*), 19
_read_grism_configuration_files() (*HSTWFC3 method*), 18
_read_reference_pixel_file() (*HSTWFC3 method*), 18
_search_ref_pixel_cal_file() (*HSTWFC3 static method*), 18

B

batman_model (*class in cascade.exoplanet_tools.exoplanet_tools*), 15

C

calibrate_timeseries() (*TSOSuite method*), 9
cascade.cpm_model.cpm_model (*module*), 9
cascade.data_model.data_model (*module*), 10
cascade.exoplanet_tools.exoplanet_tools (*module*), 11
cascade.initialize.initialize (*module*), 16
cascade.instruments.instruments (*module*), 16
cascade.TSO.TSO (*module*), 5
cascade.utilities.utilities (*module*), 20
combine_spectra() (*in module cascade.exoplanet_tools.exoplanet_tools*), 14
configurator (*class in cascade.initialize.initialize*), 16
convert_spectrum_to_brighness_temperature() (*in module cascade.exoplanet_tools.exoplanet_tools*), 13
correct_extracted_spectrum() (*TSOSuite method*), 9
create_cleaned_dataset() (*TSOSuite method*), 7

D

data (*SpectralData attribute*), 11
data_unit (*SpectralData attribute*), 11
define_batman_model() (*batman_model static method*), 15
define_eclipse_model() (*TSOSuite method*), 7
determine_source_position() (*TSOSuite method*), 7

E

eclipse_to_transit() (*in module cascade.exoplanet_tools.exoplanet_tools*), 14
EquilibriumTemperature() (*in module cascade.exoplanet_tools.exoplanet_tools*), 13
execute() (*TSOSuite method*), 5
extract_exoplanet_data() (*in module cascade.exoplanet_tools.exoplanet_tools*), 15
extract_spectrum() (*TSOSuite method*), 9

F

find() (*in module cascade.utilities.utilities*), 20

G

generate_default_initialization() (*in module cascade.initialize.initialize*), 16
get_calalog() (*in module cascade.exoplanet_tools.exoplanet_tools*), 14
get_design_matrix() (*TSOSuite static method*), 8
get_detector_cubes() (*SpitzerIRS method*), 19
get_instrument_setup() (*HSTWFC3 method*), 17
get_instrument_setup() (*SpitzerIRS method*), 19
get_spectra() (*HSTWFC3 method*), 17
get_spectra() (*SpitzerIRS method*), 19
get_spectral_images() (*HSTWFC3 method*), 17
get_spectral_images() (*SpitzerIRS method*), 19
get_spectral_trace() (*HSTWFC3 method*), 18
get_spectral_trace() (*SpitzerIRS method*), 20

H

HST (*class in cascade.instruments.instruments*), 17
HSTWFC3 (*class in cascade.instruments.instruments*), 17

I

initialize_TSO() (*TSOSuite method*), 6
isInitialized (*configurator attribute*), 16

J

JytoKmag() (*in module cascade.exoplanet_tools.exoplanet_tools*), 12

K

Kmag (*in module cascade.exoplanet_tools.exoplanet_tools*), 11
KmagToJy() (*in module cascade.exoplanet_tools.exoplanet_tools*), 11

L

lightcuve (*class in cascade.exoplanet_tools.exoplanet_tools*), 15
load_data() (*HSTWFC3 method*), 17
load_data() (*SpitzerIRS method*), 19
load_data() (*TSOSuite method*), 6
location (*HST attribute*), 17
location (*Spitzer attribute*), 19

M

mask (*SpectralData attribute*), 11
masked_array_input() (*in module cascade.exoplanet_tools.exoplanet_tools*), 11

N

NAIF_ID (*HST attribute*), 17
NAIF_ID (*Spitzer attribute*), 19
name (*HST attribute*), 17
name (*HSTWFC3 attribute*), 17
name (*Spitzer attribute*), 19
name (*SpitzerIRS attribute*), 19

O

Observation (*class in cascade.instruments.instruments*), 16
observatory_instruments (*HST attribute*), 17
observatory_instruments (*Spitzer attribute*), 19
optimal_extraction() (*TSOSuite method*), 8

P

parse_database() (*in module cascade.exoplanet_tools.exoplanet_tools*), 14

Planck() (*in module cascade.exoplanet_tools.exoplanet_tools*), 12
plot_results() (*TSOSuite method*), 9

R

reset() (*configurator method*), 16
reset_TSO() (*TSOSuite method*), 6
reshape_data() (*TSOSuite method*), 9
return_all_design_matrices() (*TSOSuite method*), 9
return_PCR() (*in module cascade.cpm_model.cpm_model*), 10
ReturnParFromDB() (*batman_model method*), 16
ReturnParFromIni() (*batman_model method*), 16

S

save_results() (*TSOSuite method*), 9
ScaleHeight() (*in module cascade.exoplanet_tools.exoplanet_tools*), 12
select_regressors() (*TSOSuite method*), 8
set_extraction_mask() (*TSOSuite method*), 8
sigma_clip_data() (*TSOSuite method*), 7
sigma_clip_data_cosmic() (*TSOSuite static method*), 7
solve_linear_equation() (*in module cascade.cpm_model.cpm_model*), 9
SpectralData (*class in cascade.data_model.data_model*), 10
SpectralDataTimeSeries (*class in cascade.data_model.data_model*), 11
spectres() (*in module cascade.utilities.utilities*), 20
Spitzer (*class in cascade.instruments.instruments*), 18
SpitzerIRS (*class in cascade.instruments.instruments*), 19
subtract_background() (*TSOSuite method*), 7
SurfaceGravity() (*in module cascade.exoplanet_tools.exoplanet_tools*), 12

T

time (*SpectralDataTimeSeries attribute*), 11
time_unit (*SpectralDataTimeSeries attribute*), 11
transit_to_eclipse() (*in module cascade.exoplanet_tools.exoplanet_tools*), 14
TransitDepth() (*in module cascade.exoplanet_tools.exoplanet_tools*), 13
TSOSuite (*class in cascade.TSO.TSO*), 5

U

uncertainty (*SpectralData attribute*), 11

V

valid_models (*lightcuve attribute*), 15
Vmag (*in module cascade.exoplanet_tools.exoplanet_tools*), 11

W

wavelength (*SpectralData attribute*), 11
wavelength_unit (*SpectralData attribute*), 11
write_timeseries_to_fits() (*in module cascade.utilities.utilities*), 20