gunagala Documentation

Release 0.1.dev0459

Anthony Horton

Apr 11, 2018

Contents

I Introduction 1

II Installation 5

1 Installing with pip 7

2 Installing from source 9

3 Running the test suite 11

III Examples 13

IV Contributing 17

V Changelog 21

4 0.1 (unreleased) 23

VI Reference/API 25

5 gunagala Package 27
5.1 Functions . . . 27

6 gunagala.imager Module 29
6.1 Functions . . . 29
6.2 Classes . . . 30

7 gunagala.optic Module 45
7.1 Functions . . . 45
7.2 Classes . . . 46

8 gunagala.optical_filter Module 49
8.1 Functions . . . 49
8.2 Classes . . . 50

9 gunagala.camera Module **55**
9.1 Classes 55

10 gunagala.psf Module **59**
10.1 Classes 59
10.2 Class Inheritance Diagram 64

11 gunagala.sky Module **65**
11.1 Classes 65
11.2 Class Inheritance Diagram 68

12 gunagala.utils Package **69**
12.1 Functions 69

Python Module Index **73**

Part I

Introduction

Gungala is a Python package for modelling the performance of astronomical instruments, including SNR/ETC/sensitivity limit calculations and generation of simulated data.

This package is not intended for rigorous, end-to-end simulations of telescope and instrument systems. Instead gunagala implements parameterised models of instrument components in order to enable rapid, efficient evaluation of instrument performance. Anticipated uses include exposure time calculators, selection of commercial off the shelf components and exploration of the design parameter space for custom components.

Gunagala includes a library of performance parameters for a number of existing commercial off the shelf instrument components (e.g. CCD cameras, optical filters, telescopes and camera lenses) and the user can easily add new/custom components either through YAML based config files or programmatically in Python.

Gunagala is named as a gesture of respect to the traditional custodians of the land on which Siding Spring Observatory sits, the Kamilaroi people of northern New South Wales. Gunagala is 'sky' in the Kamilaroi/Gamilaraay language (ref: www.dnathan.com). Aboriginal Australians have studied the night skies above Australia for at least 50000 years. To learn more about Aboriginal astronomy please visit http://www.aboriginalastronomy.com.au/.

Part II

Installation

CHAPTER 1

Installing with pip

To install using the Python package manager `pip` use the following command:

```
$ pip install git+https://github.com/AstroHuntsman/gunagala.git
```

Alternatively to install in 'editable mode' use:

```
$ pip install -e git+https://github.com/AstroHuntsman/gunagala.git
```

Depending on the configuration of your system you may want to use `pip`'s `--user` or `--root` options to change the install location. See the pip documentation for details.

`Pip` will automatically install the Python packages required by Gunagala (`numpy`, `scipy`, `astropy`, `PyYAML`, `matplotlib` and their dependencies) if they are not already installed. If you want to install specific versions of the required packages from other sources do this before installing Gunagala.

CHAPTER 2

Installing from source

The project source is in a GitHub repository at https://github.com/AstroHuntsman/gunagala. To install using git on the command line:

```
$ cd ~/Build
$ git clone https://github.com/AstroHuntsman/gunagala.git
$ cd gunagala
$ python setup.py install
```

Alternatively if you expect to make changes to the Gunagala code install with the `develop` command instead:

```
$ python setup.py develop
```

Setuptools will automatically install the Python packages required by Gunagala (`numpy`, `scipy`, `astropy`, `PyYAML`, `matplotlib` and their dependencies) if they are not already installed. If you want to install specific versions of the required packages from other sources do this before installing Gunagala.

CHAPTER 3

Running the test suite

After installing Gunagala it is recommended that you run the suite of units tests. This can be done at the command line using:

```
$ python setup.py test
```

or from within a Python interpreter with:

```
>> import gunagala
>> gunagala.test()
```

Part III

Examples

The Gunagala package includes several examples in the form of Jupyter notebooks. These can be found in the `gungala/examples` directory after installing Gunagala, or they can be viewed directly in the GitHub repository by going to https://github.com/AstroHuntsman/gunagala/tree/master/examples and clicking on the `.ipynb` files.

Part IV

Contributing

Please submit bug reports or feature requests in the form of GitHub Issues at https://github.com/AstroHuntsman/gunagala/issues. For code contributions please fork and clone the repository, create a feature branch and submit a Pull Request. We recommend the astropy Developer Documentation for a description of suitable workflows.

Part V

Changelog

CHAPTER 4

0.1 (unreleased)

Initial release

Part VI

Reference/API

CHAPTER 5

gunagala Package

5.1 Functions

test([package, test_path, args, plugins, ...])	Run the tests using py.test.

5.1.1 test

gunagala.**test**(*package=None*, *test_path=None*, *args=None*, *plugins=None*, *verbose=False*, *pastebin=None*, *remote_data=False*, *pep8=False*, *pdb=False*, *coverage=False*, *open_files=False*, ***kwargs*)

Run the tests using py.test. A proper set of arguments is constructed and passed to pytest.main.

Parameters

package
[str, optional] The name of a specific package to test, e.g. 'io.fits' or 'utils'. If nothing is specified all default tests are run.

test_path
[str, optional] Specify location to test by path. May be a single file or directory. Must be specified absolutely or relative to the calling directory.

args
[str, optional] Additional arguments to be passed to pytest.main in the `args` keyword argument.

plugins
[list, optional] Plugins to be passed to pytest.main in the `plugins` keyword argument.

verbose
[bool, optional] Convenience option to turn on verbose output from py.test. Passing True is the same as specifying `'-v'` in `args`.

pastebin
[{'failed','all',None}, optional] Convenience option for turning on py.test pastebin output.

Set to `'failed'` to upload info for failed tests, or `'all'` to upload info for all tests.

remote_data
[bool, optional] Controls whether to run tests marked with @remote_data. These tests use online data and are not run by default. Set to True to run these tests.

pep8
[bool, optional] Turn on PEP8 checking via the pytest-pep8 plugin and disable normal tests. Same as specifying `'--pep8 -k pep8'` in `args`.

pdb
[bool, optional] Turn on PDB post-mortem analysis for failing tests. Same as specifying `'--pdb'` in `args`.

coverage
[bool, optional] Generate a test coverage report. The result will be placed in the directory htmlcov.

open_files
[bool, optional] Fail when any tests leave files open. Off by default, because this adds extra run time to the test suite. Requires the psutil package.

parallel
[int, optional] When provided, run the tests in parallel on the specified number of CPUs. If parallel is negative, it will use the all the cores on the machine. Requires the pytest-xdist plugin installed. Only available when using Astropy 0.3 or later.

kwargs
Any additional keywords passed into this function will be passed on to the astropy test runner. This allows use of test-related functionality implemented in later versions of astropy without explicitly updating the package template.

CHAPTER 6

gunagala.imager Module

Imaging instruments

6.1 Functions

create_imagers([config])	Parse config and create a corresponding dictionary of Imager objects.

6.1.1 create_imagers

gunagala.imager.**create_imagers**(*config=None*)

Parse config and create a corresponding dictionary of Imager objects.

Parses a configuration and creates a dictionary of Imager objects corresponding to the imaging instruments described by that config. The config can be passed to the function as a dictionary object, otherwise the config will be read from the gunagala/data/performance.yaml and, if it exists, the performance_local.yaml file.

Parameters

config
[dict, optional] a dictionary containing the performance data configuration. If not specified load_config() will be used to attempt to load a performance.yaml and/or performance_local.yaml file and use the resulting config.

Returns

imagers: dict
dictionary of Imager objects.

6.2 Classes

Imager(optic, camera, filters, psf, sky[, ...])	Class representing an astronomical imaging instrument.

6.2.1 Imager

class gunagala.imager.**Imager**(*optic*, *camera*, *filters*, *psf*, *sky*, *num_imagers=1*, *num_per_computer=1*)

Bases: object

Class representing an astronomical imaging instrument.

Class representing a complete astronomical imaging system, including optics, optical filters and camera.

Also includes point spread function and sky background models. Optionally it can be used to represent an array of identical, co-aligned imager using the num_imagers parameter to specify the number of copies.

Parameters

optic
[gunagala.optic.Optic] Optical system model.

camera
[gunagala.camera.Camera] Camera (image sensor) model.

filters
[dict of gunagala.filter.Filter] Dictionary of optical filter models.

psf
[gunagala.psf.PSF] Point spread function model.

sky
[gunagala.sky.Sky] Sky background model.

num_imagers
[int, optional] the number of identical, co-aligned imagers represented by this Imager. The default is 1.

num_per_computer
[int, optional] number of cameras connected to each computer. Used in situations where multiple cameras must be readout sequentially so the effective readout time is equal to the readout time of a single camera multiplied by the num_per_computer. The default is 1.

Attributes

optic
[gunagala.optic.Optic] Same as parameters.

camera
[gunagala.camera.Camera] Same as parameters.

filters
[dict] Same as parameters.

psf
[gunagala.psf.PSF] Same as parameters.

sky
[gunagala.sky.Sky] Same as parameters.

num_imagers
[int] Same as parameters.

num_per_computer
[int] Same as parameters.

filter_names
[list of str] List of filter names from `filters`.

pixel_scale
[astropy.units.Quantity] Pixel scale in arcseconds/pixel units.

pixel_area
[astropy.units.Quantity] Pixel area in arseconds^2/pixel units.

field_of_view
[astropy.units.Quantity] Field of view (horizontal, vertical) in degrees.

wcs
[astropy.wcs.WCS] Template world coordinate system (WCS) for sky coordinate/pixel coordinate mapping.

wavelengths
[astropy.units.Quantity] List of wavelengths used for wavelength dependent attributes/calculations.

efficiencies
[dict of astropy.units.Quantity] End to end efficiency as a function of wavelegth for each filter bandpass.

efficiency
[dict of astropy.units.Quantity] Mean end to end efficiencies for each filter bandpass.

mean_wave
[dict of astropy.units.Quantity] Mean wavelength for each filter bandpass.

pivot_wave
[dict of astropy.units.Quantity] Pivot wavelength for each filter bandpass.

bandwidth
[dict of astropy.units.Quantity] Bandwidths for each filter bandpass (STScI definition).

sky_rate
[dict of astropy.units.Quantity] Detected electrons/s/pixel due to the sky background for each filter bandpass.

Methods Summary

`ABmag_to_flux`(mag, filter_name)	Converts brightness of the target to total flux, integrated over the filter band.
`ABmag_to_rate`(mag, filter_name)	Converts AB magnitudes to photo-electrons per second in the image sensor
`SB_to_rate`(mag, filter_name)	Converts surface brightness AB magnitudes (per arcsecond squared) to photo-electrons per pixel per second.
`exp_time_sequence`(filter_name[, ...])	Calculates a sequence of sub exposures to use to span a given range of either point source brightness or exposure time.

Continued on next page

Table 3 – continued from previous page

extended_source_etc(surface_brightness, ...)	Calculates the total exposure time required to reach a given signal to noise ratio for a given extended source surface brightness.
extended_source_limit(total_exp_time, ...[, ...])	Calculates the limiting extended source surface brightness for a given minimum signal to noise ratio and total exposure time.
extended_source_saturation_exp(...[, n_sigma])	Calculates the maximum (sub) exposure time that will definitely avoid saturation for an extended source of given surface brightness.
extended_source_saturation_mag(sub_exp_time, ...)	Calculates the surface brightness of the brightest extended source that would definitely not saturate the image sensor in a given (sub) exposure time.
extended_source_signal_noise(...[, ...])	Calculates the signal and noise for an extended source with given surface brightness.
extended_source_snr(surface_brightness, ...)	Calculates the signal to noise ratio for an extended source with given surface brightness.
flux_to_ABmag(flux, filter_name)	Converts total flux of the target, integrated over the filter band, to magnitudes.
point_source_etc(brightness, filter_name, ...)	Calculates the total exposure time required to reach a given signal to noise ratio for a given point source brightness.
point_source_limit(total_exp_time, ...[, ...])	Calculates the limiting point source surface brightness for a given minimum signal to noise ratio and total exposure time.
point_source_saturation_exp(brightness, ...)	Calculates the maximum (sub) exposure time that will definitely avoid saturation for point source of given brightness
point_source_saturation_mag(sub_exp_time, ...)	Calculates the magnitude of the brightest point source that would definitely not saturate the image sensor in a given (sub) exposure time.
point_source_signal_noise(brightness, ...[, ...])	Calculates the signal and noise for a point source of a given brightness, assuming PSF fitting photometry
point_source_snr(brightness, filter_name, ...)	Calculates the signal to noise ratio for a point source of a given brightness, assuming PSF fitting photometry
rate_to_ABmag(rate, filter_name)	Converts photo-electrons per second in the image sensor to AB magnitudes
rate_to_SB(SB_rate, filter_name)	Converts photo-electrons per pixel per second to surface brightness AB magnitudes (per arcsecond squared)
snr_vs_ABmag(exp_times, filter_name[, ...])	Calculates PSF fitting signal to noise ratio as a function of point source brightness for the combined data resulting from a given sequence of sub exposures.
total_elapsed_time(exp_list)	Calculates the total elapsed time required for a given a list of sub-exposure times.
total_exposure_time(total_elapsed_time, ...)	Calculates total exposure time given a total elapsed time and sub-exposure time.

Methods Documentation

ABmag_to_flux(*mag*, *filter_name*)

Converts brightness of the target to total flux, integrated over the filter band.

Parameters

mag
[astropy.units.Quantity] Brightness of the target in AB magnitudes

filter_name
[str] Name of the optical filter to use

Returns

flux
[astropy.units.Quantity] Corresponding total flux in Watts per square metre

Notes

The conversion between band averaged magnitudes and total flux depends somewhat on the spectrum of the source. For this calculation we assume F_ν is constant.

ABmag_to_rate(*mag*, *filter_name*)
Converts AB magnitudes to photo-electrons per second in the image sensor

Parameters

mag
[astropy.units.Quantity] Source brightness in AB magnitudes

filter_name
[str] Name of the optical filter to use

Returns

rate
[astropy.units.Quantity] Corresponding photo-electrons per second

SB_to_rate(*mag*, *filter_name*)
Converts surface brightness AB magnitudes (per arcsecond squared) to photo-electrons per pixel per second.

Parameters

mag
[astropy.units.Quantity] Source surface brightness in AB magnitudes

filter_name
[str] Name of the optical filter to use

Returns

rate
[astropy.units.Quantity] Corresponding photo-electrons per pixel per second

Notes

At the time of writing `astropy.units` did not support the commonly used (but dimensionally nonsensical) expression of surface brightness in 'magnitudes per arcsecond squared'. Consequently the `mag` surface

brightness parameter should have a units of `astropy.unit.ABmag`, the 'per arcsecond squared' is implied.

exp_time_sequence(*filter_name*, *bright_limit=None*, *shortest_exp_time=None*, *longest_exp_time=None*, *faint_limit=None*, *num_long_exp=None*, *exp_time_ratio=2.0*, *snr_target=5.0*)

Calculates a sequence of sub exposures to use to span a given range of either point source brightness or exposure time.

If required the sequence will begin with an 'HDR block' of progressly increasing exposure time, followed by 1 or more exposures of equal length with the number of long exposures either specified directly or calculated from the faintest point source that the sequence is intended to detect.

Parameters

filter_name
[str] Name of the optical filter to use.

bright_limit
[astropy.units.Quantity, optional] Brightness in ABmag of the brightest point sources that we want to avoid saturating on, will be used to calculate a suitable shortest exposure time. Optional, but one and only one of `bright_limit` and `shortest_exp_time` must be specified.

shortest_exp_time
[astropy.units.Quantity, optional] Shortest sub exposure time to include in the sequence. Optional, but one and only one of `bright_limit` and `shortest_exp_time` must be specified.

longest_exp_time
[astropy.units.Quantity] Longest sub exposure time to include in the sequence.

faint_limit
[astropy.units.Quantity, optional] Brightness in ABmag of the faintest point sources that we want to be able to detect in the combined data from the sequence. Optional, but one and only one of `faint_limit` and `num_long_exp` must be specified.

num_long_exp
[int, optional] Number of repeats of the longest sub exposure to include in the sequence. Optional, but one and only one of `faint_limit` and `num_long_exp` must be specified.

exp_time_ratio
[float, optional] Ratio between successive sub exposure times in the HDR block, default 2.0

snr_target
[float, optional] Signal to noise ratio threshold for detection at `faint_limit`, default 5.0

Returns

exp_times
[astropy.units.Quantity] Sequence of sub exposure times

extended_source_etc(*surface_brightness*, *filter_name*, *snr_target*, *sub_exp_time*, *calc_type='per pixel'*, *saturation_check=True*, *binning=1*)

Calculates the total exposure time required to reach a given signal to noise ratio for a given extended source surface brightness.

Calculates the total exposure time required to reach a given signal to noise ratio for a given extended source surface brightness. Alternatively can calculate the required time for measurements of the sky background itself by setting the source surface brightness to None.

Parameters

surface_brightness
[astropy.units.Quantity or None] Surface brightness per arcsecond^2 of the source in AB-mag units, or an equivalent count rate in photo-electrons per second per pixel. Set to None or False to calculate the required exposure time for measurements of the sky background.

filter_name
[str] Name of the optical filter to use

snr_target
[astropy.units.Quantity] The desired signal to noise ratio, dimensionless unscaled units

sub_exp_time
[astropy.units.Quantity] length of individual sub-exposures

calc_type
[{'per pixel', 'per arcsecond squared'}] Calculation type, either signal to noise ratio per pixel or signal to noise ratio per arcsecond^2. Default is 'per pixel'

saturation_check
[bool, optional] If `True` will set the exposure time to zero where the electrons per pixel in a single sub-exposure exceed the saturation level. Default is `True`.

binning
[int, optional] Pixel binning factor. Cannot be used with calculation type 'per arcsecond squared', will raise `ValueError` if you try.

Returns

total_exp_time
[astropy.units.Quantity] Total exposure time required to reach a signal to noise ratio of at least `snr_target`, rounded up to an integer multiple of `sub_exp_time`.

extended_source_limit(*total_exp_time*, *filter_name*, *snr_target*, *sub_exp_time*, *calc_type='per pixel'*, *binning=1*, *enable_read_noise=True*, *enable_sky_noise=True*, *enable_dark_noise=True*)

Calculates the limiting extended source surface brightness for a given minimum signal to noise ratio and total exposure time.

Parameters

total_exp_time
[astropy.units.Quantity] Total length of all sub-exposures. If necessary will be rounded up to an integer multiple of `sub_exp_time`

filter_name
[str] Name of the optical filter to use

snr_target
[astropy.units.Quantity] The desired signal to noise ratio, dimensionless unscaled units

sub_exp_time
[astropy.units.Quantity] length of individual sub-exposures

calc_type
[{'per pixel', 'per arcsecond squared'}] Calculation type, either signal to noise ratio per pixel or signal to noise ratio per arcsecond^2. Default is 'per pixel'

binning
[int, optional] Pixel binning factor. Cannot be used with calculation type 'per arcsecond squared', will raise `ValueError` if you try.

enable_read_noise
[bool, optional] If `False` calculates limit as if read noise were zero, default `True`

enable_sky_noise
[bool, optional] If `False` calculates limit as if sky background were zero, default `True`

enable_dark_noise
[bool, optional] If False calculates limits as if dark current were zero, default `True`

Returns

surface_brightness
[astropy.units.Quantity] Limiting source surface brightness per arcsecond squared, in AB mag units.

extended_source_saturation_exp(*surface_brightness*, *filter_name*, *n_sigma=3.0*)
Calculates the maximum (sub) exposure time that will definitely avoid saturation for an extended source of given surface brightness.

Parameters

surface_brightness
[astropy.units.Quantity] Surface brightness per arcsecond^2 of the source in ABmag units, or an equivalent count rate in photo-electrons per second per pixel

filter_name
[str] Name of the optical filter to use

n_sigma
[float, optional] Safety margin to leave between the maximum expected electrons per pixel and the nominalsaturation level, in multiples of the noise, default 3.0

Returns

sub_exp_time
[astropy.units.Quantity] Maximum length of (sub) exposure that will definitely avoid saturation

extended_source_saturation_mag(*sub_exp_time*, *filter_name*, *n_sigma=3.0*)
Calculates the surface brightness of the brightest extended source that would definitely not saturate the image sensor in a given (sub) exposure time.

Parameters

sub_exp_time
[astropy.units.Quantity] Length of the (sub) exposure.

filter_name
[str] Name of the optical filter to use.

n_sigma
[float, optional] Safety margin to leave between the maximum expected electrons per pixel and the nominal saturation level, in multiples of the noise, default 3.0

Returns

surface_brightness
[astropy.units.Quantity] Surface brightness per arcsecond^2 of the brightest extended source that will definitely not saturate, in AB magnitudes.

extended_source_signal_noise(*surface_brightness*, *filter_name*, *total_exp_time*, *sub_exp_time*, *calc_type='per pixel'*, *saturation_check=True*, *binning=1*)

Calculates the signal and noise for an extended source with given surface brightness.

Calculates the signal and noise for an extended source with given surface brightness. Alternatively can calculate the signal and noise for measurements of the sky background itself by setting the source surface brightness to None.

Parameters

surface_brightness
[astropy.units.Quantity or callable or None] Surface brightness per arcsecond^2 of the source in ABmag units, or an equivalent count rate in photo-electrons per second per pixel, or a callable object that return surface brightness in spectral flux density units as a function of wavelength. Set to None or False to calculate the signal and noise for the sky background.

filter_name
[str] Name of the optical filter to use

total_exp_time
[astropy.units.Quantity] Total length of all sub-exposures. If necessary will be rounded up to an integer multiple of `sub_exp_time`

sub_exp_time
[astropy.units.Quantity] Length of individual sub-exposures

calc_type
[{'per pixel', 'per arcsecond squared'}] Calculation type, either signal & noise per pixel or signal & noise per arcsecond^2. Default is 'per pixel'

saturation_check
[bool, optional] If `True` will set both signal and noise to zero where the electrons per pixel in a single sub-exposure exceed the saturation level. Default is `True`.

binning
[int, optional] Pixel binning factor. Cannot be used with calculation type 'per arcsecond squared', will raise `ValueError` if you try.

Returns

signal
[astropy.units.Quantity] Total signal, units determined by calculation type.

noise: astropy.units.Quantity
Total noise, units determined by calculaton type.

extended_source_snr(*surface_brightness*, *filter_name*, *total_exp_time*, *sub_exp_time*, *calc_type='per pixel'*, *saturation_check=True*, *binning=1*)

Calculates the signal to noise ratio for an extended source with given surface brightness.

Calculates the signal to noise ratio for an extended source with given surface brightness. Alternatively can calculate the signal to noise for measurements of the sky background itself by setting the source surface

brightness to None.

Parameters

surface_brightness
[astropy.units.Quantity or callable or None] Surface brightness per arcsecond^2 of the source in ABmag units, or an equivalent count rate in photo-electrons per second per pixel, or a callable object that return surface brightness in spectral flux density units as a function of wavelength. Set to None or False to calculate the signal to noise ratio for the sky background.

filter_name
[str] Name of the optical filter to use

total_exp_time
[astropy.units.Quantity] Total length of all sub-exposures. If necessary will be rounded up to an integer multiple of `sub_exp_time`

sub_exp_time
[astropy.units.Quantity] Length of individual sub-exposures

calc_type
[{'per pixel', 'per arcsecond squared'}] Calculation type, either signal to noise ratio per pixel or signal to noise ratio per arcsecond^2. Default is 'per pixel'

saturation_check
[bool, optional] If `True` will set the signal to noise ratio to zero where the electrons per pixel in a single sub-exposure exceed the saturation level. Default is `True`.

binning
[int, optional] Pixel binning factor. Cannot be used with calculation type 'per arcsecond squared', will raise `ValueError` if you try.

Returns

snr
[astropy.units.Quantity] signal to noise ratio, dimensionless unscaled units

flux_to_ABmag(*flux*, *filter_name*)
Converts total flux of the target, integrated over the filter band, to magnitudes.

Parameters

flux
[astropy.units.Quantity] Total flux in Watts per square metre

filter_name
[str] Name of the optical filter to use

Returns

mag
[astropy.units.Quantity] Corresponding brightness of the target in AB magnitudes

Notes

The conversion between band averaged magnitudes and total flux depends somewhat on the spectrum of the source. For this calculation we assume F_ν is constant.

point_source_etc(*brightness*, *filter_name*, *snr_target*, *sub_exp_time*, *saturation_check=True*)

Calculates the total exposure time required to reach a given signal to noise ratio for a given point source brightness.

Parameters

brightness
[astropy.units.Quantity] Brightness of the source in ABmag units, or an equivalent count rate in photo-electrons per second.

filter_name
[str] Name of the optical filter to use

snr_target
[astropy.units.Quantity] The desired signal to noise ratio, dimensionless unscaled units

sub_exp_time
[astropy.units.Quantity] length of individual sub-exposures

saturation_check
[bool, optional] If `True` will set the exposure time to zero where the electrons per pixel in a single sub-exposure exceed the saturation level. Default is `True`.

Returns

total_exp_time
[astropy.units.Quantity] Total exposure time required to reach a signal to noise ratio of at least `snr_target`, rounded up to an integer multiple of `sub_exp_time`.

point_source_limit(*total_exp_time*, *filter_name*, *snr_target*, *sub_exp_time*, *enable_read_noise=True*, *enable_sky_noise=True*, *enable_dark_noise=True*)

Calculates the limiting point source surface brightness for a given minimum signal to noise ratio and total exposure time.

Parameters

total_exp_time
[astropy.units.Quantity] Total length of all sub-exposures. If necessary will be rounded up to an integer multiple of `sub_exp_time`

filter_name
[str] Name of the optical filter to use

snr_target
[astropy.units.Quantity] The desired signal to noise ratio, dimensionless unscaled units

sub_exp_time
[astropy.units.Quantity] length of individual sub-exposures

calc_type
[{'per pixel', 'per arcsecond squared'}] Calculation type, either signal to noise ratio per pixel or signal to noise ratio per arcsecond^2. Default is 'per pixel'

enable_read_noise
[bool, optional] If `False` calculates limit as if read noise were zero, default `True`

enable_sky_noise
[bool, optional] If `False` calculates limit as if sky background were zero, default `True`

enable_dark_noise
[bool, optional] If False calculates limits as if dark current were zero, default `True`

Returns

brightness
[astropy.units.Quantity] Limiting point source brightness, in AB mag units.

`point_source_saturation_exp`(*brightness*, *filter_name*, *n_sigma=3.0*)
Calculates the maximum (sub) exposure time that will definitely avoid saturation for point source of given brightness

Parameters

brightness
[astropy.units.Quantity] Brightness of the source in ABmag units, or an equivalent count rate in photo-electrons per second.

filter_name
[str] Name of the optical filter to use

n_sigma
[float, optional] Safety margin to leave between the maximum expected electrons per pixel and the nominalsaturation level, in multiples of the noise, default 3.0

Returns

sub_exp_time
[astropy.units.Quantity] Maximum length of (sub) exposure that will definitely avoid saturation

`point_source_saturation_mag`(*sub_exp_time*, *filter_name*, *n_sigma=3.0*)
Calculates the magnitude of the brightest point source that would definitely not saturate the image sensor in a given (sub) exposure time.

Parameters

sub_exp_time
[astropy.units.Quantity] Length of the (sub) exposure.

filter_name
[str] Name of the optical filter to use.

n_sigma
[float, optional] Safety margin to leave between the maximum expected electrons per pixel and the nominalsaturation level, in multiples of the noise, default 3.0

Returns

brighness
[astropy.units.Quantity] AB magnitude of the brightest point source that will definitely not saturate.

point_source_signal_noise(*brightness*, *filter_name*, *total_exp_time*, *sub_exp_time*, *saturation_check=True*)

Calculates the signal and noise for a point source of a given brightness, assuming PSF fitting photometry

The returned signal and noise values are the weighted sum over the pixels in the source image, where the weights are the normalised pixel values of the PSF model being fit to the data.

Parameters

brightness
[astropy.units.Quantity] Brightness of the source in ABmag units, or an equivalent count rate in photo-electrons per second.

filter_name
[str] Name of the optical filter to use

total_exp_time
[astropy.units.Quantity] Total length of all sub-exposures. If necessary will be rounded up to an integer multiple of `sub_exp_time`

sub_exp_time
[astropy.units.Quantity] Length of individual sub-exposures

calc_type
[{'per pixel', 'per arcsecond squared'}] Calculation type, either signal & noise per pixel or signal & noise per arcsecond^2. Default is 'per pixel'

saturation_check
[bool, optional] If `True` will set both signal and noise to zero where the electrons per pixel in a single sub-exposure exceed the saturation level. Default is `True`.

Returns

signal
[astropy.units.Quantity] Effective total signal in units of electrons

noise: astropy.units.Quantity
Effective total noise in units of electrons

Notes

The PSF fitting signal to noise calculations follow the example of http://www.stsci.edu/itt/review/ihb_cy14.WFPC2/ch6_exposuretime6.html

The values will depend on the position of the centre of the PSF relative to the pixel grid, this calculation assumes the worst case of PSF centres on a pixel corner. Conversely it optimistically assumes that the PSF model exactly matches the PSF of the data.

point_source_snr(*brightness*, *filter_name*, *total_exp_time*, *sub_exp_time*, *saturation_check=True*)

Calculates the signal to noise ratio for a point source of a given brightness, assuming PSF fitting photometry

The returned signal to noise ratio refers to the weighted sum over the pixels in the source image, where the weights are the normalised pixel values of the PSF model being fit to the data.

Parameters

brightness
[astropy.units.Quantity] Brightness of the source in ABmag units, or an equivalent count rate in photo-electrons per second.

filter_name
[str] Name of the optical filter to use

total_exp_time
[astropy.units.Quantity] Total length of all sub-exposures. If necessary will be rounded up to an integer multiple of `sub_exp_time`

sub_exp_time
[astropy.units.Quantity] Length of individual sub-exposures

calc_type
[{'per pixel', 'per arcsecond squared'}] Calculation type, either signal & noise per pixel or signal & noise per arcsecond^2. Default is 'per pixel'

saturation_check
[bool, optional] If `True` will set both signal and noise to zero where the electrons per pixel in a single sub-exposure exceed the saturation level. Default is `True`.

Returns

snr
[astropy.units.Quantity] signal to noise ratio dimensionless unscaled units

rate_to_ABmag(*rate*, *filter_name*)
Converts photo-electrons per second in the image sensor to AB magnitudes

Parameters

rate
[astropy.units.Quantity] Photo-electrons per second

filter_name
[str] Name of the optical filter to use

Returns

mag
[astropy.units.Quantity] Corresponding source brightness in AB magnitudes

rate_to_SB(*SB_rate*, *filter_name*)
Converts photo-electrons per pixel per second to surface brightness AB magnitudes (per arcsecond squared)

Parameters

SB_rate
[astropy.units.Quantity] Photo-electrons per pixel per second

filter_name
[str] Name of the optical filter to use

Returns

mag
[astropy.units.Quantity] Corresponding source surface brightness in AB magnitudes

Notes

At the time of writing `astropy.units` did not support the commonly used (but dimensionally nonsensical) expression of surface brightness in 'magnitudes per arcsecond squared'. Consequently the `mag` surface brightness return value has units of `astropy.unit.ABmag`, the 'per arcsecond squared' is implied.

snr_vs_ABmag(*exp_times*, *filter_name*, *magnitude_interval=<Magnitude 0.02 mag(AB)>*, *snr_target=1.0*, *plot=None*)

Calculates PSF fitting signal to noise ratio as a function of point source brightness for the combined data resulting from a given sequence of sub exposures.

Optionally generates a plot of the results. Automatically choses limits for the magnitude range based on the saturation limit of the shortest exposure and the sensitivity limit of the combined data.

Parameters

exp_times
[astropy.units.Quantity] Sequence of sub exposure times.

filter_name
[str] Name of the optical filter to use.

magnitude_interval
[astropy.units.Quantity, optional] Step between consecutive brightness values, default 0.02 mag

snr_target
[float, optional] signal to noise threshold used to set faint limit of magnitude range, default 1.0

plot
[str, optional] Filename for the plot of SNR vs magnitude. If not given no plots will be generated.

Returns

magnitudes
[astropy.units.Quantity] Sequence of point source brightnesses in AB magnitudes

snrs
[astropy.units.Quantity] signal to noise ratios for point sources of the brightnesses in `magnitudes`

total_elapsed_time(*exp_list*)

Calculates the total elapsed time required for a given a list of sub-exposure times.

The calculation add readout time overheads (but no others, at present) and sums the elapsed time from all sub-exposures.

Parameters

exp_list
[astropy.units.Quantity] List of sub-exposure times

Returns

elapsed_time
[astropy.units.Quantity] Total elapsed time required to execute the list of sub exposures

`total_exposure_time`(*total_elapsed_time*, *sub_exp_time*)
Calculates total exposure time given a total elapsed time and sub-exposure time.

The calculation includes readout time overheads (but no others, at present) and rounds down to an integer number of sub-exposures.

Parameters

total_elapsed_time
[astropy.units.Quantity] Total elapsed time

sub_exp_time
[astropy.units.Quantity] Exposure time of individual sub-exposures

Returns

total_exposure_time
[astropy.units.Quantity] Maximum total exposure time possible in an elapsed time of no more than `total_elapsed_time`

CHAPTER 7

gunagala.optic Module

Optics, e.g. a telescope or lens

7.1 Functions

list_surfaces()	
make_throughput(surfaces)	Constructs a table of throughout vs wavelength from the numbers and types of optical surfaces

7.1.1 list_surfaces

gunagala.optic.**list_surfaces**()

7.1.2 make_throughput

gunagala.optic.**make_throughput**(*surfaces*)

Constructs a table of throughout vs wavelength from the numbers and types of optical surfaces

Utility function to provide a simple estimate of optical throughput versus wavelength given the numbers and types of optical surfaces. Uses tabulated data included within gunagala for a number of common optical surfaces. To get a list of the available surface types call list_surfaces(). Mirror coating data from by Thorlabs (http://www.tholabs.com/)

Parameters

surfaces: list of tuples
List containing tuples of (surface name,)

Returns

table: astropy.table.Table
Table with columns Wavelength and Throughput

7.2 Classes

Optic(aperture, focal_length, throughput[, ...])	Class representing the overall optical system.

7.2.1 Optic

class gunagala.optic.**Optic**(*aperture*, *focal_length*, *throughput*, *central_obstruction=<Quantity 0. mm>*)
Bases: object

Class representing the overall optical system.

The optical system includes all optics (e.g. telescope, including any field flattener, focal reducer or reimaging optics) with the exception of optical filters, which are handled by the `optical_filter.Filter` class. At present only circular pupils (aperture) and are supported, but central obstructions (also circular) can be modelled.

Parameters

aperture
[astropy.units.Quantity] Diameter of the optical system entrance pupil (effective aperture diameter).

focal_length
[astropy.units.Quantity] Effective focal length of the optical system as a whole.

throughput
[astropy.table.Table or str] Optical throughput as a function of wavelength data, either as an astropy.table.Table object or the name of a file that can be read by `astropy.table.Table.read()`. The filename can be either the path to a user file or the name of one of gunagala's included files. The table must use column names `Wavelength` and `Throughput`. If the table does not specify units then nm and dimensionless unscaled are assumed.

central_obstruction
[astropy.units.Quantity, optional] Diameter of the central obstruction of the entrance pupil, if any. If not specified an unobstructed pupil is assumed.

Attributes

aperture
[astropy.units.Quantity] Same as parameters.

central_obstruction
[astropy.units.Quantity] Same as parameters.

aperture_area
[astropy.units.Quantity] Effective collecting are of the optical system aperture, including the effects of the central obstruction, if any.

focal_length
[astropy.units.Quantity] Same as parameters.

focal_ratio
[astropy.units.Quantity] Effective focal ratio (F/D) of the optical system

theta_range
[astropy.units.Quantity] Pair of angles corresponding to the minimum and maximum angles of incidence in focal plane of the optical system. These can be used by `optical_filter.Filter` objects to calculate cone angle effects for focal plane filters. These are automatically calculated from the central obstruction and entrance pupil diameters and effective focal length assuming a telecentric output. If the optical system is far telecentricty these values should now be used.

wavelengths
[astropy.units.Quantity] Sequence of wavelength values from the tabulated throughput data, loaded from `throughput_filename`.

throughput
[astropy.units.Quantity] Sequence of throughput values from the tabulated throughput data, loaded from `throughout_filename`.

CHAPTER 8

gunagala.optical_filter Module

Optical filters

8.1 Functions

butter_band(w, w1, w2, N[, scale])	Simple Butterworth bandpass filter function in wavelength space.
cheby_band(w, w1, w2, N[, ripple, scale])	Simple Chebyshev Type I bandpass filter function in wavelength space.

8.1.1 butter_band

gunagala.optical_filter.**butter_band**(*w*, *w1*, *w2*, *N*, *scale=0.95*)

Simple Butterworth bandpass filter function in wavelength space.

Parameters

w
[astropy.units.Quantity] Wavelength

w1
[astropy.units.Quantity] Wavelength of short wavelength edge of bandpass

w2
[astropy.units.Quantity] Wavelength of long wavelength edge of bandpass

N
[int] Order of the Butterworth function

scale
[float, optional] Scaling to apply to the transmission of the Butterworth function, default 0.95

Returns

transmission
[astropy.units.Quantity] Filter transmission at wavelength `w`.

8.1.2 cheby_band

gunagala.optical_filter.**cheby_band**(*w*, *w1*, *w2*, *N*, *ripple=1*, *scale=0.95*)
Simple Chebyshev Type I bandpass filter function in wavelength space.

Parameters

w
[astropy.units.Quantity] Wavelength

w1
[astropy.units.Quantity] Wavelength of short wavelength edge of bandpass

w2
[astropy.units.Quantity] Wavelength of long wavelength edge of bandpass

N
[int] Order of the Chebyshev function

ripple
[float, optional] Scaling to apply to the ripple of the Chebyshev function, default 1.0

scale
[float, optional] Scaling to apply to the transmission of the Chebyshev function, default 0.95

Returns

transmission
[astropy.units.Quantity] Filter transmission at wavelength `w`.

8.2 Classes

`Filter`([transmission, chebyshev_params, ...])	Class representing an optical bandpass filter.

8.2.1 Filter

class gunagala.optical_filter.**Filter**(*transmission=None*, *chebyshev_params=None*, *butterworth_params=None*, *apply_aoi=False*, *n_eff=1.75*, *theta_range=None*, **kwargs*)
Bases: `object`

Class representing an optical bandpass filter.

The filter bandpass can be defined either by a table of transmission versus wavelength data or by one of the included analytic functions: Butterworth function or Chebyshev Type I.

Parameters

transmission
[astropy.table.Table or str, optional] Filter transmission as a function of wavelength data, either as an astropy.table.Table object or the name of a file that can be read by `astropy.table.Table.read()`. The filename can be either the path to a user file or the name of one of gunagala's included files. The table must use column names `Wavelength` and `Transmission`. If the table does not specify units then nm and dimensionless unscaled are assumed.

chebyshev_params
[dict, optional] Dictionary containing the parameters wave1, wave2, order, ripple and peak for the Chebyshev Type I parameterised filter model.

butterworth_params
[dict, optional] Dictionary containing the parameters wave1, wave2, order and peak for the Butterworth parameterised filter model.

apply_aoi
[bool, optional] Whether to model angle of incidence effects due to installation of the filter in a converging beam. If the filter is to be installed in a pupil or the transmission profile already includes these effects this should be set to False. If set to True then calls to the `transmission()` method will have to specify the range of angles of incidence. Default False

n_eff
[float, optional] Effective refractive index value for the filter coatings. This is used only by the angle of incidence effect model. Typical values for real interference filters range from ~1.5 to ~2, and are in general polarisation dependent (not modelled here). Default 1.75.

theta_range
[astropy.units.Quantity, optional] 2 element Quantity specifying the range of angles of incidence (min, max). If specified this will be used to model the effect of a converging beam on the calculated filter parameters (FWHM, lambda_c, etc).

Attributes Summary

`FWHM`	Full-Width at Half Maximum of the filter
`lambda_c`	Central wavelength of the filter, defined here as the mid point between the two wavelengths where transmission is half peak transmission.
`lambda_peak`	Wavelength at peak transmission of the filter
`peak`	Peak transmission of the filter
`theta_range`	2 element Quantity specifying the range of angles of incidence (min, max).

Methods Summary

`transmission`(waves[, theta_range])	Return filter transmission at the given wavelength(s).

Attributes Documentation

FWHM
Full-Width at Half Maximum of the filter

Returns

FWHM
[astropy.units.Quantity] FWHM of the filter transmission profile

`lambda_c`
Central wavelength of the filter, defined here as the mid point between the two wavelengths where transmission is half peak transmission.

Returns

lambda_c
[astropy.units.Quantity] Central wavelength

`lambda_peak`
Wavelength at peak transmission of the filter

Returns

lambda_peak
[astropy.units.Quantity] Wavelength at peak transmission

`peak`
Peak transmission of the filter

Returns

peak
[astropy.units.Quantity] Peak transmission of the filter in dimensionless unscaled units

`theta_range`
2 element Quantity specifying the range of angles of incidence (min, max).

Returns

theta_range: astropy.units.Quantity
2 element Quantity specifying the range of angles of incidence (min, max).

Methods Documentation

`transmission`(*waves*, *theta_range=None*)
Return filter transmission at the given wavelength(s).

For filter bandpasses defined by data tables this will interpolate/extrapolate as required while for filter bandpasses defined by analytic expressions it will be calculated directly.

Parameters

waves
[astropy.units.Quantity] Wavelength(s) for which the filter transmission is required

theta_range
[astropy.units.Quantity, optional] 2 element quantity specifying the range of angles of incidence (min, max). If specified this will be used to model the effect of a converging

beam on the filter bandpass. If not specified the default values set when creating the Filter instance will be used.

Returns

waves
[astropy.units.Quantity] Filter transmission at the given wavelength(s)

CHAPTER 9

gunagala.camera Module

Cameras (stricly the image sensor subsystem, not including optics, optical filters, etc)

9.1 Classes

Camera(bit_depth, full_well, gain, bias, ...)	Class representing a camera.

9.1.1 Camera

class gunagala.camera.**Camera**(*bit_depth*, *full_well*, *gain*, *bias*, *readout_time*, *pixel_size*, *resolution*, *read_noise*, *dark_current*, *QE*, *minimum_exposure*)

Bases: object

Class representing a camera.

Here 'camera' refers to the image sensor, associated electronics, shutter, etc., but does not include any of the optical components of the system.

Parameters

bit_depth
[int] Bits per pixel used by the camera analogue to digital converters.

full_well
[astropy.units.Quantity] Number of photo-electrons each pixel can receive before saturating.

gain
[astropy.units.Quantity] Number of photo-electrons corresponding to one ADU in the digital data.

bias
[astropy.units.Quantity] Bias level of image sensor, in ADU / pixel units. Used when determining saturation level.

readout_time
[astropy.units.Quantity] Time required to read the data from the image sensor.

pixel_size
[astropy.units.Quantity] Pixel pitch. Square pixels are assumed.

Resolution
[astropy.units.Quantity] Two element Quantity containing the number of pixels across the image sensor in both horizontal & vertical directions.

read_noise astropy.units.Quantity
Intrinsic noise of image sensor and readout electronics, in electrons/pixel units.

dark_current
[astropy.units.Quantity] Rate of accumlation of dark signal, in electrons/second/pixel units.

QE
[astropy.table.Table or str] Quantum efficiency as a function of wavelength data, either as an astropy.table.Table object or the name of a file that can be read by `astropy.table.Table.read()`. The filename can be either the path to a user file or the name of one of gunagala's included files. The table must use column names `QE` andv'Throughput'. If the table does not specify units then nm and electron / photon are assumed.

minimum_exposure
[astropy.units.Quantity] Length of the shortest exposure that the camera is able to take.

Attributes

bit_depth
[int] Same as parameters

full_well
[astropy.units.Quantity] Same as parameters

gain
[astropy.units.Quantity] Same as parameters

bias
[astropy.units.Quantity] Same as parameters

readout_time
[astropy.units.Quantity] Same as parameters

pixel_size
[astropy.units.Quantity] Same as parameters

resolution
[astropy.units.Quantity] Same as parameters

read_noise
[astropy.units.Quantity] Same as parameters

dark_current
[astropy.units.Quantity] Same as parameters

minimum_exposure
[astropy.units.Quantity] Same as parameters

saturation_level
[astropy.units.Quantity] Lowest of `full_well` and `2**bit_depth - 1 - bias`

max_noise
[astropy.units.Quantity] Poisson + readout noise corresponding to `saturation_level`

wavelenghts
[astropy.units.Quantity] Sequence of wavelengths from the QE data

QE
[astropy.units.Quantity] Sequence of quantum efficiency values from the QE data.

CHAPTER 10

gunagala.psf Module

Point spread functions

10.1 Classes

FittablePSF(FWHM[, pixel_scale])	Base class representing a 2D point spread function based on a Fittable2DModel from astropy.modelling.
MoffatPSF([model, shape])	Class representing a 2D Moffat profile point spread function.
PSF	Abstract base class representing a 2D point spread function.
PixellatedPSF(psf_data, psf_sampling[, ...])	Class representing a 2D point spread function based on an already pixellated data, e.g.

10.1.1 FittablePSF

class gunagala.psf.**FittablePSF**(*FWHM*, *pixel_scale=None*, ***kwargs*)

Bases: gunagala.psf.PSF, astropy.modeling.Fittable2DModel

Base class representing a 2D point spread function based on a Fittable2DModel from astropy.modelling.

Used to calculate pixelated version of the PSF and associated parameters useful for point source signal to noise and saturation limit calculations.

Parameters

FWHM
[astropy.units.Quantity] Full Width at Half-Maximum of the PSF in angle on the sky units.

pixel_scale
[astropy.units.Quantity, optional] Pixel scale (angle/pixel) to use when calculating pixellated point spread functions or related parameters. Does not need to be set on object creation but must be set before before pixellation function can be used.

Attributes Summary

FWHM	Full Width at Half-Maximum of the PSF.

Methods Summary

pixellated([size, offsets])	Calculates a pixellated version of the PSF.

Attributes Documentation

FWHM
Full Width at Half-Maximum of the PSF.

Returns

FWHM
[astropy.units.Quantity] Full Width at Half-Maximum in angle on the sky units.

Methods Documentation

pixellated(*size=21*, *offsets=(0.0*, *0.0)*)
Calculates a pixellated version of the PSF.

The pixel values are calculated using 10x oversampling, i.e. by evaluating the continuous PSF model at a 10 x 10 grid of positions within each pixel and averaging the results.

Parameters

size
[int, optional] Size of the pixellated PSF to calculate, the returned image will have `size` x `size` pixels. Default value 21.

offset
[tuple of floats, optional] y and x axis offsets of the centre of the PSF from the centre of the returned image, in pixels.

Returns

pixellated
[numpy.array] Pixellated PSF image with `size` by `size` pixels. The PSF is normalised to an integral of 1 however the sum of the pixellated PSF will be somewhat less due to truncation of the PSF wings by the edge of the image.

10.1.2 MoffatPSF

class gunagala.psf.**MoffatPSF**(*model=None*, *shape=2.5*, ***kwargs*)
Bases: gunagala.psf.FittablePSF, astropy.modeling.functional_models.Moffat2D

Class representing a 2D Moffat profile point spread function.

Used to calculate pixelated version of the PSF and associated parameters useful for point source signal to noise and saturation limit calculations.

Parameters

FWHM
[astropy.units.Quantity] Full Width at Half-Maximum of the PSF in angle on the sky units

shape
[float, optional] Shape parameter of the Moffat function, must be > 1, default 2.5

pixel_scale
[astropy.units.Quantity, optional] Pixel scale (angle/pixel) to use when calculating pixellated point spread functions or related parameters. Does not need to be set on object creation but must be set before before pixellation function can be used.

Notes

Smaller values of the shape parameter correspond to 'wingier' profiles. A value of 4.765 would give the best fit to pure Kolmogorov atmospheric turbulence. When instrumental effects are added a lower value is appropriate. IRAF uses a default of 2.5.

Attributes

n_pix: astropy.units.Quantity
Effective number of pixels

Attributes Summary

param_names	
shape	Shape parameter of the Moffat function, see Notes.

Attributes Documentation

param_names = ('amplitude', 'x_0', 'y_0', 'gamma', 'alpha')

shape
Shape parameter of the Moffat function, see Notes.

Returns

shape
[float] Shape parameter value.

10.1.3 PSF

class gunagala.psf.**PSF**
Bases: object

Abstract base class representing a 2D point spread function.

Used to calculate pixelated version of the PSF and associated parameters useful for point source signal to noise and saturation limit calculations.

Attributes Summary

n_pix	The PSF's effective number of pixels for the worse case where the PSF is centred on the corner of a pixel.
peak	The maximum fraction of the total signal that can fall in a single pixel.
pixel_scale	Pixel scale used when calculating pixellated point spread functions or related parameters.

Attributes Documentation

n_pix

The PSF's effective number of pixels for the worse case where the PSF is centred on the corner of a pixel.

The effective number of pixels is for signal to noise calculations for PSF fitting photometry. The signal to noise for PSF fitting photometry is the same as if the signal were evenly distributed over this many pixels.

Returns

n_pix
[astropy.units.Quantity] Effective number of pixels

peak

The maximum fraction of the total signal that can fall in a single pixel.

This is simply the central pixel value of the PSF when it is perfectly centred on a pixel centre. This is useful for saturation limit calculations.

Returns

peak
[astropy.units.Quantity] Maximum fraction of the total signal that can fall in a single pixel, in 1/pixel units.

pixel_scale

Pixel scale used when calculating pixellated point spread functions or related parameters.

Returns

pixel_scale
[astropy.units.Quantity] Pixel scale in angle on the sky per pixel units.

10.1.4 PixellatedPSF

class gunagala.psf.**PixellatedPSF**(*psf_data*, *psf_sampling*, *psf_centre=None*, *oversampling=10*, *pixel_scale=None*, *renormalise=True*, ***kwargs*)

Bases: gunagala.psf.PSF

Class representing a 2D point spread function based on an already pixellated data, e.g. a PSF calculated with optical design software.

Used to calculate pixelated version of the PSF and associated parameters useful for point source signal to noise and saturation limit calculations.

Parameters

psf_data: numpy.array
Pixellated PSF data.

psf_sampling: astropy.units.Quantity
Pixel scale (angle/pixel) of psf_data.

psf_centre: (float, float), optional
Pixel coordinates of the PSF centre within psf_data (zero based, (y, x)). If not given psf_data.shape / 2 will be assumed.

oversampling
[integer, optional] Oversampling factor used when shifting & resampling the PSF, default 10.

pixel_scale
[astropy.units.Quantity, optional] Pixel scale (angle/pixel) to use when calculating pixellated point spread functions or related parameters. Does not need to be set on object creation but must be set before before pixellation function can be used.

renormalise: bool, optional
Whether to renormalise the PSF to a total of 1 during initialisation, default True. Only set to False if the psf_data is already correctly normalised.

Methods Summary

pixellated([size, offsets])	Calculates a pixellated version of the PSF.

Methods Documentation

pixellated(*size=21*, *offsets=(0.0, 0.0)*)
Calculates a pixellated version of the PSF.

The pixel values are calculated by shifting and resampling the original psf_data, then binning by the oversampling factor.

Parameters

size
[int, optional] Size of the pixellated PSF to calculate, the returned image will have `size` x `size` pixels. Default value 21.

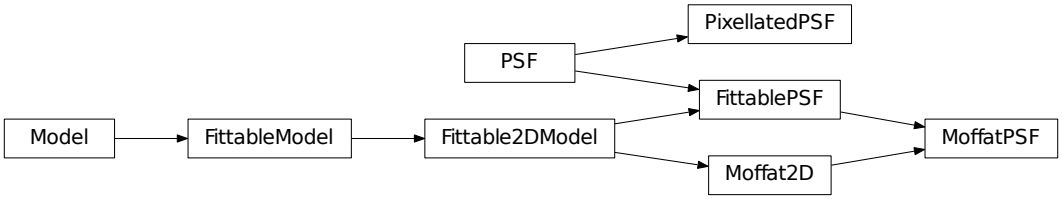
offsets
[tuple of floats, optional] y and x axis offsets of the centre of the PSF from the centre of the returned image, in pixels.

Returns

pixellated
[numpy.array] Pixellated PSF image with `size` by `size` pixels. The PSF is normalised

to an integral of 1 however the sum of the pixellated PSF will be somewhat less due to truncation of the PSF wings by the edge of the image.

10.2 Class Inheritance Diagram



CHAPTER 11

gunagala.sky Module

Sky background models

11.1 Classes

Simple(surface_brightness, **kwargs)	Simple sky background model using fixed, uniform, pre-determined surface brightness values for specific filter bands.
Sky()	Abstract base class for sky background models.
ZodiacalLight(**kwargs)	Class representing the Zodiacal Light sky background.

11.1.1 Simple

class gunagala.sky.**Simple**(*surface_brightness*, ***kwargs*)

Bases: gunagala.sky.Sky

Simple sky background model using fixed, uniform, pre-determined surface brightness values for specific filter bands.

Parameters

surface_brightness
[dict] Dictionary containing filter_name, surface brightness key, value pairs. The surface brightnesses should be astropy.units.Quantity objects in ABmag units (the 'per arcsecond' is assumed).

Methods Summary

surface_brightness(filter_name)	Returns the pre-determined sky surface brightness value for a given named filter.

Methods Documentation

surface_brightness(*filter_name*)
Returns the pre-determined sky surface brightness value for a given named filter.

Parameters

filter_name
[str] Name of the optical filter to use.

Returns

surface_brightness
[astropy.units.Quantity] Sky surface brightness in ABmag units (the 'per square arcsecond' is implied).

11.1.2 Sky

class gunagala.sky.**Sky**
Bases: object

Abstract base class for sky background models.

Methods Summary

surface_brightness(**kwargs)

Methods Documentation

surface_brightness(***kwargs*)

11.1.3 ZodiacalLight

class gunagala.sky.**ZodiacalLight**(***kwargs*)
Bases: gunagala.sky.Sky

Class representing the Zodiacal Light sky background.

Includes methods that return the absolute surface brightness spectral flux density at the ecliptic poles as well as the relative brightness variations as a function of position on the sky.

Attributes

waves
[astropy.units.Quantity] Sequence of wavelengths used by the internal model

self.sfd
[astropy.units.Quantity] Sequence of ecliptic pole surface brightness spectral flux density values corresponding to the wavelengths in `waves`, in energy flux per unit wavelength units.

self.photon_sfd
[astropy.units.Quantity] Sequence of ecliptic pole surface brightness spectral flux density values corresponding to the wavelengths in `waves`, in photon flux per unit wavelength units.

Methods Summary

`relative_brightness`(position, time)	Calculate the Zodiacal Light surface brightness relative to that at the ecliptic poles for a given sky position and observing time.
`surface_brightness`(**kwargs)	Generates a function that will calculate Zodiacal Light ecliptic pole surface brightness as a function of wavelength, in photon spectral flux density units.

Methods Documentation

relative_brightness(*position*, *time*)
Calculate the Zodiacal Light surface brightness relative to that at the ecliptic poles for a given sky position and observing time.

At present to model includes the annual rotation of the Zodiacal Light distribution as the Earth orbits the Sun but does not include second order effects due to the inclination of the Earth's orbit relative to the mid plane of the Zodiacal dust disc.

Parameters

position
[astropy.coordinates.SkyCoord or str] Sky position(s) in the form of either a `astropy.coordinates.SkyCoord` object or a string that can be converted into one.

time
[astropy.time.Time or str] Time of observation in the form of either an `astropy.time.Time` or a string that can be converted into one.

Returns

rel_SB
[numpy.array] Relative sky brightness of the Zodiacal light at the given sky position(s)

surface_brightness(***kwargs*)
Generates a function that will calculate Zodiacal Light ecliptic pole surface brightness as a function of wavelength, in photon spectral flux density units.

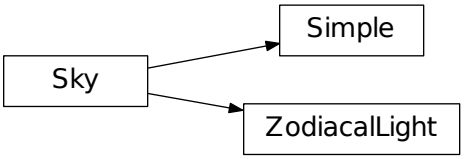
The returned function take a single `astropy.units.Quantity` parameter, the wavelength(s) at which the Zodiacal Light surface brightness is required. It returns an `astropy.units.Quantity` object containing the corresponding surface brightness values. See `imager.Imager` for a usage example.

Returns

surface_brightness

[callable] Function that calculates the ecliptic pole surface brightness for arbitrary wavelength(s).

11.2 Class Inheritance Diagram



CHAPTER 12

gunagala.utils Package

12.1 Functions

array_sequence_equal(array_sequence[, reference])	Determine if all array objects in a sequence are equal.
bin_array(data, binning_factor[, bin_func])	Bin 2D array data by a given factor using a given binning function.
ensure_unit(arg, unit)	Ensures that the argument has the requested units, performing conversions as necessary.
get_table_data(data_table, column_names, ...)	Parses a data table to extract specified columns, converted to Quantity with specified units.

12.1.1 array_sequence_equal

gunagala.utils.**array_sequence_equal**(*array_sequence*, *reference=None*)

Determine if all array objects in a sequence are equal.

Parameters

array_sequence: sequence of numpy.array
Sequence of numpy.array or compatible type (e.g. astropy.units.Quantity) objects to compare. The objects must support element-wise comparision and implement an any() method.

reference: numpy.array, optional
If given all arrays in the sequence will be compared with reference, otherwise they will be compared with each other. Must be a numpy.array or compatible type (e.g. Quantity).

Returns

equal: bool
True if all arrays in the sequence are equal (or equal to reference, if given), otherwise False.

12.1.2 bin_array

gunagala.utils.**bin_array**(*data*, *binning_factor*, *bin_func=<function sum>*)

Bin 2D array data by a given factor using a given binning function.

Parameters

data: numpy.array
Array to be binned.

binning_factor: int
Size of the binning regions, in pixels.

bin_func: function, optional
Function to be used to combine the pixel values within each binning region. The function must accept a numpy.array as the first argument, and accept an axis keyword argument to specify which array axis to peform the combination on. Default numpy.sum().

Returns

binned: numpy.array
Binned array.

12.1.3 ensure_unit

gunagala.utils.**ensure_unit**(*arg*, *unit*)

Ensures that the argument has the requested units, performing conversions as necessary.

Parameters

arg
[astropy.units.Quantity or compatible] Argument to be coerced into the requested units. Can be an `astropy.units.Quantity` instance or any numeric type or sequence that is compatible with the `Quantity` constructor (e.g. a `numpy.array`, `list` of `float`, etc.).

unit
[astropy.units.Unit] Requested units.

Returns

arg
[astropy.units.Quantity] `arg` as an `astropy.units.Quantity` with units of `unit`.

12.1.4 get_table_data

gunagala.utils.**get_table_data**(*data_table*, *column_names*, *column_units*, *data_dir='data/performance_data'*, ***kwargs*)

Parses a data table to extract specified columns, converted to Quantity with specified units.

Parameters

data_table: astropy.table.Table or str
The data table for parsing, either as an astropy.table.Table object or the name of a file that

can be read by `astropy.table.Table.read()`. The filename can be either the path to a user file or the name of one of gunagala's included files.

column_names: sequence
Names of the columns to extract from the table

column_units: sequence
Desired units for the extracted columns. If data_table specifies units for its columns then the extracted columns will be converted to these units. If not then the specified units will be added to the corresponding column.

Additional keyword arguments will be passed to the call to astropy.table.Table.read()

if reading a Table from a file. See the documentation for Table.read() for details

of the available parameters.

Returns

data: tuple of astropy.units.Quantity
Tuple of Quantity objects corresponding to the named columns, with the specified units.

Python Module Index

g

gunagala.camera, 55
gunagala.imager, 29
gunagala.optic, 45
gunagala.optical_filter, 49
gunagala.psf, 59
gunagala.sky, 65
gunagala.utils, 69

Index

A

ABmag_to_flux() (gunagala.imager.Imager method), 32
ABmag_to_rate() (gunagala.imager.Imager method), 33
array_sequence_equal() (in module gunagala.utils), 69

B

bin_array() (in module gunagala.utils), 70
butter_band() (in module gunagala.optical_filter), 49

C

Camera (class in gunagala.camera), 55
cheby_band() (in module gunagala.optical_filter), 50
create_imagers() (in module gunagala.imager), 29

E

ensure_unit() (in module gunagala.utils), 70
exp_time_sequence() (gunagala.imager.Imager method), 34
extended_source_etc() (gunagala.imager.Imager method), 34
extended_source_limit() (gunagala.imager.Imager method), 35
extended_source_saturation_exp() (gunagala.imager.Imager method), 36
extended_source_saturation_mag() (gunagala.imager.Imager method), 36
extended_source_signal_noise() (gunagala.imager.Imager method), 37
extended_source_snr() (gunagala.imager.Imager method), 37

F

Filter (class in gunagala.optical_filter), 50
FittablePSF (class in gunagala.psf), 59
flux_to_ABmag() (gunagala.imager.Imager method), 38
FWHM (gunagala.optical_filter.Filter attribute), 51
FWHM (gunagala.psf.FittablePSF attribute), 60

G

get_table_data() (in module gunagala.utils), 70
gunagala.camera (module), 55
gunagala.imager (module), 29
gunagala.optic (module), 45
gunagala.optical_filter (module), 49
gunagala.psf (module), 59
gunagala.sky (module), 65
gunagala.utils (module), 69

I

Imager (class in gunagala.imager), 30

L

lambda_c (gunagala.optical_filter.Filter attribute), 52
lambda_peak (gunagala.optical_filter.Filter attribute), 52
list_surfaces() (in module gunagala.optic), 45

M

make_throughput() (in module gunagala.optic), 45
MoffatPSF (class in gunagala.psf), 60

N

n_pix (gunagala.psf.PSF attribute), 62

O

Optic (class in gunagala.optic), 46

P

param_names (gunagala.psf.MoffatPSF attribute), 61
peak (gunagala.optical_filter.Filter attribute), 52
peak (gunagala.psf.PSF attribute), 62
pixel_scale (gunagala.psf.PSF attribute), 62
pixellated() (gunagala.psf.FittablePSF method), 60
pixellated() (gunagala.psf.PixellatedPSF method), 63
PixellatedPSF (class in gunagala.psf), 62
point_source_etc() (gunagala.imager.Imager method), 39
point_source_limit() (gunagala.imager.Imager method), 39

point_source_saturation_exp() (gunagala.imager.Imager method), 40
point_source_saturation_mag() (gunagala.imager.Imager method), 40
point_source_signal_noise() (gunagala.imager.Imager method), 40
point_source_snr() (gunagala.imager.Imager method), 41
PSF (class in gunagala.psf), 61

R

rate_to_ABmag() (gunagala.imager.Imager method), 42
rate_to_SB() (gunagala.imager.Imager method), 42
relative_brightness() (gunagala.sky.ZodiacalLight method), 67

S

SB_to_rate() (gunagala.imager.Imager method), 33
shape (gunagala.psf.MoffatPSF attribute), 61
Simple (class in gunagala.sky), 65
Sky (class in gunagala.sky), 66
snr_vs_ABmag() (gunagala.imager.Imager method), 43
surface_brightness() (gunagala.sky.Simple method), 66
surface_brightness() (gunagala.sky.Sky method), 66
surface_brightness() (gunagala.sky.ZodiacalLight method), 67

T

test() (in module gunagala), 27
theta_range (gunagala.optical_filter.Filter attribute), 52
total_elapsed_time() (gunagala.imager.Imager method), 43
total_exposure_time() (gunagala.imager.Imager method), 44
transmission() (gunagala.optical_filter.Filter method), 52

Z

ZodiacalLight (class in gunagala.sky), 66