Sinistra Documentation

Release 0.1

Gillen Brown

May 24, 2016

Contents

Sinistra is a Python library of functions I've found useful in my time doing astronomy. They may or may not be useful to anyone else.

The name Sinistra comes from the name of the astronomy professor at Hogwarts in the Harry Potter universe, Professor Sinistra.

Contents:

CHAPTER 1

Utilities

sinistra.utilities.**ab_to_vega**(*ab_mag*, *band*)
Converts a AB mag to Vega.

This basically contains a bunch of AB-Vega conversions, so I don't have to look them up every time I want to do anything.

Parameters

- **ab_mag** (*float*) – magnitude in the AB system to be converted to Vega.
- **band** (*str*) – Band the magnitude is in. Can be one of the following: "w1", "w2", "w3", "w4".

Returns float containing the Vega magnitude

sinistra.utilities.**check_if_file**(*possible_location*)
Check if a file already exists at a given location.

Parameters **possible_location** (*str*) – File to check. Can be a path to a file, too.

Returns bool representing whether or not a file already exists there.

sinistra.utilities.**empty_data**(*datatype*)
Makes an empty data of a given datatype.

This is useful for filling tables that have missing values.

Here is what the various datatypes return:

Float: np.nan

Integer: -999999999999

String: Empty string.

Parameters **datatype** – data type, obtained by using *.dtype* on some numpy object.

sinistra.utilities.**flux_conv**(*flux_zeropont*, *counts_zeropoint*)
Calculate the conversion factor from counts to flux.

This is done by using the zeropoints of the magnitudes in flux and counts. The magnitude should be the same

no matter how we calculate it, which is how we can derive this:

$$\text{mag}_{\text{flux}} = \text{mag}_{\text{counts}}$$
$$-2.5\log_{10}(F) + Z_F = -2.5\log_{10}(C) + Z_C$$

where $F =$ Flux, $C =$ counts, and Z_F and Z_C represent the respective zeropoints when using flux or counts

$$-2.5(\log_{10}(F) - \log_{10}(C)) = Z_C - Z_F$$
$$\log_{10}\left(\frac{F}{C}\right) = \frac{Z_F - Z_C}{2.5}$$
$$\frac{F}{C} = 10^{\frac{Z_F - Z_C}{2.5}}$$
$$F = C * 10^{\frac{Z_F - Z_C}{2.5}}$$
$$F = C * \text{Flux Conversion}$$
$$\text{where Flux Conversion} = 10^{\frac{Z_F - Z_C}{2.5}}$$

This flux conversion is what is calculated here.

NOTE: Make sure the magnitudes really are the same (ie AB/Vega).

Parameters

- **flux_zeropoint** (*float*) – zero point for calculating magnitudes when using flux, such that mag = -2.5 log(flux) + *flux_zeropoint*. The actual value will depend on what flux units you want. If you want microJanskys, use 23.9, for example.
- **counts_zeropoint** – zero point for calculating magnitues when using counts, such that mag = -2.5 log(counts) + *counts_zeropoint*. This is often found in the header of the image in question.

Returns float containing the conversion from counts to flux, such that flux = counts * flux_conv. See above for derivation.

sinistra.utilities.**flux_to_mag**(*flux*, *zeropoint*)

Convert flux to magnitude with the given zeropoint.

$$m = -2.5\log_{10}(F) + C$$

Parameters

- **flux** – flux in whatever units. Choose your zeropoint correctly to make this work with the units flux is in.
- **zeropoint** – zeropoint of the system (in mags)

Returns magnitude that corresponds to the given flux

sinistra.utilities.**gaussian**(*x*, *mean*, *sigma*, *amplitude*)

The Gaussian density at the given value.

The Gaussian density is defined as

$$f(x) = Ae^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Note that if you want a normalized Gaussian (so that the total area under the curve is 1), then use the *normed_gaussian()* function.

Parameters

- **x** (*float*) – location to get the Gaussian density at.

- **mean** (*float*) – Mean of the Gaussian.
- **sigma** (*float*) – Standard deviation of the Gaussian. Should be positive
- **amplitude** – Height of the highest point of the Gaussian.

Returns Gaussain density of the given gaussian at the given x value.

sinistra.utilities.**mag_errors_to_percent_flux_errors**(*mag_error*)
Converts a magnitude error into a percent flux error.

$$m = -2.5 \log_{10}(F) + C$$
$$dm = \frac{-2.5}{\ln(10)} \frac{dF}{F}$$
$$\frac{dF}{F} = \frac{\ln(10)}{2.5} dm$$

The minus sign just tells us that increasing flux gives decreasing magnitudes, so we can safely ignore it.

Parameters **mag_error** – magnitude error

Returns percentage flux error corresponding to this magnitude error.

sinistra.utilities.**mag_to_flux**(*mag*, *zeropoint*)
Convert a magnitude into a flux.

We get the conversion by starting with the definition of the magnitude scale.

$$m = -2.5 \log_{10}(F) + C$$
$$2.5 \log_{10}(F) = C - m$$
$$F = 10^{\frac{C-m}{2.5}}$$

Parameters

- **mag** – magnitdue to be converted into a flux.
- **zeropoint** – zeropoint (in mags) of the magnitude system being used

Returns flux that corresponds to the given magnitude

sinistra.utilities.**normed_gaussian**(*x*, *mean*, *sigma*)
The Gaussian density at the given value.

The Gaussian density is defined as

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Noe that if you want a Gaussian that is not normalized (ie where you can set the amplitude), then use the *gaussian()* function.

Parameters

- **x** (*float*) – location to get the Gaussian density at.
- **mean** (*float*) – Mean of the Gaussian.
- **sigma** (*float*) – Standard deviation of the Gaussian. Should be positive

Returns Gaussain density of the given gaussian at the given x value.

sinistra.utilities.**percent_flux_errors_to_mag_errors**(*percent_flux_error*)
Converts a percentage flux error into a magnitude error.

$$m = -2.5 \log_{10}(F) + C$$
$$dm = \frac{-2.5}{\ln(10)} \frac{dF}{F}$$

Parameters **percent_flux_error** – percentage flux error

Returns magnitude error corresponding to the percentage flux error.

sinistra.utilities.**reduced_chi_sq**(*model*, *data*, *errors*)
Does a reduced chi squared calculation

$$\chi^2 = \sum_{k=1}^{n} \left(\frac{\text{model}_k - \text{data}_k}{\text{error}_k} \right)^2$$
$$\chi^2_{\text{red}} = \frac{\chi^2}{n}$$

where n is the number of data points.

Parameters

- **model** – list of values that describe a possible fit to the data
- **data** – list of values that are the data do be fitted
- **errors** – list of errors on the data

Returns value for the reduced chi squared value of the fit of the model to the data

sinistra.utilities.**vega_to_ab**(*vega_mag*, *band*)
Converts a Vega mag to AB.

This basically contains a bunch of AB-Vega conversions, so I don't have to look them up every time I want to do anything.

Parameters

- **vega_mag** (*float*) – magnitude in the Vega system to be converted to AB.
- **band** (*str*) – Band the magnitude is in. Can be one of the following: "w1", "w2", "w3", "w4".

Returns float containing the ab magnitude

CHAPTER 2

Classes

class sinistra.classes.**AsymmetricData**(*value*, *upper_error*, *lower_error*)
Bases: *sinistra.classes.Data*

Class that represents a data point, with both a value and upper and lower errors.

Addition and subtraction operators are implemented smartly.

class sinistra.classes.**Data**(*value*, *error*)
Bases: object

Class that represents a data point. Has a value, as well as errors.

Addition and subtraction operators are implemented smartly, too.

CHAPTER 3

Astropy Helpers

sinistra.astropy_helpers.**match_one**(*table_1*, *table_2*, *ra_col_1='ra'*, *ra_col_2='ra'*, *dec_col_1='dec'*, *dec_col_2='dec'*, *max_sep=3.0*, *include_all_from_1=False*)

Matches objects from two astropy tables by ra/dec. All objects from the first will be matched to one in the second.

Parameters

- **table_1** – First astopy table object containing objects with ra/dec information.
- **table_2** – First astopy table object containing objects with ra/dec information.
- **ra_col_1** – Name of the ra column in table_1. Defaults to "ra".
- **ra_col_2** – Name of the ra column in table_2. Defaults to "ra".
- **dec_col_1** – Name of the dec column in table_1. Defaults to "dec".
- **dec_col_2** – Name of the dec column in table_2. Defaults to "dec".
- **max_sep** – Maximum separation (in arcseconds) allowed for two objects to be considered a match.
- **include_all_from_1** – Whether or not to include all rows from table 1, not just those that have matches.

Returns Astropy table object containing the matches between the two input table objects. All columns from both catalogs will be included, as well as a separate separation column.

sinistra.astropy_helpers.**pretty_write**(*table*, *out_file*, *clobber=False*)

Writes an astropy table in a nice format.

Parameters

- **table** (*astropy.table.Table*) – Astropy table object to write to file.
- **out_file** (*str*) – Place to write the resulting ascii file.
- **clobber** (*bool*) – Whether or not to overwrite an existing file, if it exists. If this is false, the function will exit with an error if a file already exists here. If clobber is True, it will overwrite the file there.

sinistra.astropy_helpers.**symmetric_match**(*table_1*, *table_2*, *ra_col_1='ra'*, *ra_col_2='ra'*, *dec_col_1='dec'*, *dec_col_2='dec'*, *max_sep=3.0*)

Matches objects from two astropy tables by ra/dec.

This function does symmetric matching. This meansn that to be defined as a match, both objects must be each other's closest match. Their separation must also be less than the *max_sep* parameter.

Parameters

- **table_1** – First astopy table object containing objects with ra/dec information.
- **table_2** – First astopy table object containing objects with ra/dec information.
- **ra_col_1** – Name of the ra column in table_1. Defaults to "ra".
- **ra_col_2** – Name of the ra column in table_2. Defaults to "ra".
- **dec_col_1** – Name of the dec column in table_1. Defaults to "dec".
- **dec_col_2** – Name of the dec column in table_2. Defaults to "dec".
- **max_sep** – Maximum separation (in arcseconds) allowed for two objects to be considered a match.

Returns Astropy table object containing the matches between the two input table objects. All columns from both catalogs will be included, as well as a separate separation column.

sinistra.astropy_helpers.**symmetric_match_both**(*table_1*, *table_2*, *ra_col_1='ra'*, *ra_col_2='ra'*, *dec_col_1='dec'*, *dec_col_2='dec'*, *max_sep=3.0*)

Matches objects from two astropy tables by ra/dec, including all objects.

This function does symmetric matching. This meansn that to be defined as a match, both objects must be each other's closest match. Their separation must also be less than the *max_sep* parameter.

Each object from both tables is included, even if there are no matches for that object. The empty space will be filled with the appropriate empty data.

Parameters

- **table_1** – First astopy table object containing objects with ra/dec information.
- **table_2** – First astopy table object containing objects with ra/dec information.
- **ra_col_1** – Name of the ra column in table_1. Defaults to "ra".
- **ra_col_2** – Name of the ra column in table_2. Defaults to "ra".
- **dec_col_1** – Name of the dec column in table_1. Defaults to "dec".
- **dec_col_2** – Name of the dec column in table_2. Defaults to "dec".
- **max_sep** – Maximum separation (in arcseconds) allowed for two objects to be considered a match.

Returns Astropy table object containing the matches between the two input table objects. All columns from both catalogs will be included, as well as a separate separation column.

CHAPTER 4

Photometry

sinistra.phot.**aperture_grid**(*image*, *spacing*, *output=False*, *clobber=False*)

Makes a grid of apertures and writes to a qphot-compatible coords file.

Parameters

- **image** (*str*) – location of the image to be used.
- **spacing** – x and y spacing between locations. For example, if one aperture is located at x, the next is x + spacing. The same thing holds for the y direction.
- **spacing** – float
- **output** (*bool / str*) – Either False, or the name of the coordinates file that the output will be written to. If you don't want to write to a file, pass in False.
- **clobber** – Whether or not to check for an already existing file of this name. If it already exists, it will raise an error.

Returns List of tuples that contain the x,y coordinates of each aperture.

sinistra.phot.**fit_gaussian_negative**(*data*, *upper_cutoff*, *plot=False*, *savename=None*, *data_label='Counts'*)

Fits a Gaussain to all data underneath some cutoff.

This is useful when doing things where the lower side of a Gaussian tells about the intrinsic scatter of something, whereas the upper end is contaminated by real objects. An example is the flux within many randomly placed apertures. The lower side will tell the sky error in the aperture flux.

Parameters

- **data** (*list, np.array*) – list of data to be fitted.
- **upper_cutoff** (*float*) – Data below this number will be used to fit the Gaussian, while data above this will be rejected.
- **plot** (*bool*) – Whether or not to create a plot showing the best fit to the histogram of the data.
- **savename** (*str*) – If you are creating a plot, enter a file path here to save the figure. If this is left as none, then the plot will not be saved. If you are running in an Jupyter notebook, the figure will show up even if it is not saved, so it's not essential.
- **data_label** (*str*) – Descriptor of the data that will be used on the x-axis of the data histogram. Defaults to "Flux" unless otherwise specified.

Returns the mean and sigma of the best fit Gaussian.

sinistra.phot.**sky_error**(*image*, *aperture_size*, *flux_conv*, *plot=True*)

Figures out the sky error with an aperture of a given size.

This is done by placing apertures in a grid throughout the image, then doing aperture photometry within those apertures. Most of those will be on empty sky. By examining the spread of the fluxes of the apertures that were on empty sky, we can get an idea of the general sky scatter in an aperture of that size.

Parameters

- **image** (*str*) – Image to do this process on. Pass in the path to the image, or just the name of the image if it is in the current directory.
- **aperture_size** (*float*) – The diameter of the aperture you want to use. NOTE THAT THIS IS DIAMETER, NOT RADIUS!
- **flux_conv** (*float*) – multiplicative factor to get from counts in the image to real fluxes in whatever units you want. Should be defined such that "flux" = "counts" * *flux_conv*. This can be determined from the zeropoint of the image.
- **plot** (*bool*) – Whether or not to plot the Gaussian that results from this process.

Returns flux error within that aperture size, in real flux units. Note you need to multiply by the aperture correction to get total errors!

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

S