AegeanTools Documentation

Release 2.2.4

PaulHancock

Oct 08, 2021

CONTENTS:

CHAPTER

ONE

AEGEANTOOLS MODULES

1.1 angle_tools

Tools for manipulating angles on the surface of a sphere - distance - bearing between two points - translation along a path - paths are either great circles or rhumb lines

also angle <-> string conversion tools for Aegean

AegeanTools.angle_tools.**bear**(*ra1*, *dec1*, *ra2*, *dec2*)

Calculate the bearing of point 2 from point 1 along a great circle. The bearing is East of North and is in [0, 360), whereas position angle is also East of North but (-180,180]

Parameters

ra1, dec1, ra2, dec2 [float] The sky coordinates (degrees) of the two points.

Returns

bear [float] The bearing of point 2 from point 1 (degrees).

AegeanTools.angle_tools.**bear_rhumb**(*ra1*, *dec1*, *ra2*, *dec2*)

Calculate the bearing of point 2 from point 1 along a Rhumb line. The bearing is East of North and is in [0, 360), whereas position angle is also East of North but (-180,180]

Parameters

ra1, dec1, ra2, dec2 [float] The sky coordinates (degrees) of the two points.

Returns

dist [float] The bearing of point 2 from point 1 along a Rhumb line (degrees).

AegeanTools.angle_tools.**dec2dec**(*dec*)

Convert sexegessimal RA string into a float in degrees.

Parameters

dec [str] A string separated representing the Dec. Expected format is *[+-]hh:mm[:ss.s]* Colons can be replaced with any whit space character.

Returns

dec [float] The Dec in degrees.

AegeanTools.angle_tools.**dec2dms**(*x*)

Convert decimal degrees into a sexagessimal string in degrees.

Parameters

x [float] Angle in degrees

Returns

dms [str] String of format [+-]DD:MM:SS.SS or XX:XX:XX.XX if x is not finite.

AegeanTools.angle_tools.**dec2hms**(*x*)
Convert decimal degrees into a sexagessimal string in hours.

Parameters

x [float] Angle in degrees

Returns

dms [string] String of format HH:MM:SS.SS or XX:XX:XX.XX if x is not finite.

AegeanTools.angle_tools.**dist_rhumb**(*ra1*, *dec1*, *ra2*, *dec2*)
Calculate the Rhumb line distance between two points [1]. A Rhumb line between two points is one which follows a constant bearing.

Parameters

ra1, dec1, ra2, dec2 [float] The position of the two points (degrees).

Returns

dist [float] The distance between the two points along a line of constant bearing.

Notes

AegeanTools.angle_tools.**gcd**(*ra1*, *dec1*, *ra2*, *dec2*)
Calculate the great circle distance between to points using the haversine formula [1].

Parameters

ra1, dec1, ra2, dec2 [float] The coordinates of the two points of interest. Units are in degrees.

Returns

dist [float] The distance between the two points in degrees.

Notes

This duplicates the functionality of astropy but is faster as there is no creation of SkyCoords objects.

AegeanTools.angle_tools.**ra2dec**(*ra*)
Convert sexegessimal RA string into a float in degrees.

Parameters

ra [str] A string separated representing the RA. Expected format is *hh:mm[:ss.s]* Colons can be replaced with any whit space character.

Returns

ra [float] The RA in degrees.

AegeanTools.angle_tools.**translate**(*ra*, *dec*, *r*, *theta*)
Translate a given point a distance r in the (initial) direction theta, along a great circle.

Parameters

ra, dec [float] The initial point of interest (degrees).

r, theta [float] The distance and initial direction to translate (degrees).

Returns

ra, dec [(float, float)] The translated position (degrees).

AegeanTools.angle_tools.**translate_rhumb**(*ra*, *dec*, *r*, *theta*)
Translate a given point a distance r in the (initial) direction theta, along a Rhumb line.

Parameters

ra, dec [float] The initial point of interest (degrees).

r, theta [float] The distance and initial direction to translate (degrees).

Returns

ra, dec [float] The translated position (degrees).

1.2 BANE

This module contains all of the BANE specific code The function filter_image should be imported from elsewhere and run as is.

AegeanTools.BANE.**barrier**(*events*, *sid*, *kind='neighbour'*)
act as a multiprocessing barrier

AegeanTools.BANE.**filter_image**(*im_name*, *out_base*, *step_size=None*, *box_size=None*, *twopass=False*, *cores=None*, *mask=True*, *compressed=False*, *nslice=None*)
Create a background and noise image from an input image. Resulting images are written to *outbase_bkg.fits* and *outbase_rms.fits*

Parameters

im_name [str] Image to filter.

out_base [str or None] The output filename base. Will be modified to make _bkg and _rms files. If None, then no files are written.

step_size [(int,int)] Tuple of the x,y step size in pixels

box_size [(int,int)] The size of the box in pixels

twopass [bool] Perform a second pass calculation to ensure that the noise is not contaminated by the background. Default = False

cores [int] Number of CPU corse to use. Default = all available

nslice [int] The image will be divided into this many horizontal stripes for processing. Default = None = equal to cores

mask [bool] Mask the output array to contain np.nna wherever the input array is nan or not finite. Default = true

compressed [bool] Return a compressed version of the background/noise images. Default = False

Returns

bkg, rms [*numpy.ndarray*] The computed background and rms maps (not compressed)

AegeanTools.BANE.**filter_mc_sharemem**(*filename*, *step_size*, *box_size*, *cores*, *shape*, *nslice=None*, *domask=True*)
Calculate the background and noise images corresponding to the input file. The calculation is done via a box-car approach and uses multiple cores and shared memory.

Parameters

filename [str] Filename to be filtered.

step_size [(int, int)] Step size for the filter.

box_size [(int, int)] Box size for the filter.

cores [int] Number of cores to use. If None then use all available.

nslice [int] The image will be divided into this many horizontal stripes for processing. Default = None = equal to cores

shape [(int, int)] The shape of the image in the given file.

domask [bool] True(Default) = copy data mask to output.

Returns

bkg, rms [numpy.ndarray] The interpolated background and noise images.

AegeanTools.BANE.**get_step_size**(*header*)

Determine the grid spacing for BANE operation.

This is set to being 4x the synthesized beam width. If the beam is not circular then the "width" is sqrt(a*b)

For the standard 4 pix/beam, the step size will be 16 pixels.

Parameters

header

Returns

step_size [(int, int)] The grid spacing for BANE operation

AegeanTools.BANE.**sigma_filter**(*filename*, *region*, *step_size*, *box_size*, *shape*, *domask*, *sid*)

Calculate the background and rms for a sub region of an image. The results are written to shared memory - irms and ibkg.

Parameters

filename [string] Fits file to open

region [list] Region within the fits file that is to be processed. (row_min, row_max).

step_size [(int, int)] The filtering step size

box_size [(int, int)] The size of the box over which the filter is applied (each step).

shape [tuple] The shape of the fits image

domask [bool] If true then copy the data mask to the output.

sid [int] The stripe number

Returns

None

AegeanTools.BANE.**sigmaclip**(*arr*, *lo*, *hi*, *reps=3*)

Perform sigma clipping on an array, ignoring non finite values.

During each iteration return an array whose elements c obey: mean -std*lo < c < mean + std*hi

where mean/std are the mean std of the input array.

Parameters

arr [iterable] An iterable array of numeric types.

lo [float] The negative clipping level.

hi [float] The positive clipping level.

reps [int] The number of iterations to perform. Default = 3.

Returns

mean [float] The mean of the array, possibly nan

std [float] The std of the array, possibly nan

Notes

Scipy v0.16 now contains a comparable method that will ignore nan/inf values.

AegeanTools.BANE.**write_fits**(*data*, *header*, *file_name*)
Combine data and a fits header to write a fits file.

Parameters

data [numpy.ndarray] The data to be written.

header [astropy.io.fits.hduheader] The header for the fits file.

file_name [string] The file to write

Returns

None

1.3 catalogs

Module for reading at writing catalogs

AegeanTools.catalogs.**check_table_formats**(*files*)
Determine whether a list of files are of a recognizable output type.

Parameters

files [str] A list of file names

Returns

result [bool] True if *all* the file names are supported

AegeanTools.catalogs.**get_table_formats**()
Create a list of file extensions that are supported for writing.

Returns

fmts [list] A list of file name extensions that are supported.

AegeanTools.catalogs.**load_catalog**(*filename*)
Load a catalogue and extract the source positions (only)

Parameters

filename [str] Filename to read. Supported types are csv, tab, tex, vo, vot, and xml.

Returns

catalogue [list] A list of [(ra, dec), . . .]

AegeanTools.catalogs.**load_table**(*filename*)
Load a table from a given file.

Supports csv, tab, tex, vo, vot, xml, fits, and hdf5.

Parameters

filename [str] File to read

Returns

table [Table] Table of data.

AegeanTools.catalogs.**nulls**(*x*)
Convert values of -1 into None.

Parameters

x [float or int] Value to convert

Returns

val [[x, None]]

AegeanTools.catalogs.**save_catalog**(*filename*, *catalog*, *meta=None*, *prefix=None*)
Save a catalogue of sources using filename as a model. Meta data can be written to some file types (fits, votable).

Each type of source will be in a separate file:

- base_comp.ext *AegeanTools.models.ComponentSource*
- base_isle.ext *AegeanTools.models.IslandSource*
- base_simp.ext *AegeanTools.models.SimpleSource*

Where filename = base.ext

Parameters

filename [str] Name of file to write, format is determined by extension.

catalog [list] A list of sources to write. Sources must be of type *AegeanTools.models.ComponentSource*, *AegeanTools.models.SimpleSource*, or *AegeanTools.models.IslandSource*.

prefix [str] Prepend each column name with "prefix_". Default is to prepend nothing.

meta [dict] Meta data to be written to the output file. Support for metadata depends on file type.

Returns

None

AegeanTools.catalogs.**show_formats**()
Print a list of all the file formats that are supported for writing. The file formats are determined by their extensions.

Returns

None

AegeanTools.catalogs.**table_to_source_list**(*table*, *src_type=<class 'AegeanTools.models.ComponentSource'>*)
Convert a table of data into a list of sources.

A single table must have consistent source types given by src_type. src_type should be one of *AegeanTools.models.ComponentSource*, *AegeanTools.models.SimpleSource*, or *AegeanTools.models.IslandSource*.

Parameters

table [Table] Table of sources

src_type [class] Sources must be of type *AegeanTools.models.ComponentSource*, *AegeanTools.models.SimpleSource*, or *AegeanTools.models.IslandSource*.

Returns

sources [list] A list of objects of the given type.

AegeanTools.catalogs.**update_meta_data**(*meta=None*)

Modify the metadata dictionary. DATE, PROGRAM, and PROGVER are added/modified.

Parameters

meta [dict] The dictionary to be modified, default = None (empty)

Returns

An updated dictionary.

AegeanTools.catalogs.**writeAnn**(*filename*, *catalog*, *fmt*)

Write an annotation file that can be read by Kvis (.ann) or DS9 (.reg). Uses ra/dec from catalog. Draws ellipses if bmaj/bmin/pa are in catalog. Draws 30" circles otherwise.

Only *AegeanTools.models.ComponentSource* will appear in the annotation file unless there are none, in which case *AegeanTools.models.SimpleSource* (if present) will be written. If any *AegeanTools.models.IslandSource* objects are present then an island contours file will be written.

Parameters

filename [str] Output filename base.

catalog [list] List of sources.

fmt [['ann', 'reg']] Output file type.

Returns

None

See also:

AegeanTools.catalogs.writeIslandContours

AegeanTools.catalogs.**writeDB**(*filename*, *catalog*, *meta=None*)

Output an sqlite3 database containing one table for each source type

Parameters

filename [str] Output filename

catalog [list] List of sources of type *AegeanTools.models.ComponentSource*, *AegeanTools.models.SimpleSource*, or *AegeanTools.models.IslandSource*.

meta [dict] Meta data to be written to table *meta*

Returns

None

AegeanTools.catalogs.**writeFITSTable**(*filename*, *table*)

Convert a table into a FITSTable and then write to disk.

Parameters

filename [str] Filename to write.

table [Table] Table to write.

Returns

None

Notes

Due to a bug in numpy, *int32* and *float32* are converted to *int64* and *float64* before writing.

AegeanTools.catalogs.**writeIslandBoxes**(*filename*, *catalog*, *fmt*)
Write an output file in ds9 .reg, or kvis .ann format that contains bounding boxes for all the islands.

Parameters

filename [str] Filename to write.

catalog [list] List of sources. Only those of type *AegeanTools.models.IslandSource* will have contours drawn.

fmt [str] Output format type. Currently only 'reg' and 'ann' are supported. Default = 'reg'.

Returns

None

See also:

AegeanTools.catalogs.writeIslandContours()

AegeanTools.catalogs.**writeIslandContours**(*filename*, *catalog*, *fmt='reg'*)
Write an output file in ds9 .reg format that outlines the boundaries of each island.

Parameters

filename [str] Filename to write.

catalog [list] List of sources. Only those of type *AegeanTools.models.IslandSource* will have contours drawn.

fmt [str] Output format type. Currently only 'reg' is supported (default)

Returns

None

See also:

AegeanTools.catalogs.writeIslandBoxes()

AegeanTools.catalogs.**write_catalog**(*filename*, *catalog*, *fmt=None*, *meta=None*, *prefix=None*)
Write a catalog (list of sources) to a file with format determined by extension.

Sources must be of type *AegeanTools.models.ComponentSource*, *AegeanTools.models.SimpleSource*, or *AegeanTools.models.IslandSource*.

Parameters

filename [str] Base name for file to write. *_simp*, *_comp*, or *_isle* will be added to differentiate the different types of sources that are being written.

catalog [list] A list of source objects. Sources must be of type *AegeanTools.models.ComponentSource*, *AegeanTools.models.SimpleSource*, or *AegeanTools.models.IslandSource*.

fmt [str] The file format extension.

prefix [str] Prepend each column name with "prefix_". Default is to prepend nothing.

meta [dict] A dictionary to be used as metadata for some file types (fits, VOTable).

Returns

None

AegeanTools.catalogs.**write_table**(*table*, *filename*)

Write a table to a file.

Parameters

table [Table] Table to be written

filename [str] Destination for saving table.

Returns

None

1.4 cluster

Cluster and crossmatch tools and analysis functions.

Includes: - DBSCAN clustering

AegeanTools.cluster.**check_attributes_for_regroup**(*catalog*)

Check that the catalog has all the attributes reqired for the regrouping task.

Parameters

catalog [list] List of python objects, ideally derived from *AegeanTools.models.SimpleSource*

Returns

result [bool] True if the first entry in the catalog has the required attributes

AegeanTools.cluster.**norm_dist**(*src1*, *src2*)

Calculate the normalised distance between two sources. Sources are elliptical Gaussians.

The normalised distance is calculated as the GCD distance between the centers, divided by quadrature sum of the radius of each ellipse along a line joining the two ellipses.

For ellipses that touch at a single point, the normalized distance will be 1/sqrt(2).

Parameters

src1, src2 [object] The two positions to compare. Objects must have the following parameters: (ra, dec, a, b, pa).

Returns

dist: float The normalised distance.

AegeanTools.cluster.**pairwise_ellpitical_binary**(*sources*, *eps*, *far=None*)

Do a pairwise comparison of all sources and determine if they have a normalized distance within eps.

Form this into a matrix of shape NxN.

Parameters

sources [list] A list of sources (objects with parameters: ra,dec,a,b,pa)

eps [float] Normalised distance constraint.

far [float] If sources have a dec that differs by more than this amount then they are considered to be not matched. This is a short-cut around performing GCD calculations.

Returns

prob [numpy.ndarray] A 2d array of True/False.

See also:

AegeanTools.cluster.norm_dist()

AegeanTools.cluster.**regroup**(*catalog*, *eps*, *far=None*, *dist=<function norm_dist>*)

Regroup the islands of a catalog according to their normalised distance. Return a list of island groups. Sources have their (island,source) parameters relabeled.

Parameters

catalog [str or object] Either a filename to read into a source list, or a list of objects with the following properties[units]: ra[deg], dec[deg], a[arcsec], b[arcsec],pa[deg], peak_flux[any]

eps [float] maximum normalised distance within which sources are considered to be grouped

far [float] (degrees) sources that are further than this distance appart will not be grouped, and will not be tested. Default = None.

dist [func] a function that calculates the distance between two sources must accept two SimpleSource objects. Default = *AegeanTools.cluster.norm_dist()*

Returns

islands [list] A list of islands. Each island is a list of sources.

See also:

AegeanTools.cluster.norm_dist()

AegeanTools.cluster.**regroup_dbscan**(*srccat*, *eps=4*)

Regroup the islands of a catalog according using DBSCAN.

Return a list of island groups.

Parameters

srccat [[*object*]] A list of objects with parameters ra,dec (both in decimal degrees)

eps [float] maximum normalized distance within which sources are considered to be grouped

Returns

islands [list of lists] Each island contains integer indices for members from srccat (in descending dec order).

AegeanTools.cluster.**regroup_vectorized**(*srccat*, *eps*, *far=None*, *dist=<function norm_dist>*)

Regroup the islands of a catalog according to their normalised distance.

Assumes srccat is recarray-like for efficiency. Return a list of island groups.

Parameters

srccat [np.rec.arry or pd.DataFrame] Should have the following fields[units]: ra[deg],dec[deg], a[arcsec],b[arcsec],pa[deg], peak_flux[any]

eps [float] maximum normalised distance within which sources are considered to be grouped

far [float] (degrees) sources that are further than this distance apart will not be grouped, and will not be tested. Default = 0.5.

dist [func] a function that calculates the distance between a source and each element of an array of sources. Default = *AegeanTools.cluster.norm_dist()*

Returns

islands [list of lists] Each island contians integer indices for members from srccat (in descending dec order).

AegeanTools.cluster.**resize**(*catalog*, *ratio=None*, *psfhelper=None*)

Resize all the sources in a given catalogue. Either use a ratio to blindly scale all sources by the same amount, or use a psf map to deconvolve the sources and then convolve them with the new psf

Sources that cannot be rescaled are not returned

Parameters

catalog [list] List of objects

ratio [float, default=None] Ratio for scaling the sources

psfhelper [*AegeanTools.wcs_helpers.WCSHelper*, default=None] A wcs helper object that contains psf information for the target image/projection

Returns

catalog [list] Modified list of objects

AegeanTools.cluster.**sky_dist**(*src1*, *src2*)

Great circle distance between two sources. A check is made to determine if the two sources are the same object, in this case the distance is zero.

Parameters

src1, src2 [object] Two sources to check. Objects must have parameters (ra,dec) in degrees.

Returns

distance [float] The distance between the two sources.

See also:

AegeanTools.angle_tools.gcd()

1.5 fits_image

Tools for interacting with fits images (HUDLists)

class AegeanTools.fits_image.**FitsImage**(*filename=None*, *hdu_index=0*, *beam=None*, *cube_index=None*)
An object that handles the loading and manipulation of a fits file.

get_background_rms()
Calculate the rms of the image. The rms is calculated from the interqurtile range (IQR), to reduce bias from source pixels.

Returns

rms [float] The image rms.

Notes

The rms value is cached after first calculation.

get_hdu_header()
Get the image header.

get_pixels()
Get the image data.

Returns

pixels [numpy.ndarray] 2d Array of image pixels.

pix2sky(*pixel*)
Get the sky coordinates for a given image pixel.

Parameters

pixel [(float, float)] Image coordinates.

Returns

ra,dec [float] Sky coordinates (degrees)

set_pixels(*pixels*)
Set the image data. Will not work if the new image has a different shape than the current image.

Parameters

pixels [numpy.ndarray] New image data

Returns

None

sky2pix(*skypos*)
Get the pixel coordinates for a given sky position (degrees).

Parameters

skypos [(float,float)] ra,dec position in degrees.

Returns

x,y [float] Pixel coordinates.

1.6 fits_interp

A module to allow fits files to be shrunk in size using decimation, and to be grown in size using interpolation.

AegeanTools.fits_interp.**compress**(*datafile*, *factor*, *outfile=None*)

Compress a file using decimation.

Parameters

datafile [str or HDUList] Input data to be loaded. (HDUList will be modified if passed).

factor [int] Decimation factor.

outfile [str] File to be written. Default = None, which means don't write a file.

Returns

hdulist [HDUList] A decimated HDUList

See also:

AegeanTools.fits_interp.expand()

AegeanTools.fits_interp.**expand**(*datafile*, *outfile=None*)

Expand and interpolate the given data file using the given method. Datafile can be a filename or an HDUList

It is assumed that the file has been compressed and that there are *BN_?* keywords in the fits header that describe how the compression was done.

Parameters

datafile [str or HDUList] filename or HDUList of file to work on

outfile [str] filename to write to (default = None)

Returns

hdulist [HDUList] HDUList of the expanded data.

See also:

AegeanTools.fits_interp.compress()

AegeanTools.fits_interp.**load_file_or_hdu**(*filename*)

Load a file from disk and return an HDUList If filename is already an HDUList return that instead

Parameters

filename [str or HDUList] File or HDU to be loaded

Returns

hdulist [HDUList]

1.7 fitting

Provide fitting routines and helper fucntions to Aegean

AegeanTools.fitting.**Bmatrix**(*C*)

Calculate a matrix which is effectively the square root of the correlation matrix C

Parameters

C [2d array] A covariance matrix

Returns

B [2d array] A matrix B such the B.dot(B') = inv(C)

AegeanTools.fitting.**Cmatrix**(*x*, *y*, *sx*, *sy*, *theta*)

Construct a correlation matrix corresponding to the data. The matrix assumes a gaussian correlation function.

Parameters

x, y [array-like] locations at which to evaluate the correlation matirx

sx, sy [float] major/minor axes of the gaussian correlation function (sigmas)

theta [float] position angle of the gaussian correlation function (degrees)

Returns

data [array-like] The C-matrix.

AegeanTools.fitting.**RB_bias**(*data*, *pars*, *ita=None*, *acf=None*)

Calculate the expected bias on each of the parameters in the model pars. Only parameters that are allowed to vary will have a bias. Calculation follows the description of Refrieger & Brown 1998 (cite).

Parameters

data [2d-array] data that was fit

pars [lmfit.Parameters] The model

ita [2d-array] The ita matrix (optional).

acf [2d-array] The acf for the data.

Returns

bias [array] The bias on each of the parameters

AegeanTools.fitting.**bias_correct**(*params*, *data*, *acf=None*)

Calculate and apply a bias correction to the given fit parameters

Parameters

params [lmfit.Parameters] The model parameters. These will be modified.

data [2d-array] The data which was used in the fitting

acf [2d-array] ACF of the data. Default = None.

Returns

None

See also:

AegeanTools.fitting.RB_bias()

AegeanTools.fitting.**condon_errors**(*source*, *theta_n*, *psf=None*)

Calculate the parameter errors for a fitted source using the description of Condon'97 All parameters are assigned errors, assuming that all params were fit. If some params were held fixed then these errors are overestimated.

Parameters

source [*AegeanTools.models.SimpleSource*] The source which was fit.

theta_n [float or None] A measure of the beam sampling. (See Condon'97).

psf [*AegeanTools.wcs_helpers.Beam*] The psf at the location of the source.

Returns

None

AegeanTools.fitting.**covar_errors**(*params*, *data*, *errs*, *B*, *C=None*)

Take a set of parameters that were fit with lmfit, and replace the errors with the 1sigma errors calculated using the covariance matrix.

Parameters

params [lmfit.Parameters] Model

data [2d-array] Image data

errs [2d-array ?] Image noise.

B [2d-array] B matrix.

C [2d-array] C matrix. Optional. If supplied then Bmatrix will not be used.

Returns

params [lmfit.Parameters] Modified model.

AegeanTools.fitting.**do_lmfit**(*data*, *params*, *B=None*, *errs=None*, *dojac=True*)

Fit the model to the data data may contain 'flagged' or 'masked' data with the value of np.NaN

Parameters

data [2d-array] Image data

params [lmfit.Parameters] Initial model guess.

B [2d-array] B matrix to be used in residual calculations. Default = None.

errs [1d-array]

dojac [bool] If true then an analytic jacobian will be passed to the fitting routine.

Returns

result [?] lmfit.minimize result.

params [lmfit.Params] Fitted model.

See also:

AegeanTools.fitting.lmfit_jacobian()

AegeanTools.fitting.**elliptical_gaussian**(*x*, *y*, *amp*, *xo*, *yo*, *sx*, *sy*, *theta*)

Generate a model 2d Gaussian with the given parameters. Evaluate this model at the given locations x,y.

Parameters

x, y [numeric or array-like] locations at which to evaluate the gaussian

amp [float] Peak value.

xo, yo [float] Center of the gaussian.

sx, sy [float] major/minor axes in sigmas

theta [float] position angle (degrees) CCW from x-axis

Returns

data [numeric or array-like] Gaussian function evaluated at the x,y locations.

AegeanTools.fitting.**elliptical_gaussian_with_alpha**(*x*, *y*, *v*, *amp*, *xo*, *yo*, *vo*, *sx*, *sy*, *theta*, *alpha*, *beta=None*)

Generate a model 2d Gaussian with spectral terms. Evaluate this model at the given locations x,y,dv.

amp is the amplitude at the reference frequency vo

The model is: S(x,v) = amp (v/vo) ^ (alpha + beta x log(v/vo))

When beta is none it is ignored.

Parameters

x, y, v [numeric or array-like] locations at which to evaluate the gaussian

amp [float] Peak value.

xo, yo, vo: float Center of the gaussian.

sx, sy [float] major/minor axes in sigmas

theta [float] position angle (degrees) CCW from x-axis

alpha, beta: float The spectral terms of the fit.

Returns

data [numeric or array-like] Gaussian function evaluated at the x,y locations.

AegeanTools.fitting.**emp_hessian**(*pars*, *x*, *y*)

Calculate the hessian matrix empirically. Create a hessian matrix corresponding to the source model 'pars' Only parameters that vary will contribute to the hessian. Thus there will be a total of nvar x nvar entries, each of which is a len(x) x len(y) array.

Parameters

pars [lmfit.Parameters] The model

x, y [list] locations at which to evaluate the Hessian

Returns

h [np.array] Hessian. Shape will be (nvar, nvar, len(x), len(y))

See also:

AegeanTools.fitting.hessian()

Notes

Uses *AegeanTools.fitting.emp_jacobian()* to calculate the first order derivatives.

AegeanTools.fitting.**emp_jacobian**(*pars*, *x*, *y*)

An empirical calculation of the Jacobian Will work for a model that contains multiple Gaussians, and for which some components are not being fit (don't vary).

Parameters

pars [lmfit.Model] The model parameters

x, y [list] Locations at which the jacobian is being evaluated

Returns

j [2d array] The Jacobian.

See also:

AegeanTools.fitting.jacobian()

AegeanTools.fitting.**errors**(*source*, *model*, *wcshelper*)

Convert pixel based errors into sky coord errors

Parameters

source [*AegeanTools.models.SimpleSource*] The source which was fit.

model [lmfit.Parameters] The model which was fit.

wcshelper [*AegeanTools.wcs_helpers.WCSHelper*] WCS information.

Returns

source [*AegeanTools.models.SimpleSource*] The modified source obejct.

AegeanTools.fitting.**hessian**(*pars*, *x*, *y*)

Create a hessian matrix corresponding to the source model 'pars' Only parameters that vary will contribute to the hessian. Thus there will be a total of nvar x nvar entries, each of which is a len(x) x len(y) array.

Parameters

pars [lmfit.Parameters] The model

x, y [list] locations at which to evaluate the Hessian

Returns

h [np.array] Hessian. Shape will be (nvar, nvar, len(x), len(y))

See also:

AegeanTools.fitting.emp_hessian()

AegeanTools.fitting.**jacobian**(*pars*, *x*, *y*)

Analytical calculation of the Jacobian for an elliptical gaussian Will work for a model that contains multiple Gaussians, and for which some components are not being fit (don't vary).

Parameters

pars [lmfit.Model] The model parameters

x, y [list] Locations at which the jacobian is being evaluated

Returns

j [2d array] The Jacobian.

See also:

AegeanTools.fitting.emp_jacobian()

AegeanTools.fitting.**lmfit_jacobian**(*pars*, *x*, *y*, *errs=None*, *B=None*, *emp=False*)

Wrapper around *AegeanTools.fitting.jacobian()* and *AegeanTools.fitting.emp_jacobian()* which gives the output in a format that is required for lmfit.

Parameters

pars [lmfit.Model] The model parameters

x, y [list] Locations at which the jacobian is being evaluated

errs [list] a vector of 1sigma errors (optional). Default = None

B [2d-array] a B-matrix (optional) see *AegeanTools.fitting.Bmatrix()*

emp [bool] If true the use the empirical Jacobian, otherwise use the analytical one. Default = False.

Returns

j [2d-array] A Jacobian.

See also:

AegeanTools.fitting.Bmatrix()

AegeanTools.fitting.jacobian()

AegeanTools.fitting.emp_jacobian()

AegeanTools.fitting.**make_ita**(*noise*, *acf=None*)

Create the matrix ita of the noise where the noise may be a masked array where ita(x,y) is the correlation between pixel pairs that have the same separation as x and y.

Parameters

noise [2d-array] The noise image

acf [2d-array] The autocorrelation matrix. (None = calculate from data). Default = None.

Returns

ita [2d-array] The matrix ita

AegeanTools.fitting.**nan_acf**(*noise*)

Calculate the autocorrelation function of the noise where the noise is a 2d array that may contain nans

Parameters

noise [2d-array] Noise image.

Returns

acf [2d-array] The ACF.

AegeanTools.fitting.**new_errors**(*source*, *model*, *wcshelper*)

Convert pixel based errors into sky coord errors Uses covariance matrix for ra/dec errors and calculus approach to a/b/pa errors

Parameters

source [*AegeanTools.models.SimpleSource*] The source which was fit.

model [lmfit.Parameters] The model which was fit.

wcshelper [*AegeanTools.wcs_helpers.WCSHelper*] WCS information.

Returns

source [*AegeanTools.models.SimpleSource*] The modified source obejct.

`AegeanTools.fitting.`**ntwodgaussian_lmfit**(*params*)

Convert an lmfit.Parameters object into a function which calculates the model.

Parameters

params [lmfit.Parameters] Model parameters, can have multiple components.

Returns

model [func] A function f(x,y) that will compute the model.

1.8 flags

Flag constants for use by Aegean.

1.9 MIMAS

MIMAS - The Multi-resolution Image Mask for Aegean Software

TODO: Write an in/out reader for MOC formats described by http://arxiv.org/abs/1505.02937

class `AegeanTools.MIMAS.`**Dummy**(*maxdepth=8*)

A state storage class for MIMAS to work with.

Attributes

add_region [list] List of `AegeanTools.MIMAS.Region` to be added.

rem_region [list] List of `AegeanTools.MIMAS.Region` to be subtracted.

include_circles [[[ra, dec, radius],...]] List of circles to be added to the region, units are degrees.

exclude_circles [[[ra, dec, radius], ...]] List of circles to be subtracted from the region, units are degrees.

include_polygons [[[ra,dec, ...], ...]] List of polygons to be added to the region, units are degrees.

exclude_polygons [[[ra,dec, ...], ...]] List of polygons to be subtracted from the region, units are degrees.

maxdepth [int] Depth or resolution of the region for HEALPix. There are 4*2**maxdepth pixels at the deepest layer. Default = 8.

galactic: bool If true then all ra/dec coordinates will be interpreted as if they were in galactic lat/lon (degrees)

`AegeanTools.MIMAS.`**box2poly**(*line*)

Convert a string that describes a box in ds9 format, into a polygon that is given by the corners of the box

Parameters

line [str] A string containing a DS9 region command for a box.

Returns

poly [[ra, dec, ...]] The corners of the box in clockwise order from top left.

AegeanTools.MIMAS.**circle2circle**(*line*)
Parse a string that describes a circle in ds9 format.

Parameters

line [str] A string containing a DS9 region command for a circle.

Returns

circle [[ra, dec, radius]] The center and radius of the circle.

AegeanTools.MIMAS.**combine_regions**(*container*)
Return a region that is the combination of those specified in the container. The container is typically a results instance that comes from argparse.

Order of construction is: add regions, subtract regions, add circles, subtract circles, add polygons, subtract polygons.

Parameters

container [*AegeanTools.MIMAS.Dummy*] The regions to be combined.

Returns

region [*AegeanTools.regions.Region*] The constructed region.

AegeanTools.MIMAS.**galactic2fk5**(*l*, *b*)
Convert galactic l/b to fk5 ra/dec

Parameters

l, b [float] Galactic coordinates in radians.

Returns

ra, dec [float] FK5 ecliptic coordinates in radians.

AegeanTools.MIMAS.**intersect_regions**(*flist*)
Construct a region which is the intersection of all regions described in the given list of file names.

Parameters

flist [list] A list of region filenames.

Returns

region [*AegeanTools.regions.Region*] The intersection of all regions, possibly empty.

AegeanTools.MIMAS.**mask2mim**(*maskfile*, *mimfile*, *threshold=1.0*, *maxdepth=8*)
Use a fits file as a mask to create a region file.

Pixels in mask file that are equal or above the threshold will be included in the reigon, while those that are below the threshold will not.

Parameters

maskfile [str] Input file in fits format.

mimfile [str] Output filename

threshold [float] threshold value for separating include/exclude values

maxdepth [int] Maximum depth (resolution) of the healpix pixels

AegeanTools.MIMAS.**mask_catalog**(*regionfile*, *infile*, *outfile*, *negate=False*, *racol='ra'*, *deccol='dec'*)

Apply a region file as a mask to a catalog, removing all the rows with ra/dec inside the region If negate=False then remove the rows with ra/dec outside the region.

Parameters

regionfile [str] A file which can be loaded as a *AegeanTools.regions.Region*. The catalogue will be masked according to this region.

infile [str] Input catalogue.

outfile [str] Output catalogue.

negate [bool] If True then pixels *outside* the region are masked. Default = False.

racol, deccol [str] The name of the columns in *table* that should be interpreted as ra and dec. Default = 'ra', 'dec'

See also:

AegeanTools.MIMAS.mask_table()

AegeanTools.catalogs.load_table()

AegeanTools.MIMAS.**mask_file**(*regionfile*, *infile*, *outfile*, *negate=False*)

Created a masked version of file, using a region.

Parameters

regionfile [str] A file which can be loaded as a *AegeanTools.regions.Region*. The image will be masked according to this region.

infile [str] Input FITS image.

outfile [str] Output FITS image.

negate [bool] If True then pixels *outside* the region are masked. Default = False.

See also:

AegeanTools.MIMAS.mask_plane()

AegeanTools.MIMAS.**mask_plane**(*data*, *wcs*, *region*, *negate=False*)

Mask a 2d image (data) such that pixels within 'region' are set to nan.

Parameters

data [2d-array] Image array.

wcs [astropy.wcs.WCS] WCS for the image in question.

region [*AegeanTools.regions.Region*] A region within which the image pixels will be masked.

negate [bool] If True then pixels *outside* the region are masked. Default = False.

Returns

masked [2d-array] The original array, but masked as required.

AegeanTools.MIMAS.**mask_table**(*region*, *table*, *negate=False*, *racol='ra'*, *deccol='dec'*)

Apply a given mask (region) to the table, removing all the rows with ra/dec inside the region If negate=False then remove the rows with ra/dec outside the region.

Parameters

region [*AegeanTools.regions.Region*] Region to mask.

table [Astropy.table.Table] Table to be masked.

negate [bool] If True then pixels *outside* the region are masked. Default = False.

racol, deccol [str] The name of the columns in *table* that should be interpreted as ra and dec. Default = 'ra', 'dec'

Returns

masked [Astropy.table.Table] A view of the given table which has been masked.

AegeanTools.MIMAS.**mim2fits**(*mimfile*, *fitsfile*)
Convert a MIMAS region (.mim) file into a MOC region (.fits) file.

Parameters

mimfile [str] Input file in MIMAS format.

fitsfile [str] Output file.

AegeanTools.MIMAS.**mim2reg**(*mimfile*, *regfile*)
Convert a MIMAS region (.mim) file into a DS9 region (.reg) file.

Parameters

mimfile [str] Input file in MIMAS format.

regfile [str] Output file.

AegeanTools.MIMAS.**poly2poly**(*line*)
Parse a string of text containing a DS9 description of a polygon.

This function works but is not very robust due to the constraints of healpy.

Parameters

line [str] A string containing a DS9 region command for a polygon.

Returns

poly [[ra, dec, ...]] The coordinates of the polygon.

AegeanTools.MIMAS.**reg2mim**(*regfile*, *mimfile*, *maxdepth*)
Parse a DS9 region file and write a MIMAS region (.mim) file.

Parameters

regfile [str] DS9 region (.reg) file.

mimfile [str] MIMAS region (.mim) file.

maxdepth [str] Depth/resolution of the region file.

AegeanTools.MIMAS.**save_as_image**(*region*, *filename*)
Convert a MIMAS region (.mim) file into a image (eg .png)

Parameters

region [*AegeanTools.regions.Region*] Region of interest.

filename [str] Output filename.

AegeanTools.MIMAS.**save_region**(*region*, *filename*)
Save the given region to a file

Parameters

region [*AegeanTools.regions.Region*] A region.

filename [str] Output file name.

1.10 models

Different types of sources that Aegean is able to fit

class AegeanTools.models.**ComponentSource**

A Gaussian component, aka a source, that was measured by Aegean.

See also:

AegeanTools.flags

Attributes

island [int] The island which this component is part of.

source [int] The source number within the island.

background, local_rms [float] Background and local noise level in the image at the location of this source.

ra, err_ra, dec, err-dec [float] Sky location of the source including uncertainties. Decimal degrees.

ra_str, dec_str [str] Sky location in HH:MM:SS.SS +DD:MM:SS.SS format.

galactic [bool] If true then ra,dec are interpreted as glat,glon instead. Default = False. This is a class attribute, not an instance attribute.

peak_flux, err_peak_flux [float] The peak flux and associated uncertainty.

int_flux, err_int_flux [float] Integrated flux and associated uncertainty.

a, err_a, b, err_b, pa, err_pa: float Shape parameters for this source and associated uncertainties. a/b are in arcsec, pa is in degrees East of North.

residual_mean, residual_std [float] The mean and standard deviation of the model-data for this island of pixels.

psf_a, psf_b, psf_pa [float] The shape parameters for the point spread function (degrees).

flags [int] Flags. See *AegeanTools.flags*.

uuid [str] Unique ID for this source. This is random and not dependent on the source properties.

class AegeanTools.models.**DummyLM**

A dummy copy of the lmfit results, for use when no fitting was done.

Attributes

residual [[np.nan, np.nan]] The residual background and rms.

success: bool False - the fitting has failed.

class AegeanTools.models.**GlobalFittingData**

A class to hold the properties associated with an image. [These were once in the global scope of a monolithic script, hence the name]. (should be) Read-only once created. Used by island fitting subprocesses.

Attributes

img [*AegeanTools.fits_image.FitsImage*] Image that is being analysed, aka the input image.

dcurve [2d-array] Image of +1,0,-1 representing the curvature of the input image.

rmsimg, bkgimg [2d-array] The noise and background of the input image.

hdu_header [HDUHeader] FITS header for the input image.

beam [*AegeanTools.wcs_helpers.Beam*] The synthesized beam of the input image.

data_pix [2d-array] A link to the data array that is contained within the *img*.

dtype [{np.float32, np.float64}] The data type for the input image. Will be enforced upon writing.

region [*AegeanTools.regions.Region*] The region that will be used to limit the source finding of Aegean.

wcshelper [*AegeanTools.wcs_helpers.WCSHelper*] A helper object for WCS operations, created from *hdu_header*.

blank [bool] If true, then the input image will be blanked at the location of each of the measured islands.

class AegeanTools.models.**IslandFittingData**(*isle_num=0*, *i=None*, *scalars=None*, *offsets=(0, 0, 1, 1)*, *doislandflux=False*)

All the data required to fit a single island. Instances are pickled and passed to the fitting subprocesses

Attributes

isle_num [int] island number

i [2d-array] a 2D numpy array of pixel values

scalars [(innerclip, outerclip, max_summits)] Inner and outer clipping limits (sigma), and the maximum number of components that should be fit.

offsets [(xmin, xmax, ymin, ymax)] The offset between the boundaries of the island i, within the larger image.

doislandflux [boolean] If true then also measure properties of the island.

class AegeanTools.models.**IslandSource**

An island of pixels.

See also:

AegeanTools.flags

Attributes

island: int The island identification number

components [int] The number of components that make up this island.

background, local_rms [float] Background and local noise level in the image at the location of this source.

ra, dec [float] Sky location of the brightest pixel in this island. Decimal degrees.

ra_str, dec_str [str] Sky location in HH:MM:SS.SS +DD:MM:SS.SS format.

galactic [bool] If true then ra,dec are interpreted as glat,glon instead. Default = False. This is a class attribute, not an instance attribute.

peak_flux, peak_pixel [float] Value of the brightest pixel for this source.

int_flux, err_int_flux [float] Integrated flux and associated uncertainty.

x_width, y_width [int] The extent of the island in pixel space. The width is of the smallest bounding box.

max_angular_size [float] The maximum angular size of the island in sky coordinates (degrees).

pa [float] Position angle for the line representing the maximum angular size.

pixels [int] The number of pixels covered by this island.

area [float] The area of this island in sky coordinates (square degrees).

beam_area [float] The area of the synthesized beam of the image at the location of the brightest pixel. (square degrees).

eta [float] A factor that accounts for the difference between the integrated flux counted by summing pixels, and the integrated flux that would be produced by integrating an appropriately sized Gaussian.

extent [float]

contour [list] A list of pixel coordinates that mark the pixel boundaries for this island of pixels.

max_angular_size_anchors [[x1, y1, x2, y2]] The end points of the vector that describes the maximum angular size of this island.

flags [int] Flags. See *AegeanTools.flags*.

uuid [str] Unique ID for this source. This is random and not dependent on the source properties.

class AegeanTools.models.**PixelIsland**(*dim=2*)

An island of pixels within an image or cube

Attributes

dim [int] The number of dimensions of this island. dim >=2, default is 2 (ra/dec).

bounding_box [[(min, max), (min, max), ...]] A bounding box for this island. len(bounding_box)==dim.

mask [np.array(dtype=bool)] A mask that represents the island within the bounding box.

calc_bounding_box(*data*, *offsets*)

Compute the bounding box for a data cube of dimension dim. The bounding box will be the smallest nd-cube that bounds the non-zero entries of the cube.

Parameters

data [np.ndarray] Data array with dimension equal to self.dim

offsets [[xmin, ymin, ...]] The offset between the image zero index and the zero index of data. len(offsets)==dim

set_mask(*data*)

Parameters

data [np.array]

class AegeanTools.models.**SimpleSource**

The base source class for an elliptical Gaussian.

See also:

AegeanTools.flags

Attributes

background, local_rms [float] Background and local noise level in the image at the location of this source.

ra, dec [float] Sky location of this source. Decimal degrees.

galactic [bool] If true then ra,dec are interpreted as glat,glon instead. Default = False. This is a class attribute, not an instance attribute.

peak_flux, err_peak_flux [float] The peak flux value and associated uncertainty.

peak_pixel [float] Value of the brightest pixel for this source.

flags [int] Flags. See *AegeanTools.flags*.

a, b, pa [float] Shape parameters for this source.

uuid [str] Unique ID for this source. This is random and not dependent on the source properties.

`as_list()`
Return an *ordered* list of the source attributes

`AegeanTools.models.classify_catalog(catalog)`
Look at a list of sources and split them according to their class.

Parameters

catalog [iterable] A list or iterable object of {SimpleSource, IslandSource, ComponentSource} objects, possibly mixed. Any other objects will be silently ignored.

Returns

components [list] List of sources of type ComponentSource

islands [list] List of sources of type IslandSource

simples [list] List of source of type SimpleSource

`AegeanTools.models.island_itergen(catalog)`
Iterate over a catalog of sources, and return an island worth of sources at a time. Yields a list of components, one island at a time

Parameters

catalog [iterable] A list or iterable of *AegeanTools.models.ComponentSource* objects.

Yields

group [list] A list of all sources within an island, one island at a time.

1.11 msq2

Provie a class which performs the marching squares algorithm on an image. The desired output is a set of regions / contours.

`class AegeanTools.msq2.MarchingSquares(data)`
Implementation of a marching squares algorithm. With reference to http://devblog.phillipspiess.com/2010/02/23/better-know-an-algorithm-1-marching-squares/ but written in python

`do_march()`
March about and trace the outline of our object

Returns

perimeter [list] The pixels on the perimeter of the region [[x1, y1], ...]

`do_march_all()`
Recursive march in the case that we have a fragmented shape.

Returns

perimeters [[perimeter1, ...]] The perimeters of all the regions in the image.

See also:

AegeanTools.msq2.MarchingSquares.do_march()

`find_start_point()`
Find the first location in our array that is not empty

`solid(x, y)`
Determine whether the pixel x,y is nonzero

Parameters

x, y [int] The pixel of interest.

Returns

solid [bool] True if the pixel is not zero.

`step(x, y)`
Move from the current location to the next

Parameters

x, y [int] The current location

`walk_perimeter(startx, starty)`
Starting at a point on the perimeter of a region, 'walk' the perimeter to return to the starting point. Record the path taken.

Parameters

startx, starty [int] The starting location. Assumed to be on the perimeter of a region.

Returns

perimeter [list] A list of pixel coordinates [[x1,y1], ...] that constitute the perimeter of the region.

1.12 regions

Describe sky areas as a collection of HEALPix pixels

`class AegeanTools.regions.Region(maxdepth=11)`
A Region object represents a footprint on the sky. This is done in a way similar to a MOC. The region is stored as a list of healpix pixels, allowing for binary set-like operations.

Attributes

maxdepth [int] The depth or resolution of the region. At the deepest level there will be 4*2**maxdepth pixels on the sky. Default = 11

pixeldict [dict] A dictionary of sets, each set containing the pixels within the region. The sets are indexed by their layer number.

demoted [set] A representation of this region at the deepest layer.

`add_circles`(*ra_cen*, *dec_cen*, *radius*, *depth=None*)
Add one or more circles to this region

Parameters

ra_cen, dec_cen, radius [float or list] The center and radius of the circle or circles to add to this region.

depth [int] The depth at which the given circles will be inserted.

`add_pixels`(*pix*, *depth*)
Add one or more HEALPix pixels to this region.

Parameters

pix [int or iterable] The pixels to be added

depth [int] The depth at which the pixels are added.

`add_poly`(*positions*, *depth=None*)
Add a single polygon to this region.

Parameters

positions [[[ra, dec], ...]] Positions for the vertices of the polygon. The polygon needs to be convex and non-intersecting.

depth [int] The deepth at which the polygon will be inserted.

`get_area`(*degrees=True*)
Calculate the total area represented by this region.

Parameters

degrees [bool] If True then return the area in square degrees, otherwise use steradians. Default = True.

Returns

area [float] The area of the region.

`get_demoted`()
Get a representation of this region at the deepest level.

Returns

demoted [set] A set of pixels, at the highest resolution.

`intersect`(*other*)
Combine with another Region by performing intersection on their pixlists.

Requires both regions to have the same maxdepth.

Parameters

other [*AegeanTools.regions.Region*] The region to be combined.

`classmethod load`(*mimfile*)
Create a region object from the given file.

Parameters

mimfile [str] File to load.

Returns

region [*AegeanTools.regions.Region*] A region object

`static radec2sky`(*ra*, *dec*)

Convert [ra], [dec] to [(ra[0], dec[0]),....] and also ra,dec to [(ra,dec)] if ra/dec are not iterable

Parameters

ra, dec [float or iterable] Sky coordinates

Returns

sky [numpy.array] array of (ra,dec) coordinates.

`save`(*mimfile*)

Save this region to a file

Parameters

mimfile [str] File to write

`static sky2ang`(*sky*)

Convert ra,dec coordinates to theta,phi coordinates ra -> phi dec -> theta

Parameters

sky [numpy.array] Array of (ra,dec) coordinates. See *AegeanTools.regions.Region.radec2sky()*

Returns

theta_phi [numpy.array] Array of (theta,phi) coordinates.

`classmethod sky2vec`(*sky*)

Convert sky positions in to 3d-vectors on the unit sphere.

Parameters

sky [numpy.array] Sky coordinates as an array of (ra,dec)

Returns

vec [numpy.array] Unit vectors as an array of (x,y,z)

See also:

AegeanTools.regions.Region.vec2sky()

`sky_within`(*ra*, *dec*, *degin=False*)

Test whether a sky position is within this region

Parameters

ra, dec [float] Sky position.

degin [bool] If True the ra/dec is interpreted as degrees, otherwise as radians. Default = False.

Returns

within [bool] True if the given position is within one of the region's pixels.

symmetric_difference(*other*)
Combine with another Region by performing the symmetric difference of their pixlists.

Requires both regions to have the same maxdepth.

Parameters

other [*AegeanTools.regions.Region*] The region to be combined.

union(*other*, *renorm=True*)
Add another Region by performing union on their pixlists.

Parameters

other [*AegeanTools.regions.Region*] The region to be combined.

renorm [bool] Perform renormalisation after the operation? Default = True.

classmethod vec2sky(*vec*, *degrees=False*)
Convert [x,y,z] vectors into sky coordinates ra,dec

Parameters

vec [numpy.array] Unit vectors as an array of (x,y,z)

degrees

Returns

sky [numpy.array] Sky coordinates as an array of (ra,dec)

See also:

AegeanTools.regions.Region.sky2vec()

without(*other*)
Subtract another Region by performing a difference operation on their pixlists.

Requires both regions to have the same maxdepth.

Parameters

other [*AegeanTools.regions.Region*] The region to be combined.

write_fits(*filename*, *moctool=''*)
Write a fits file representing the MOC of this region.

Parameters

filename [str] File to write

moctool [str] String to be written to fits header with key "MOCTOOL". Default = ''

write_reg(*filename*)
Write a ds9 region file that represents this region as a set of diamonds.

Parameters

filename [str] File to write

1.13 source_finder

The Aegean source finding program.

class AegeanTools.source_finder.**SourceFinder**(***kwargs*)

The Aegean source finding algorithm

Attributes

global_data [*AegeanTools.models.GlobalFittingData*] State holder for properties.

sources [list] List of sources that have been found/measured.

log [logging.log] Logger to use. Default = None

estimate_lmfit_parinfo(*data*, *rmsimg*, *curve*, *beam*, *innerclip*, *outerclip=None*, *offsets=(0, 0)*, *max_summits=None*)

Estimates the number of sources in an island and returns initial parameters for the fit as well as limits on those parameters.

Parameters

data [2d-array] (sub) image of flux values. Background should be subtracted.

rmsimg [2d-array] Image of 1sigma values

curve [2d-array] Image of curvature values [-1,0,+1]

beam [AegeanTools.fits_image.Beam] The beam information for the image.

innerclip, outerclip [float] Inerr and outer level for clipping (sigmas).

offsets [(int, int)] The (x,y) offset of data within it's parent image

max_summits [int] If not None, only this many summits/components will be fit. More components may be present in the island, but subsequent components will not have free parameters.

Returns

model [lmfit.Parameters] The initial estimate of parameters for the components within this island.

find_sources_in_image(*filename*, *hdu_index=0*, *outfile=None*, *rms=None*, *bkg=None*, *max_summits=None*, *innerclip=5*, *outerclip=4*, *cores=None*, *rmsin=None*, *bkgin=None*, *beam=None*, *doislandflux=False*, *nopositive=False*, *nonegative=False*, *mask=None*, *imgpsf=None*, *blank=False*, *docov=True*, *cube_index=None*, *progress=False*)

Run the Aegean source finder.

Parameters

filename [str or HDUList] Image filename or HDUList.

hdu_index [int] The index of the FITS HDU (extension).

outfile [str] file for printing catalog (NOT a table, just a text file of my own design)

rms [float] Use this rms for the entire image (will also assume that background is 0)

max_summits [int] Fit up to this many components to each island (extras are included but not fit)

innerclip, outerclip [float] The seed (inner) and flood (outer) clipping level (sigmas).

cores [int] Number of CPU cores to use. None means all cores.

rmsin, bkgin [str or HDUList] Filename or HDUList for the noise and background images. If either are None, then it will be calculated internally.

beam [(major, minor, pa)] Floats representing the synthesised beam (degrees). Replaces whatever is given in the FITS header. If the FITS header has no BMAJ/BMIN then this is required.

doislandflux [bool] If True then each island will also be characterized.

nopositive, nonegative [bool] Whether to return positive or negative sources. Default nopositive=False, nonegative=True.

mask [str] The filename of a region file created by MIMAS. Islands outside of this region will be ignored.

imgpsf [str or HDUList] Filename or HDUList for a psf image.

blank [bool] Cause the output image to be blanked where islands are found.

docov [bool] If True then include covariance matrix in the fitting process. Default=True

cube_index [int] For image cubes, cube_index determines which slice is used.

progress [bool] If true then show a progress bar when fitting island groups

Returns

sources [list] List of sources found.

`load_globals`(*filename*, *hdu_index=0*, *bkgin=None*, *rmsin=None*, *beam=None*, *verb=False*, *rms=None*, *bkg=None*, *cores=1*, *do_curve=False*, *mask=None*, *psf=None*, *blank=False*, *docov=True*, *cube_index=None*)

Populate the global_data object by loading or calculating the various components

Parameters

filename [str or HDUList] Main image which source finding is run on

hdu_index [int] HDU index of the image within the fits file, default is 0 (first)

bkgin, rmsin [str or HDUList] background and noise image filename or HDUList

beam [`AegeanTools.fits_image.Beam`] Beam object representing the synthsized beam. Will replace what is in the FITS header.

verb [bool] Verbose. Write extra lines to INFO level log.

rms, bkg [float] A float that represents a constant rms/bkg levels for the image. Default = None, which causes the rms/bkg to be loaded or calculated.

cores [int] Number of cores to use if different from what is autodetected.

do_curve [bool] If True a curvature map will be created, default=True.

mask [str or *AegeanTools.regions.Region*] filename or Region object

psf [str or HDUList] Filename or HDUList of a psf image

blank [bool] True = blank output image where islands are found. Default = False.

docov [bool] True = use covariance matrix in fitting. Default = True.

cube_index [int] For an image cube, which slice to use.

priorized_fit_islands(*filename*, *catalogue*, *hdu_index=0*, *outfile=None*, *bkgin=None*, *rmsin=None*, *cores=1*, *rms=None*, *bkg=None*, *beam=None*, *imgpsf=None*, *catpsf=None*, *stage=3*, *ratio=None*, *outerclip=3*, *doregroup=True*, *regroup_eps=None*, *docov=True*, *cube_index=None*, *progress=False*)

Take an input catalog, and image, and optional background/noise images fit the flux and ra/dec for each of the given sources, keeping the morphology fixed

Multiple cores can be specified, and will be used.

Parameters

filename [str or HDUList] Image filename or HDUList.

catalogue [str or list] Input catalogue file name or list of ComponentSource objects.

hdu_index [int] The index of the FITS HDU (extension).

outfile [str] file for printing catalog (NOT a table, just a text file of my own design)

rmsin, bkgin [str or HDUList] Filename or HDUList for the noise and background images. If either are None, then it will be calculated internally.

cores [int] Number of CPU cores to use. None means all cores.

rms [float] Use this rms for the entire image (will also assume that background is 0)

beam [(float, float, float)] (major, minor, pa) representing the synthesised beam (degrees). Replaces whatever is given in the FITS header. If the FITS header has no BMAJ/BMIN then this is required.

imgpsf [str or HDUList] Filename or HDUList for a psf image.

catpsf [str or HDUList] Filename or HDUList for the catalogue psf image.

stage [int] Refitting stage

ratio [float] If not None - ratio of image psf to catalog psf, otherwise interpret from catalogue or image if possible

outerclip [float] The flood (outer) clipping level (sigmas).

doregroup [bool, Default=True] Relabel all the islands/groups to ensure that nearby components are jointly fit.

regroup_eps: float, Default=None The linking parameter for regouping. Components that are closer than this distance (in arcmin) will be jointly fit. If NONE, then use 4x the average source major axis size (after rescaling if required).

docov [bool, Default=True] If True then include covariance matrix in the fitting process.

cube_index [int, Default=None] For image cubes, slice determines which slice is used.

progress [bool, Default=True] Show a progress bar when fitting island groups

Returns

sources [list] List of sources measured.

result_to_components(*result*, *model*, *island_data*, *isflags*)

Convert fitting results into a set of components

Parameters

result [lmfit.MinimizerResult] The fitting results.

model [lmfit.Parameters] The model that was fit.

island_data [*AegeanTools.models.IslandFittingData*] Data about the island that was fit.

isflags [int] Flags that should be added to this island in addition to those within the model)

Returns

sources [list] A list of components, and islands if requested.

`save_background_files`(*image_filename*, *hdu_index=0*, *bkgin=None*, *rmsin=None*, *beam=None*, *rms=None*, *bkg=None*, *cores=1*, *outbase=None*)

Generate and save the background and RMS maps as FITS files. They are saved in the current directly as aegean-background.fits and aegean-rms.fits.

Parameters

image_filename [str or HDUList] Input image.

hdu_index [int] If fits file has more than one hdu, it can be specified here. Default = 0.

bkgin, rmsin [str or HDUList] Background and noise image filename or HDUList

beam [`AegeanTools.fits_image.Beam`] Beam object representing the synthsized beam. Will replace what is in the FITS header.

rms, bkg [float] A float that represents a constant rms/bkg level for the image. Default = None, which causes the rms/bkg to be loaded or calculated.

cores [int] Number of cores to use if different from what is autodetected.

outbase [str] Basename for output files.

`save_image`(*outname*)

Save the image data. This is probably only useful if the image data has been blanked.

Parameters

outname [str] Name for the output file.

`AegeanTools.source_finder.characterise_islands`(*islands*, *im*, *bkg*, *rms*, *wcshelper*, *err_type='best'*, *max_summits=None*, *do_islandfit=False*)

Do the source characterisation based on the initial estimate of the island properties.

Parameters

islands [[lmfit.Parameters, ...]] The initial estimate of parameters for the components within each island.

im, bkg, rms [np.ndarray] The image, background, and noise maps

wcshelper [*AegeanTools.wcs_helpers.WCSHelper*] A wcs helper object

err_type [str or None] The method for calculating uncertainties on parameters: 'best' - Uncertainties measured based on covariance matrix of the

fit and of the data See Hancock et al. 2018 for a description of this process.

'condon' - Uncertainties are *calculated* based on Condon'98 (?year) 'raw' - uncertainties directly from the covariance matrix only 'none' or None - No uncertainties, all will be set to -1.

max_summits [int] The maximum number of summits that will be fit. The final model may contain additional components but only the first few will be fit.

do_islandfit [bool] If True, then also characterise islands as well as components. Default=False.

Returns

sources [[*AegeanTools.models.SimpleSource*, ...]] A list of characterised sources of type AegeanTools.models.impleSource, *AegeanTools.models.ComponentSource*, or *AegeanTools.models.IslandSource*.

AegeanTools.source_finder.**check_cores**(*cores*)

Determine how many cores we are able to use. Return 1 if we are not able to make a queue via pprocess.

Parameters

cores [int] The number of cores that are requested.

Returns

cores [int] The number of cores available.

AegeanTools.source_finder.**estimate_parinfo_image**(*islands*, *im*, *rms*, *wcshelper*, *max_summits=None*, *log=<Logger dummy (WARNING)>*)

Estimate the initial parameters for fitting for each of the islands of pixels. The source sizes will be initialised as the psf of the image, which is either determined by the WCS of the image file or the psf map if one is supplied.

Parameters

islands [[AegeanTools.models.IslandFittingData, ...]] A list of islands which will be converted into groups of sources

im, rms [numpy.ndarray] The image and noise maps

wcshelper [*AegeanTools.wcs_helpers.WCSHelper*] A wcshelper object valid for the image map

max_summits [int or None] The maximum number of summits that will be fit. Any in addition to this will be estimated but their parameters will have vary=False.

log [*logging.Logger* or None] For handling logs (or not)

max_summits [int] The maximum number of summits that will be fit. Any in addition to this will be estimated but their parameters will have vary=False.

log [*logging.Logger* or None] For handling logs (or not)

Returns

sources [[lmfit.Parameters, ...]] The initial estimate of parameters for the components within each island.

AegeanTools.source_finder.**find_islands**(*im*, *bkg*, *rms*, *seed_clip=5.0*, *flood_clip=4.0*, *log=<Logger dummy (WARNING)>*)

This function designed to be run as a stand alone process

Parameters

im, bkg, rms [numpy.ndarray] Image, background, and rms maps

seed_clip, flood_clip [float] The seed clip which is used to create islands, and flood clip which is used to grow islands. The units are in SNR.

log [*logging.Logger* or None] For handling logs (or not)

Returns

islands [[*AegeanTools.models.PixelIsland*, ...]] a list of islands

AegeanTools.source_finder.**fit_islands_parinfo**(*models*, *im*, *rms*, *wcshelper*)

Turn a list of sources into a set of islands and parameter estimates which can then be characterised.

Parameters

models [[lmfit.Parinfo, ...]] A list of sources in the catalogue.

im [np.ndarray] The image map

wcshelper [*AegeanTools.wcs_helpers.WCSHelper*] A wcs object valid for the image map

Returns

islands [[AegeanTools.models.SimpleSource, ...]] a list of islands

AegeanTools.source_finder.**fix_shape**(*source*)
Ensure that a>=b for a given source object. If a<b then swap a/b and increment pa by 90. err_a/err_b are also swapped as needed.

Parameters

source [object] any object with a/b/pa/err_a/err_b properties

AegeanTools.source_finder.**get_aux_files**(*basename*)
Look for and return all the aux files that are associated with this filename. Will look for: - background (_bkg.fits) - rms (_rms.fits) - mask (.mim) - catalogue (_comp.fits) - psf map (_psf.fits)

will return filenames if they exist, or None where they do not.

Parameters

basename [str] The name/path of the input image.

Returns

aux [dict] Dict of filenames or None with keys (bkg, rms, mask, cat, psf)

AegeanTools.source_finder.**pa_limit**(*pa*)
Position angle is periodic with period 180 deg Constrain pa such that -90<pa<=90

Parameters

pa [float] Initial position angle.

Returns

pa [float] Rotate position angle.

AegeanTools.source_finder.**priorized_islands_parinfo**(*sources*, *im*, *wcshelper*, *stage=3*)
Turn a list of sources into a set of islands and parameter estimates which can then be characterised.

Parameters

sources [[AegeanTools.models.SimpleSource, ...]] A list of sources in the catalogue.

im [np.ndarray] The image map

wcshelper [*AegeanTools.wcs_helpers.WCSHelper*] A wcs object valid for the image map

stage [int] The priorized fitting stage which determines which parameters are fit/fixed. One of: 1 - Fit for flux only. All other params are fixed. 2 - Fit for flux and position. Shape parameters are fixed. 3 - Fit for flux, position, and shape.

Returns

islands [[*AegeanTools.models.ComponentSource*, ...]] a list of components

AegeanTools.source_finder.**save_catalogue**(*sources*, *output*, *format=None*)
Write a catalogue of sources

Parameters

sources [[AegeanTools.models.SimpleSource, ...]] A list of characterised sources of type SimpleSource, ComponentSource, or IslandSource.

output [str] Output filename

format [str] A descriptor of the output format. Options are: #TODO add a bunch of options 'auto' or None - infer from filename extension

Returns

None

AegeanTools.source_finder.**theta_limit**(*theta*)

Angle theta is periodic with period pi. Constrain theta such that -pi/2<theta<=pi/2.

Parameters

theta [float] Input angle.

Returns

theta [float] Rotate angle.

1.14 wcs_helpers

This module contains two classes that provide WCS functions that are not part of the WCS toolkit, as well as some wrappers around the provided tools to make them a lot easier to use.

class AegeanTools.wcs_helpers.**Beam**(*a*, *b*, *pa*)

Small class to hold the properties of the beam. Properties are a,b,pa. No assumptions are made as to the units, but both a and b have to be >0.

class AegeanTools.wcs_helpers.**WCSHelper**(*wcs*, *beam*, *pixscale*, *refpix*, *psf_file=None*)

A wrapper around astropy.wcs that provides extra functionality, and hides the c/fortran indexing troubles.

Additionally allow psf information to be described in a map instead of the fits header of the image.

Useful functions not provided by astropy.wcs

- sky2pix/pix2sky functions for vectors and ellipses.
- functions for calculating the beam in sky/pixel coords
- the ability to change the beam according to dec-lat

This class tracks both the synthesized beam of the image (beam) and the point spread function (psf). You may think that these things are the same and interchangeable but they are not always. The beam is defined in the wcs of the image header, while the psf can be defined by providing a new image file with 3 dimensions (ra, dec, psf) where the psf is (a, b, pa). # TODO: Check that the above is consistent with the code, and adjust until they are.

Attributes

wcs [astropy.wcs.WCS] WCS object

beam [*AegeanTools.wcs_helpers.Beam*] The synthesized beam as defined by the fits header (at the reference location).

pixscale [(float, float)] The pixel scale at the reference location (degrees)

refpix [(float, float)] The reference location in pixel coordinates

psf_file [str] Filename for a psf map

psf_map [np.ndarray] A map of the psf as a function of sky position.

psf_wcs [np.ndarray] The WCS object for the psf map

classmethod from_file(*filename*, *beam=None*, *psf_file=None*)
Create a new WCSHelper class from a given fits file.

Parameters

filename [string] The file to be read

beam [*AegeanTools.wcs_helpers.Beam* or None] The synthesized beam. If the supplied beam is None then one is constructed form the header.

psf_file [str] Filename for a psf map

Returns

obj [*AegeanTools.wcs_helpers.WCSHelper*] A helper object

classmethod from_header(*header*, *beam=None*, *psf_file=None*)
Create a new WCSHelper class from the given header.

Parameters

header [*astropy.fits.HDUHeader* or string] The header to be used to create the WCS helper

beam [*AegeanTools.wcs_helpers.Beam* or None] The synthesized beam. If the supplied beam is None then one is constructed form the header.

psf_file [str] Filename for a psf map

Returns

obj [*AegeanTools.wcs_helpers.WCSHelper*] A helper object.

get_beamarea_deg2(*ra*, *dec*)
Calculate the area of the synthesized beam in square degrees.

Parameters

ra, dec [float] The sky coordinates at which the calculation is made.

Returns

area [float] The beam area in square degrees.

get_beamarea_pix(*ra*, *dec*)
Calculate the beam area in square pixels.

Parameters

ra, dec [float] The sky coordinates at which the calculation is made

dec

Returns

area [float] The beam area in square pixels.

get_psf_pix2pix(*x*, *y*)
Determine the beam in pixels at the given location in pixel coordinates. The psf is in pixel coordinates.

Parameters

x , y [float] The pixel coordinates at which the beam is determined.

Returns

a, b, theta [(float, float, float)] The psf semi-major axis (pixels), semi-minor axis (pixels), and rotation angle (degrees). If a psf is defined then it is the psf that is returned, otherwise the image restoring beam is returned.

`get_psf_sky2pix`(*ra*, *dec*)
Determine the psf (a,b,pa) at a given sky location. The psf is in pixel coordinates.

Parameters

ra, dec [float] The sky position (degrees).

Returns

a, b, pa [(float, float, float)] The psf semi-major axis (pixels), semi-minor axis (pixels), and rotation angle (degrees). If a psf is defined then it is the psf that is returned, otherwise the image restoring beam is returned.

`get_psf_sky2sky`(*ra*, *dec*)
Determine the point spread function in sky coordinates at a given sky location. The psf is returned in degrees.

Parameters

ra, dec [float] The sky position (degrees).

Returns

a, b, pa [(float, float, float)] The psf semi-major axis, semi-minor axis, and position angle in (degrees). If a psf is defined then it is the psf that is returned, otherwise the image restoring beam is returned.

`get_skybeam`(*ra*, *dec*)
Determine the beam at the given sky location.

Parameters

ra, dec [float] The sky coordinates at which the beam is determined.

Returns

beam [*AegeanTools.wcs_helpers.Beam*] A beam object, with a/b/pa in sky coordinates

`pix2sky`(*pixel*)
Convert pixel coordinates into sky coordinates. Computed on the image wcs.

Parameters

pixel [(float, float)] The (x,y) pixel coordinates

Returns

sky [(float, float)] The (ra,dec) sky coordinates in degrees

`pix2sky_ellipse`(*pixel*, *sx*, *sy*, *theta*)
Convert an ellipse from pixel to sky coordinates.

Parameters

pixel [(float, float)] The (x, y) coordinates of the center of the ellipse.

sx, sy [float] The major and minor axes (FHWM) of the ellipse, in pixels.

theta [float] The rotation angle of the ellipse (degrees). theta = 0 corresponds to the ellipse being aligned with the x-axis.

Returns

ra, dec [float] The (ra, dec) coordinates of the center of the ellipse (degrees).

a, b [float] The semi-major and semi-minor axis of the ellipse (degrees).

pa [float] The position angle of the ellipse (degrees).

`pix2sky_vec`(*pixel*, *r*, *theta*)

Given and input position and vector in pixel coordinates, calculate the equivalent position and vector in sky coordinates.

Parameters

pixel [(float, float)] origin of vector in pixel coordinates

r [float] magnitude of vector in pixels

theta [float] angle of vector in degrees

Returns

ra, dec [float] The (ra, dec) of the origin point (degrees).

r, pa [float] The magnitude and position angle of the vector (degrees).

`psf_sky2pix`(*pos*)

Convert sky coordinates into pixel coordinates. Computed on the psf wcs.

Parameters

pos [(float, float)] The (ra, dec) sky coordinates (degrees)

Returns

pixel [(float, float)] The (x,y) pixel coordinates

`sky2pix`(*pos*)

Convert sky coordinates into pixel coordinates. Computed on the image wcs.

Parameters

pos [(float, float)] The (ra, dec) sky coordinates (degrees)

Returns

pixel [(float, float)] The (x,y) pixel coordinates

`sky2pix_ellipse`(*pos*, *a*, *b*, *pa*)

Convert an ellipse from sky to pixel coordinates.

Parameters

pos [(float, float)] The (ra, dec) of the ellipse center (degrees).

a, b, pa: float The semi-major axis, semi-minor axis and position angle of the ellipse (degrees).

Returns

x, y [float] The (x, y) pixel coordinates of the ellipse center.

sx, sy [float] The major and minor axes (FWHM) in pixels.

theta [float] The rotation angle of the ellipse (degrees). theta = 0 corresponds to the ellipse being aligned with the x-axis.

`sky2pix_vec`(*pos*, *r*, *pa*)

Convert a vector from sky to pixel coords. The vector has a magnitude, angle, and an origin on the sky.

Parameters

pos [(float, float)] The (ra, dec) of the origin of the vector (degrees).

r [float] The magnitude or length of the vector (degrees).

pa [float] The position angle of the vector (degrees).

Returns

x, y [float] The pixel coordinates of the origin.

r, theta [float] The magnitude (pixels) and angle (degrees) of the vector.

sky_sep(*pix1*, *pix2*)
calculate the GCD sky separation (degrees) between two pixels.

Parameters

pix1, pix2 [(float, float)] The (x,y) pixel coordinates for the two positions.

Returns

dist [float] The distance between the two points (degrees).

AegeanTools.wcs_helpers.**fix_aips_header**(*header*)
Search through an image header. If the keywords BMAJ/BMIN/BPA are not set, but there are AIPS history cards, then we can populate the BMAJ/BMIN/BPA. Fix the header if possible, otherwise don't. Either way, don't complain.

Parameters

header [astropy.io.fits.HDUHeader] Fits header which may or may not have AIPS history cards.

Returns

header [astropy.io.fits.HDUHeader] A header which has BMAJ, BMIN, and BPA keys, as well as a new HISTORY card.

AegeanTools.wcs_helpers.**get_beam**(*header*)
Create a *AegeanTools.wcs_helpers.Beam* object from a fits header.

BPA may be missing but will be assumed to be zero.

if BMAJ or BMIN are missing then return None instead of a beam object.

Parameters

header [astropy.io.fits.HDUHeader] The fits header.

Returns

beam [*AegeanTools.wcs_helpers.Beam*] Beam object, with a, b, and pa in degrees.

AegeanTools.wcs_helpers.**get_pixinfo**(*header*)
Return some pixel information based on the given hdu header pixarea - the area of a single pixel in deg2 pixscale - the side lengths of a pixel (assuming they are square)

Parameters

header [astropy.io.fits.HDUHeader] FITS header information

Returns

pixarea [float] The are of a single pixel at the reference location, in square degrees.

pixscale [(float, float)] The pixel scale in degrees, at the reference location.

Notes

The reference location is not always at the image center, and the pixel scale/area may change over the image, depending on the projection.

CHAPTER

TWO

AEGEANTOOLS SCRIPTS

The following scripts are provided as part of the AegeanTools package:

- *aegean* - Aegean source finding
- *BANE* - Background and Noise Estimation
- *MIMAS* - Multi-resolution Image Mask for Aegean Software
- *AeRes* - Aegean Residuals
- *AeReg* - Aegean Regrouping
- *SR6* - Shrink Ray

CHAPTER

THREE

AEREG

The regrouping and rescaling operations that were introduced as part of the priorized fitting have been moved into the cluster module. The script `AeReg` will allow a user to access these operations from the command line such that they can see how the regrouping and rescaling operations will work before having to do the priorized fitting.

```
usage: regroup [-h] --input INPUT --table TABLES [--eps EPS] [--noregroup] [--ratio␣
↪RATIO] [--psfheader PSFHEADER] [--debug]

optional arguments:
  -h, --help            show this help message and exit

Required:
  --input INPUT         The input catalogue.
  --table TABLES        Table outputs, format inferred from extension.

Clustering options:
  --eps EPS             The grouping parameter epsilon (~arcmin)
  --noregroup           Do not perform regrouping (default False)

Scaling options:
  --ratio RATIO         The ratio of synthesized beam sizes (image psf / input catalog␣
↪psf).
  --psfheader PSFHEADER
                        A file from which the *target* psf is read.

Other options:
  --debug               Debug mode.
```

CHAPTER

FOUR

AERES

If you want to get residual maps, or model maps, from Aegean then this tool is what you are looking for.

AeRes will take an image, and Aegean catalog, and write a new image with all the sources removed. You can also ask for an image that has just the sources in it.

You can use AeRes as shown below:

```
Usage: AeRes -c input.vot -f image.fits -r residual.fits [-m model.fits]

Options:
  -h, --help            show this help message and exit
  -c CATALOG, --catalog=CATALOG
                        Catalog in a format that Aegean understands. RA/DEC
                        should be in degrees, a/b/pa should be in
                        arcsec/arcsec/degrees.
  -f FITSFILE, --fitsimage=FITSFILE
                        Input fits file.
  -r RFILE, --residual=RFILE
                        Output residual fits file.
  -m MFILE, --model=MFILE
                        Output model file [optional].
  --add                 Add components instead of subtracting them.
  --mask                Instead of subtracting sources, just mask them
  --sigma=SIGMA         If masking, pixels above this SNR are masked (requires
                        input catalogue to list rms)
  --frac=FRAC           If masking, pixels above frac*peak_flux are masked for
                        each source
  --debug               Debug mode.
```

The acceptable formats for the catalogue file are anything that Aegean can write. Use `aegean.py --tformats` to see the formats that Aegean can support on your machine. Usually the best idea is to just edit a table that Aegean has created.

CHAPTER

FIVE

BANE

5.1 Motivation

Aegean has an inbuilt background and noise calculation algorithm (the zones algorithm) which is very basic and is useful for images that have a slowly changing background and noise. For images with more complicated background and noise statistics it is advised that you use an external program to pre-compute these maps and then feed them into Aegean with the –background and –noise flags. Since I have not come across a program that can calculate these images in a speedy manner I have built one myself.

5.2 Aim

The quick-and-dirty method for calculating the background and noise of an image is to pass a sliding boxcar filter over the image and, for each pixel, calculate the mean and standard deviation of all pixels within a box centred on that pixel. The problem with this approach is two-fold: one - although it is easy to code it is very time consuming, and two - the standard deviation is biased in the presence of sources.

The aim of BANE is to provide an accurate measure of the background and noise properties of an image, and to do so in a relatively short amount of time.

5.3 Methodology

There are two main techniques that BANE uses to reduce the compute time for a background and noise calculation, whilst retaining a high level of accuracy.

- Since radio images have a high level of correlation between adjacent pixels, BANE does not calculate the mean and standard deviation for every pixel. It will calculate these quantities on a sparse grid of pixels and then interpolate to give the final background and noise images. For a grid spacing of 5x5 pixels this reduces the total computations by a factor of 25, with only a small amount of time required for interpolation.
- To avoid contamination from source pixels BANE performs sigma clipping. Pixels that are greater than 3sigma from the mean are masked, and this processes is repeated 3 times. The non-masked pixels are then used to calculate the median and std which are equated to be the background and rms.

BANE offers the user a set of parameters that can be used to tune the speed/accuracy to a users desire. The parameters are the grid spacing (in each of the x,y directions), and the size of the box (again in x,y directions) over which the background and noise is calculated. A grid spacing of 1x1 is equivalent to a traditional box-car smooth using the median and std.

Since we define the noise to be the variance about the median, it is necessary for BANE to make two passes over the data: the first pass calculates the background level, and the second pass calculates the deviation from this background

level. This requirement doubles the run time of BANE, however for images where the background level is known to be slowly changing (on scales of the box size), a single pass is all that is required.

5.4 Processing steps

The implementation of the process isn't that important but the idea is as follows:

1. select every Nth pixel in the image to form a grid (where N is the grid size, and can be different in the x and y directions).
2. around each grid point draw a box that is MxM pixels wide (where M is the box size, and can be different in the x,y directions).
3. do sigma clipping (3 rounds at 3sigma) to remove the contribution of source pixels
4. calculate the median of all pixels within the box and use that as the background
5. run a linear interpolation between the grid points to make a background image
6. calculate a background subtracted image (data-background)
7. repeat steps 1-4 on the background subtracted image, but instead of calculating the median, use the std.

5.5 Usage

The usage of BANE is described in the help text as follows:

```
usage: BANE [-h] [--out OUT_BASE] [--grid STEP_SIZE STEP_SIZE]
            [--box BOX_SIZE BOX_SIZE] [--cores CORES] [--stripes STRIPES]
            [--nomask] [--noclobber] [--debug] [--compress] [--cite]
            [image]

positional arguments:
  image

optional arguments:
  -h, --help            show this help message and exit

Configuration Options:
  --out OUT_BASE        Basename for output images default:
                        FileName_{bkg,rms}.fits
  --grid STEP_SIZE STEP_SIZE
                        The [x,y] size of the grid to use. Default = ~4* beam
                        size square.
  --box BOX_SIZE BOX_SIZE
                        The [x,y] size of the box over which the rms/bkg is
                        calculated. Default = 5*grid.
  --cores CORES         Number of cores to use. Default = all available.
  --stripes STRIPES     Number of slices.
  --nomask              Don't mask the output array [default = mask]
  --noclobber           Don't run if output files already exist. Default is to
                        run+overwrite.
  --debug               debug mode, default=False
```

(continues on next page)

(continued from previous page)

```
  --compress            Produce a compressed output file.
  --cite                Show citation information.
```

5.6 Description of options

- `--compress`: This option causes the output files to be very small. This compression is done by writing a fits image without any interpolation. Files that are produced in this way have extra keys in their fits header, which are recognized by Aegean. When compressed files are loaded by aegean they are interpolated (expanded) to their normal sizes.
- `--nomask`: By default BANE will mask the output image to have the same masked pixels as the input image. This means that nan/blank pixels in the input image will be nan in the output image. This doesn't happen if `--compress` is selected.
- `--stripes`: BANE will break the image into this many sections and process each in turn. By default this is equal to the number of cores, so that all stripes will be processed at the same time. By setting stripes>cores it is possible to reduce the instantaneous memory usage of BANE at the cost of run time.

CHAPTER

SIX

MIMAS

6.1 Motivation

Prior to 1.8.1, the Aegean source-finding program operated on the entire input image. To return a list of sources that were contained within a sub region other programs were required (For example stilts). Normally this is not a big concern as the filtering process is rather fast. Since radio telescopes have circular primary beam patterns, and fits images are forced to be rectangular, the images produced by imaging pipelines would contain the area of interest along with some amount of extra sky. If the pixels outside the area of interest are not flagged or masked by the imaging pipeline then extra tools are required. Not being able to find any nifty tools to do this job for me, I decided to create the Milti-resolution Image Mask for Aegean Software - MIMAS. There are three main features that I was looking for, each of which are solved by MIMAS.

6.2 Aims

MIMAS was created with the following three goals in mind:

- to be able to create and manipulate arbitrary shaped regions that could be used to describe areas of sky. The method of manipulation is intended to parallel that of set operations so that you can easily take the intersection, union, or difference of regions, in order to create regions as simple as circles and polygons, to some horrendous thing that describes the sky coverage of a survey.
- to be able to store these regions in a file format that can be easily stored and transmitted.
- to be able to use these regions to mask image files, or to restrict the operation of Aegean to a sub section of a given image.

6.3 Methodology

MIMAS is a wrapper script that uses the regions module that is now part of AegeanTools. The regions module contains a suite of unit tests and a single class called Region. The Region class is built on top of the HealPy module, which is in turn a wrapper around the HEALPix software.

6.4 Usage

MIMAS has five modes of operation:

- create a new region from a combination of: stored regions, circles, or polygons.
- create a new region from a DS9 .reg file
- convert a region.mim file into a .reg format that can be used as an overlay for DS9.
- use a .fits image to create a .mim region file as if the image were a mask
- use a region file and a .fits image to create a new fits image where pixels that are OUTSIDE the given region have been masked.

The operation of MIMAS is explained by the following help text:

```
usage: MIMAS [-h] [-o OUTFILE] [-depth N] [+r [filename [filename ...]]]
             [-r [filename [filename ...]]] [+c ra dec radius]
             [-c ra dec radius] [+p [ra [dec ...]]] [-p [ra [dec ...]]] [-g]
             [--mim2reg region.mim region.reg]
             [--reg2mim region.reg region.mim]
             [--mim2fits region.mim region_MOC.fits]
             [--mask2mim mask.fits region.mim] [--intersect region.mim]
             [--area region.mim] [--maskcat region.mim INCAT OUTCAT]
             [--maskimage region.mim file.fits masked.fits]
             [--fitsmask mask.fits file.fits masked_file.fits] [--negate]
             [--colnames RA_name DEC_name] [--threshold THRESHOLD]
             [--fitsimage] [--debug] [--version] [--cite]

optional arguments:
  -h, --help            show this help message and exit

Creating/modifying regions:
  Must specify -o, plus or more [+-][cr]

  -o OUTFILE            output filename
  -depth N              maximum nside=2**N to be used to represent this
                        region. [Default=8]
  +r [filename [filename ...]]
                        add a region specified by the given file (.mim format)
  -r [filename [filename ...]]
                        exclude a region specified by the given file ( .mim
                        format)
  +c ra dec radius      add a circle to this region (decimal degrees)
  -c ra dec radius      exclude the given circles from a region
  +p [ra [dec ...]]     add a polygon to this region ( decimal degrees)
  -p [ra [dec ...]]     remove a polygon from this region ( decimal degrees)
  -g                    Interpret input coordinates are galactic instead of
                        equatorial.

Using already created regions:
  --mim2reg region.mim region.reg
                        convert region.mim into region.reg
  --reg2mim region.reg region.mim
                        Convert a .reg file into a .mim file
```

(continues on next page)

(continued from previous page)

```
  --mim2fits region.mim region_MOC.fits
                        Convert a .mim file into a MOC.fits file
  --mask2mim mask.fits region.mim
                        Convert a masked image into a region file
  --intersect region.mim, +i region.mim
                        Write out the intersection of the given regions.
  --area region.mim     Report the area of a given region

Masking files with regions:
  --maskcat region.mim INCAT OUTCAT
                        use region.mim as a mask on INCAT, writing OUTCAT
  --maskimage region.mim file.fits masked.fits
                        use region.mim to mask the image file.fits and write
                        masked.fits
  --fitsmask mask.fits file.fits masked_file.fits
                        Use a fits file as a mask for another fits file.
                        Values of blank/nan/zero are considered to be
                        mask=True.
  --negate              By default all masks will exclude data that are within
                        the given region. Use --negate to exclude data that is
                        outside of the region instead.
  --colnames RA_name DEC_name
                        The name of the columns which contain the RA/DEC data.
                        Default=(ra,dec).

Extra options:
  --threshold THRESHOLD
                        Threshold value for input mask file.
  --fitsimage           Save the region as a fits image
  --debug               debug mode [default=False]
  --version             show program's version number and exit
  --cite                Show citation information.

Regions are added/subtracted in the following order, +r -r +c -c +p -p. This
means that you might have to take multiple passes to construct overly
complicated regions.
```

6.4.1 Data model and operation

At the most basic level, The Regions class takes a description of a sky area, either a circle or a polygon, and converts it into a list of HELAPix pixels. These pixels are stored as a python set, making it easy to implement set operations on these regions. HEALpix is a parameterization of the sky that maps diamond shaped regions of equal area, onto a pixel number. There are many interesting properties of the nested HEALPix parameterization that make it easy to implement the Region class. Firstly, HEALPix can represent areas of sky that are as coarse as 1/12th of the entire sky, to regions that are 1/2^30 times smaller. A depth or resolution parameter of 2^12 represents a pixel size of less than one arcminute. By making use of different resolutions of pixels, it is possible to represent any region in an efficient manner. The sky area that is represented by a Region is a combination of pixels of different resolutions, with the smallest resolution being supplied by the user.

6.5 File format

The MIMAS program is able to take a description of a region and save it to a file for use by many programs. Since he underlying data model is a dictionary of sets, the fastest and easiest file format to use is that given by the cPickle module (a binary file). These files are small, fast to read and write, and accurately reproduce the region object that was stored. The MIMAS program writes files with an extension of .mim.

6.6 Interaction with Aegean

Region files with .mim extension that are created by MIMAS can be used to restrict Aegean to the given region of an image. Use the `--region region.mim` option when running Aegean to enable this.

CHAPTER

SEVEN

SR6

BANE is able to output compressed background and rms images using the `--compress` option. If you have a compressed file and want to expand it to have the same number of pixels as your original image then you need to use SR6.

If you have an image that you, for some reason, want to compress using a super-lossy algorithm known as decimation, then SR6 is what you want.

Usage is:

```
usage: SR6.py [-h] [-o OutputFile] [-f factor] [-x]
              [-i {linear,nearest,cubic}] [-m MaskFile] [--debug] [--version]
              infile

optional arguments:
  -h, --help            show this help message and exit

Shrinking and expanding files:
  infile                input filename
  -o OutputFile         output filename
  -f factor             reduction factor
  -x                    Operation is expand instead of compress.
  -i {linear,nearest,cubic}
                        Interpolation method
  -m MaskFile           File to use for masking pixels.
  --debug               Debug output
  --version             show program's version number and exit
```

In order to be able to expand a file, the file needs to have some special keywords in the fits header. These are inserted automatically by BANE, but you could probably fidget them for yourself if you had the need.

You should be able to shrink any file that you choose.

d# aegean

CHAPTER

EIGHT

SIMPLE USAGE

Suggested basic usage (with mostly default parameters):

```
aegean RadioImage.fits --table=Catalog.fits
```

Usage and short description can be obtained via `aegean`, which is replicated below.

```
This is Aegean 2.2.2-(2020-06-07)
usage: aegean [-h] [--find] [--cores CORES] [--hdu HDU_INDEX]
              [--beam BEAM BEAM BEAM] [--telescope TELESCOPE] [--lat LAT]
              [--slice SLICE] [--forcerms RMS] [--forcebkg BKG]
              [--noise NOISEIMG] [--background BACKGROUNDIMG] [--psf IMGPSF]
              [--autoload] [--out OUTFILE] [--table TABLES] [--tformats]
              [--blankout] [--colprefix COLUMN_PREFIX]
              [--maxsummits MAX_SUMMITS] [--seedclip INNERCLIP]
              [--floodclip OUTERCLIP] [--island] [--nopositive] [--negative]
              [--region REGION] [--nocov] [--condon] [--priorized PRIORIZED]
              [--ratio RATIO] [--noregroup] [--input INPUT] [--catpsf CATPSF]
              [--save] [--outbase OUTBASE] [--debug] [--versions] [--cite]
              [image]

positional arguments:
  image

optional arguments:
  -h, --help            show this help message and exit

Configuration Options:
  --find                Source finding mode. [default: true, unless --save or
                        --measure are selected]
  --cores CORES         Number of CPU cores to use for processing [default:
                        all cores]
  --hdu HDU_INDEX       HDU index (0-based) for cubes with multiple images in
                        extensions. [default: 0]
  --beam BEAM BEAM BEAM
                        The beam parameters to be used is "--beam major minor
                        pa" all in degrees. [default: read from fits header].
  --telescope TELESCOPE
                        DEPRECATED
  --lat LAT             DEPRECATED
  --slice SLICE         If the input data is a cube, then this slice will
                        determine the array index of the image which will be
```

(continues on next page)

(continued from previous page)

```
                        processed by aegean

Input Options:                                                                  ␣
↪                                                                               ␣
↪    [29/99695]
  --forcerms RMS        Assume a single image noise of rms. [default: None]
  --forcebkg BKG        Assume a single image background of bkg. [default:
                        None]
  --noise NOISEIMG      A .fits file that represents the image noise (rms),
                        created from Aegean with --save or BANE. [default:
                        none]
  --background BACKGROUNDIMG
                        A .fits file that represents the background level,
                        created from Aegean with --save or BANE. [default:
                        none]
  --psf IMGPSF          A .fits file that represents the local PSF.
  --autoload            Automatically look for background, noise, region, and
                        psf files using the input filename as a hint.
                        [default: don't do this]

Output Options:
  --out OUTFILE         Destination of Aegean catalog output. [default: No
                        output]
  --table TABLES        Additional table outputs, format inferred from
                        extension. [default: none]
  --tformats            Show a list of table formats supported in this
                        install, and their extensions
  --blankout            Create a blanked output image. [Only works if
                        cores=1].
  --colprefix COLUMN_PREFIX
                        Prepend each column name with "prefix_". Default =
                        prepend nothing

Source finding/fitting configuration options:
  --maxsummits MAX_SUMMITS
                        If more than *maxsummits* summits are detected in an
                        island, no fitting is done, only estimation. [default:
                        no limit]
  --seedclip INNERCLIP  The clipping value (in sigmas) for seeding islands.
                        [default: 5]
  --floodclip OUTERCLIP
                        The clipping value (in sigmas) for growing islands.
                        [default: 4]
  --island              Also calculate the island flux in addition to the
                        individual components. [default: false]
  --nopositive          Don't report sources with positive fluxes. [default:
                        false]
  --negative            Report sources with negative fluxes. [default: false]
  --region REGION       Use this regions file to restrict source finding in
                        this image. Use MIMAS region (.mim) files.
  --nocov               Don't use the covariance of the data in the fitting
                        procces. [Default = False]
```

(continues on next page)

(continued from previous page)

```
  --condon              replace errors with those suggested by Condon'97.
                        [Default = False]

Priorized Fitting config options:
  in addition to the above source fitting options

  --priorized PRIORIZED
                        Enable priorized fitting level n=[1,2,3]. 1=fit flux,
                        2=fit flux/position, 3=fit flux/position/shape. See
                        the GitHub wiki for more details.
  --ratio RATIO         The ratio of synthesized beam sizes (image psf / input
                        catalog psf). For use with priorized.
  --noregroup           Do not regroup islands before priorized fitting.
  --input INPUT         If --priorized is used, this gives the filename for a
                        catalog of locations at which fluxes will be measured.
  --catpsf CATPSF       A psf map corresponding to the input catalog. This
                        will allow for the correct resizing of sources when
                        the catalog and image psfs differ.

Extra options:
  --save                Enable the saving of the background and noise images.
                        Sets --find to false. [default: false]
  --outbase OUTBASE     If --save is True, then this specifies the base name
                        of the background and noise images. [default: inferred
                        from input image]
  --debug               Enable debug mode. [default: false]
  --versions            Show the file versions of relevant modules. [default:
                        false]
  --cite                Show citation information.
```

8.1 Example usage:

The following commands can be run from the Aegean directory right out of the box, since they use the test images that are included with Aegean.

- Blind source finding on a test image and report results to stdout
  - `aegean tests/test_files/1904-66_SIN.fits`
- As above but put the results into a text file
  - `aegean tests/test_files1904-66_SIN.fits --table out.csv`
  - The above creates a file `out_comp.csv` for the components that were fit
- Do source finding using a catalog input as the initial parameters for the sources
  - `aegean --priorized 1 --input out_comp.csv tests/test_files/1904-66_SIN.fits`
- Source-find an image and save results to multiple tables
  - `aegean --table catalog.csv,catalog.vot,catalog.fits tests/test_files1904-66_SIN.fits`
- Source-find an image and report the components and islands that were found

  - `aegean --table catalog.vot --island tests/test_files1904-66_SIN.fits`
  - The above creates two files: `catalog_comp.vot` for the components, and `catalog_isle.vot` for the islands. The island column of the components maps to the island column of the islands.

- Source-find a sub-region of an image
  - `aegean --region=region.mim tests/test_files1904-66_SIN.fits`
  - The `region.mim` is a region file in the format created by *MIMAS*

CHAPTER

NINE

OUTPUT FORMATS

Aegean supports a number of output formats. There is the Aegean default, which is a set of columns separated by spaces, with header lines starting with #. The format is described within the output file itself.

The Aegean default output (which goes to STDOUT) does not contain all of the columns listed below. Tables created with the `--table` option contain all the following columns, and as much meta-data as I can manage to pack in.

9.1 Table description

Columns included in output tables have the following columns:

- island - numerical indication of the island from which the source was fitted
- source - source number within that island
- background - background flux density in Jy/beam
- local_rms - local rms in Jy/beam
- ra_str - RA J2000 sexigessimal format
- dec_str - dec J2000 sexigessimal format
- ra - RA in degrees
- err_ra - source-finding fitting error on RA in degrees
- dec - dec in degrees
- err_dec - source-finding fitting error on dec in degrees
- peak_flux - peak flux density in Jy/beam
- err_peak_flux - source-finding fitting error on peak flux density in Jy/beam
- int_flux - integrated flux density in Jy. This is calculated from a/b/peak_flux and the synthesized beam size. It is not fit directly.
- err_int_flux - source-finding fitting error on integrated flux density in Jy
- a - fitted semi-major axis in arcsec
- err_a - error on fitted semi-major axis in arcsec
- b - fitted semi-minor axis in arcsec
- err_b- error on fitted semi-minor axis in arcsec
- pa - fitted position angle in degrees
- err_pa - error on fitted position angle in degrees

- flags - fitting flags (should be all 0 for a good fit)
- residual_mean - mean of the residual flux remaining in the island after fitted Gaussian is subtracted
- residual_std - standard deviation of the residual flux remaining in the island after fitted Gaussian is subtracted
- uuid - a universally unique identifier for this component.
- psf_a - the semi-major axis of the point spread function at this location (arcsec)
- psf_b - the semi-minor axis of the point spread function at this location (arcsec)
- psf_pa - the position angle of the point spread function at this location (arcsec)

An island source will have the following columns:

- island - numerical indication of the island
- components - the number of components within this island
- background - background flux density in Jy/beam
- local_rms - local rms in Jy/beam
- ra_str - RA J2000 sexigessimal format
- dec_str - dec J2000 sexigessimal format
- ra - RA in degrees, of the brightest pixel in the island
- dec - dec in degrees, of the brightest pixel in the island
- peak_flux - peak flux density in Jy/beam, of the brightest pixel in the island
- int_flux - integrated flux density in Jy. Computed by summing pixels in the island, and dividing by the synthesized beam size.
- err_int_flux - Error in the above. Currently Null/None since I don't know how to calculate it.
- eta - a correction factor for int_flux that is meant to account for the flux that was not included because it was below the clipping limit. For a point source the true flux should be int_flux/eta. For extended sources this isn't always the case so use with caution.
- x_width - the extent of the island in the first pixel dimension, in pixels
- y_width - the extent of the island in the second pixel dimension, in pixels
- max_angular_size - the largest distance between to points on the boundary of the island, in degrees.
- pa - the position angle of the max_angular_size line
- pixels - the number of pixels within the island
- beam_area - the area of the synthesized beam (psf) in deg^2
- area - the area of the island in deg^2
- flags - fitting flags (should be all 0 for a good fit)
- uuid - a universally unique identifier for this island.

Note: Column names with 'ra/dec' will be replaced with a 'lat/lon' version if the input image has galactic coordinates in the WCS.

9.2 Table Types

The most useful output is to use tables. Table output is supported by sqlite and astropy and there are three main types: database, votable, and ascii table. Additionally you can output ds9 region files by specifying a .reg file extension.

9.3 Database:

This format requires that the sqlite module is available. This is nearly always true by default, but if you get a crash then check that you can `import sqlite3` from a python terminal before submitting a bug report.

Use `--table out.db` to create a database file containing one table for each source type that was discovered. The table names are 'components', 'islands', and 'simples'. Islands are created when –island is enabled. Components are elliptical gaussian fits and are the default type of source to create. Simples are sources that have been created by using the –measure option.

The columns of the database are self explanatory though they have no units. All fluxes are in Jy, major and minor axes are in arcseconds, and the position angle is in degrees. Errors that would normally be reported as -1 in other formats are stored as nulls in the database tables.

9.4 VOTable:

VOTables are difficult to work with as a human, but super awesome to work with when you have TopCat or some other VO enabled software.

VOTable output is supported by AstroPy (0.3+ I think). If you don't have the right version of AstroPy you can still run Aegean but will not be able to write VOTables. You will be told this when Aegean runs.

Use `--table out.vot` or `--table out.xml` to create a VOTable. Each type of sources that you find will be saved to a different file. Components are saved to out_comp.vot, islands are saved to out_isle.vot, and simple sources will be saved to out_simp.vot (or xml as appropriate). See above for a description of the source types.

9.5 ASCII tables:

ASCII tables are supported by AstroPy (0.4+ I think). As with VOTables, if you don't have the right version of AstroPy then Aegean will still run but it will tell you that you can't write ASCII tables.

There are currently four types of ascii tables that can be used:

- csv -> comma separated values
- tab -> tab separated values
- tex -> LaTeX formatted table
- html -> an html formatted table

Use `--table out.html,out.tex` etc.. for the type of table you are interested in. All tables have column headers that are the same as the variable names. These should be easily discernible. The units are Jy for fluxes, arcseconds for major/minor axes, and degrees for position angles.

As with other table formats the file names will be modified to out_comp.html, out_simp.csv, etc… to denote the different types of sources that are contained within.

9.6 FITS binary tables

use extension `fits` or `FITS` (but not `fit` or `FIT`) to write output tables. Functionality supported by AstroPy. These are binary tables and only the header is human readable.

9.7 DS9 region files

Use extension `reg` for the output table to get `DS9` region files. Both components and islands are supported in this format with `_comp.reg` and `_isle.reg` being the corresponding filenames.

Component sources in the `_comp.reg` files will be shown as ellipses at the location of each component, with the fitted size/orientation. Each ellipse will be annotated with the island and component number such that Island 10, component 0 will appear as `(10,0)`.

Island sources will appear as an outline of the pixels that comprise the island. Each island also has an annotation of the island number, and a diagonal line that represents the largest angular scale.

9.8 Flags

There are six different flags that can be set by Aegean during the source finding and fitting process. In the `STDOUT` version of the Aegean catalog the flags column is written in binary format with a header that read ZWNCPES. These six flags correspond to:

Abbreviation	Name	Numerical value	description
S	FITERRSMAL	1	This flag is set when islands are not able to be fit due to there being fewer pixels than free parameters.
E	FITERR	2	This flag is set when an error occurs during the fitting process. eg the fit doesn't converge.
P	FIXED2PSF	4	If a component is forced to have the shape of the local point spread function then this flag is set. This flag is often set at the same time as the FITERRSMALL, or FIXEDCRICULAR
C	FIXEDCRICULAR	8	If a source is forced to have a circular shape then this flag will be fit.
N	NOTFIT	16	If a component is not fit then this flag is set. This can because and island has reached the `--maxsummits` limit, or `--measure` mode has been invoked.
W	WCSERR	32	If the conversion from pixel to sky coordinates doesn't work then this flag will be set. This can happen for strange projections, but more likely when an image contains pixles that don't have valid sky coordinates.
Z	PRIORIZED	64	This flag is set when the source was fit using priorized fitting.

Note that the flags column will be the summation of the numerical value of the above flags. So flags=7 means that flags P, E, and S have been set. This all makes more sense when you print the flags in binary format.

CHAPTER

TEN

PRIORIZED FITTING

This functionality is designed to take an input catalog of sources (previously created by Aegean), and use the source positions and morphologies to measure the flux of these sources within an image.

When `--priorized x` is invoked the following will happen:

- input catalog is read from the file specified by `--input`. This file needs to contain all the properties of a source, including island numbers and uuids. The easiest way to make these files is to just take the output from Aegean and modify it as needed.
- The sources within the catalog are regrouped. The regrouping will recreate islands of sources based on their positions and morphologies. Sources will be grouped together if they overlap at the FHWM. Note that this is different from the default island grouping that Aegean does, which is based on pixels within an island. If `--noregroup` is set then the island grouping will be based on the (isle,source) id's in the input catalog.
- Fitting will be done on a per island basis, with multiple sources being fit at the same time. The user is able to control which parameters are allowed to vary at this stage by supplying a number x to `--priorized x`.
- Fitting will be done on all pixels that are greater than the `--floodclip` limit. If an island has no pixels above this limit then no output source will be generated. Note the special case of `--floodclip -1` which will simply use all pixels within some rectangular region around each input source.
- Output will be written to files as specified by `--table`.

The parameters that are free/fixed in the fitting process depends on the 'level' of priorized fitting that is requested. Level:

1. Only the flux is allowed to vary. Use this option where you would have otherwise used `--measure`.
2. Flux and positions are allowed to vary, shape is fixed.
3. Everything is allowed to vary.

In the case that the psf of the input catalogue and the supplied image are different there are three options for describing this difference:

1. Use the `--ratio` option, which specifies the ratio of major axes (image psf / catalogue psf). This method works well for small images where the psf doesn't really change over the image, or when the difference is small.
2. Supply a psf map for the input catalogue using the `--catpsf` option. This will give you ultimate fine control over what the psf of your input catalogue is.
3. Include the psf parameters in the input catalogue as columns `psf_a, psf_b, psf_pa`

Note: If you know how to perform the deconvolve-convolve step for two synthesized beams that are not simply scaled versions of each other, then please let me know so that I can implement this.

10.1 Notes on input tables:

Any [[format|Output-Formats]] that Aegean can write, is an acceptable input format. The easiest way to create an input table is to modify and existing catalogue. The following columns are used for priorized fitting:

- Required:
  - `ra`, `dec`, `peak_flux`, `a`, `b`, `pa`
- Optional:
  - `psf_a`, `psf_b`, `psf_pa` used for re-scaling the source shapes.
  - `uuid` copied from input to output catalogues
  - `err_ra`, `err_dec` copied from input to output catalogues when positions are not being fit
  - `err_a`, `err_b`, `err_pa` copied from input to output catalogues when shapes are not being fit

Parameters `a`, `b`, `err_a`, `err_b`, `psf_a`, and `psf_b` all have units of arcsec. Parameters `ra`, `dec`, `pa`,`err_ra`, `err_dec`, and `err_pa` all have units of degrees.

BIBLIOGRAPHY

[1] Rhumb line

[1] Haversine formula

PYTHON MODULE INDEX

a

INDEX

F

G

H

I

J

L

M

N

P

R

S

T

U

V

W