astLib

Matt Hilton & Steven Boada

Jul 08, 2021

CONTENTS:

CHAPTER

ONE

INTRODUCTION AND INSTALLATION

astLib provides some tools for research astronomers who use Python. It is divided into several modules:

- astCalc (general calculations, e.g. luminosity distance etc.)
- astCoords (coordinate conversions etc.)
- astImages (clip sections from .fits etc.)
- astPlots (provides a flexible image plot class, e.g. plot image with catalogue objects overlaid)
- astSED (calculate colours, magnitudes from stellar population models or spectral templates, fit photometric observations using stellar population models etc.)
- astStats (statistics, e.g. biweight location/scale estimators etc.)
- astWCS (routines for using FITS World Coordinate System information)

The astWCS module is a higher level interface to PyWCSTools, a simple SWIG (http://www.swig.org) wrapping of some of the routines from WCSTools by Jessica Mink (http://tdc-www.harvard.edu/software/wcstools/). It is used by some routines in astCoords, astImages and astPlots.

The goal of astLib was to provide features useful to astronomers that are not included in the scipy (http://scipy.org), numpy (http://numpy.scipy.org) or matplotlib (http://matplotlib.sourceforge.net) modules on which astLib depends. For a far more extensive set of Python astronomy modules, see astropy (http://www.astropy.org/).

Some scripts using astLib can be found in the examples/ folder provided with the source code distribution.

1.1 Software needed

astLib requires:

- Python (tested on versions 3.6+)
- Astropy - http://www.astropy.org (tested on version 3.2.1)
- Numpy - http://numpy.scipy.org (tested on version 1.18.1)
- SciPy - http://scipy.org (tested on version 1.3.1)
- Matplotlib - http://matplotlib.sourceforge.net (tested on version 3.1.1)

From astLib 0.10.0, pyfits is no longer supported, as STScI have depreciated it. If you still require pyfits instead of Astropy, please continue to use astLib 0.9.3.

Optional:

- Python Imaging Library - http://www.pythonware.com/products/pil (tested on version 1.1.7)

Other versions of the software listed above are likely to work.

1.2 Installation

You can install astLib via pip:

```
pip install pip
```

You may also install using the standard `setup.py` script, e.g., as root:

```
sudo python setup.py install
```

Alternatively,

```
python setup.py install --user
```

will install `astLib` under `$HOME/.local` (on Ubuntu), and in some other default location on Mac.

You can also use the `--prefix` option, e.g.,

```
python setup.py install --prefix=$HOME/local
```

and then add, e.g., `$HOME/local/lib/python3.6/site-packages` to $PYTHONPATH (adjust the path according to your Python version number).

```
export PYTHONPATH=$HOME/local/lib/python3.6/site-packages:$PYTHONPATH
```

1.3 Installation on recent versions of macOS

Some users have reported that the standard method for installing `astLib` does not work on recent versions of macOS (e.g., Big Sur), due to the default compiler flags. The current workaround for this is to install using:

```
CFLAGS="-Wno-error=implicit-function-declaration" python setup.py install
```

Thanks to Michael Cowley and Stefano Covino for helping to resolve this issue.

1.4 Usage

To access the routines in the astLib modules, simply:

```
from astLib import astCalc
from astLib import astCoords
from astLib import astWCS
```

etc.

The astWCS module currently provides access to what are (I think) the most commonly needed WCS information and functions (such as converting between pixel and WCS coordinates etc.). However, should you wish to access the wrapped WCSTools routines themselves directly:

```
from PyWCSTools import wcs
from PyWCSTools import wcscon
```

etc.

Note that PyWCSTools only includes some functions from wcs.c and wcscon.c at present. For examples of usage, look at the Python code for the astLib.astWCS module. Documentation for the WCSTools routines can be found here: http://tdc-www.harvard.edu/software/wcstools/subroutines/libwcs.wcs.html.

As of version 0.11.x+, by default the `astWCS.WCS` class is using the `astropy.wcs` module instead of PyWCSTools (this allows one to benefit from some features of `astropy.wcs` without having to re-write code based on `astWCS.WCS`). To use PyWCSTools instead, set `useAstropyWCS = False` when creating a `WCS` object.

1.5 Known issues

This may no longer apply, but just in case…

Recent versions of matplotlib (on which astLib depends) now use locale information. On systems where the decimal point separator is not '.' (e.g. Germany), the astWCS coordinate conversions routines will give strange results if this is not accounted for. As of version 0.3.0, the astWCS module will detect if this is the case and print a warning message to the console.

The workaround for this issue is to add the following after importing any python modules that expicitly set the locale (such as matplotlib):

```
import locale
locale.setlocale(locale.LC_NUMERIC, 'C')"
```

Thanks to Markus Demleitner for pointing this out.

1.6 Documentation

Documentation is available on the web at http://astlib.readthedocs.io.

1.7 Bugs

Please email bug reports to matt.hilton@mykolab.com, and/or use the GitHub issues page.

Please include details of your operating system, python version, and versions of the python packages required by astLib that you have installed on your machine. For any WCS-related bugs, it would be helpful if you could also include the image header as a text file so that I can reproduce them easily.

CHAPTER

TWO

API REFERENCE

You can find automatically generated documentation for each module in the `astLib` package below.

2.1 astCalc

Module for performing common calculations.

(c) 2007-2011 Matt Hilton

(c) 2013-2014 Matt Hilton & Steven Boada

The focus in this module is at present on calculations of distances in a given cosmology. The parameters for the cosmological model are set using the variables OMEGA_M0, OMEGA_L0, OMEGA_R0, H0 in the module namespace.

`astLib.astCalc.C_LIGHT = 300000.0`
The speed of light in km/s.

`astLib.astCalc.DeltaVz(z)`
Calculates the density contrast of a virialised region S{Delta}V(z), assuming a S{Lambda}CDM-type flat cosmology. See, e.g., Bryan & Norman 1998 (ApJ, 495, 80).

Parameters `z` (*float*) – redshift

Return type float

Returns density contrast of a virialised region at redshift z

Note If OMEGA_M0+OMEGA_L0 is not equal to 1, this routine exits and

prints an error message to the console.

`astLib.astCalc.Ez(z)`
Calculates the value of E(z), which describes evolution of the Hubble parameter with redshift, at redshift z for the current set of cosmological parameters. See, e.g., Bryan & Norman 1998 (ApJ, 495, 80).

Parameters `z` (*float*) – redshift

Return type float

Returns value of E(z) at redshift z

`astLib.astCalc.Ez2(z)`
Calculates the value of E(z)^2, which describes evolution of the Hubble parameter with redshift, at redshift z for the current set of cosmological parameters. See, e.g., Bryan & Norman 1998 (ApJ, 495, 80).

Parameters `z` (*float*) – redshift

Return type float

Returns value of E(z)^2 at redshift z

astLib.astCalc.**H0 = 70.0**
The Hubble parameter (in km/s/Mpc) at z=0.

astLib.astCalc.**OMEGA_L0 = 0.7**
The dark energy density (in the form of a cosmological constant) at z=0.

astLib.astCalc.**OMEGA_M0 = 0.3**
The matter density parameter at z=0.

astLib.astCalc.**OMEGA_R0 = 8.24e-05**
The radiation density at z=0 (note this is only used currently in calculation of *Ez*).

astLib.astCalc.**OmegaLz**(*z*)
Calculates the dark energy density of the universe at redshift z.

Parameters **z** (*float*) – redshift

Return type float

Returns dark energy density of universe at redshift z

astLib.astCalc.**OmegaMz**(*z*)
Calculates the matter density of the universe at redshift z. See, e.g., Bryan & Norman 1998 (ApJ, 495, 80).

Parameters **z** (*float*) – redshift

Return type float

Returns matter density of universe at redshift z

astLib.astCalc.**OmegaRz**(*z*)
Calculates the radiation density of the universe at redshift z.

Parameters **z** (*float*) – redshift

Return type float

Returns radiation density of universe at redshift z

astLib.astCalc.**RVirialXRayCluster**(*kT*, *z*, *betaT*)
Calculates the virial radius (in Mpc) of a galaxy cluster at redshift z with X-ray temperature kT, assuming self-similar evolution and a flat cosmology. See Arnaud et al. 2002 (A&A, 389, 1) and Bryan & Norman 1998 (ApJ, 495, 80). A flat S{Lambda}CDM-type flat cosmology is assumed.

Parameters

- **kT** (*float*) – cluster X-ray temperature in keV
- **z** (*float*) – redshift
- **betaT** (*float*) – the normalisation of the virial relation, for which Evrard et

al. 1996 (ApJ,469, 494) find a value of 1.05 :rtype: float :return: virial radius of cluster in Mpc

Note If OMEGA_M0+OMEGA_L0 is not equal to 1, this routine exits and

prints an error message to the console.

astLib.astCalc.**absMag**(*appMag*, *distMpc*)
Calculates the absolute magnitude of an object at given luminosity distance in Mpc.

Parameters

- **appMag** (*float*) – apparent magnitude of object
- **distMpc** (*float*) – distance to object in Mpc

Return type float

Returns absolute magnitude of object

astLib.astCalc.**dVcdz**(*z*)
Calculates the line of sight comoving volume element per steradian dV/dz at redshift z.

Parameters **z** (*float*) – redshift

Return type float

Returns comoving volume element per steradian

astLib.astCalc.**da**(*z*)
Calculates the angular diameter distance in Mpc at redshift z.

Parameters **z** (*float*) – redshift

Return type float

Returns angular diameter distance in Mpc

astLib.astCalc.**dc**(*z*)
Calculates the line of sight comoving distance in Mpc at redshift z.

Parameters **z** (*float*) – redshift

Return type float

Returns transverse comoving distance (proper motion distance) in Mpc

astLib.astCalc.**dc2z**(*distanceMpc*)
Calculates the redshift z corresponding to the comoving distance given in Mpc.

Parameters **distanceMpc** (*float*) – distance in Mpc

Return type float

Returns redshift

astLib.astCalc.**dl**(*z*)
Calculates the luminosity distance in Mpc at redshift z.

Parameters **z** (*float*) – redshift

Return type float

Returns luminosity distance in Mpc

astLib.astCalc.**dl2z**(*distanceMpc*)
Calculates the redshift z corresponding to the luminosity distance given in Mpc.

Parameters **distanceMpc** (*float*) – distance in Mpc

Return type float

Returns redshift

astLib.astCalc.**dm**(*z*)
Calculates the transverse comoving distance (proper motion distance) in Mpc at redshift z.

Parameters **z** (*float*) – redshift

Return type float

Returns transverse comoving distance (proper motion distance) in Mpc

astLib.astCalc.**t0**()
Calculates the age of the universe in Gyr at z=0 for the current set of cosmological parameters.

Return type float

Returns age of the universe in Gyr at z=0

astLib.astCalc.**tl**(*z*)
Calculates the lookback time in Gyr to redshift z for the current set of cosmological parameters.

Parameters **z** (*float*) – redshift

Return type float

Returns lookback time in Gyr to redshift z

astLib.astCalc.**tl2z**(*tlGyr*)
Calculates the redshift z corresponding to lookback time tlGyr given in Gyr.

Parameters **tlGyr** (*float*) – lookback time in Gyr

Return type float

Returns redshift

Note Raises ValueError if tlGyr is not positive.

astLib.astCalc.**tz**(*z*)
Calculates the age of the universe at redshift z for the current set of cosmological parameters.

Parameters **z** (*float*) – redshift

Return type float

Returns age of the universe in Gyr at redshift z

astLib.astCalc.**tz2z**(*tzGyr*)
Calculates the redshift z corresponding to age of the universe tzGyr given in Gyr.

Parameters **tzGyr** (*float*) – age of the universe in Gyr

Return type float

Returns redshift

Note Raises ValueError if Universe age not positive

2.2 astCoords

Module for coordinate manipulation (conversions, calculations etc.).

(c) 2007-2012 Matt Hilton

(c) 2013-2016 Matt Hilton & Steven Boada

astLib.astCoords.**calcAngSepDeg**(*RADeg1*, *decDeg1*, *RADeg2*, *decDeg2*)
Calculates the angular separation of two positions on the sky (specified in decimal degrees) in decimal degrees. Note that RADeg2, decDeg2 can be numpy arrays.

Parameters

- **RADeg1** (*float*) – R.A. in decimal degrees for position 1
- **decDeg1** (*float*) – dec. in decimal degrees for position 1
- **RADeg2** (*float or numpy array*) – R.A. in decimal degrees for position 2
- **decDeg2** (*float or numpy array*) – dec. in decimal degrees for position 2

Return type float or numpy array, depending upon type of RADeg2, decDeg2

Returns angular separation in decimal degrees

astLib.astCoords.**calcRADecSearchBox**(*RADeg*, *decDeg*, *radiusSkyDeg*)
Calculates minimum and maximum RA, dec coords needed to define a box enclosing a circle of radius radiusSkyDeg around the given RADeg, decDeg coordinates. Useful for freeform queries of e.g. SDSS, UKIDSS etc.. Uses *calcAngSepDeg*, so has the same limitations.

Parameters

- **RADeg** (*float*) – RA coordinate of centre of search region
- **decDeg** (*float*) – dec coordinate of centre of search region
- **radiusSkyDeg** (*float*) – radius in degrees on the sky used to define search region

Return type list

Returns [RAMin, RAMax, decMin, decMax] - coordinates in decimal degrees defining search box

astLib.astCoords.**convertCoords**(*inputSystem*, *outputSystem*, *coordX*, *coordY*, *epoch*)
Converts specified coordinates (given in decimal degrees) between J2000, B1950, and Galactic.

Parameters

- **inputSystem** (*string*) – system of the input coordinates (either "J2000", "B1950" or "GALACTIC")
- **outputSystem** (*string*) – system of the returned coordinates (either "J2000", "B1950" or "GALACTIC")
- **coordX** (*float*) – longitude coordinate in decimal degrees, e.g. R. A.
- **coordY** (*float*) – latitude coordinate in decimal degrees, e.g. dec.
- **epoch** (*float*) – epoch of the input coordinates

Return type list

Returns coordinates in decimal degrees in requested output system

astLib.astCoords.**decimal2dms**(*decDeg*, *delimiter*)
Converts decimal degrees to string in Degrees:Minutes:Seconds format with user specified delimiter.

Parameters

- **decDeg** (*float*) – coordinate in decimal degrees
- **delimiter** (*string*) – delimiter character in returned string

Return type string

Returns coordinate string in D:M:S format

astLib.astCoords.**decimal2hms**(*RADeg*, *delimiter*)
Converts decimal degrees to string in Hours:Minutes:Seconds format with user specified delimiter.

Parameters

- **RADeg** (*float*) – coordinate in decimal degrees
- **delimiter** (*string*) – delimiter character in returned string

Return type string

Returns coordinate string in H:M:S format

astLib.astCoords.**dms2decimal**(*decString*, *delimiter*)
Converts a delimited string of Degrees:Minutes:Seconds format into decimal degrees.

Parameters

- **decString** (*string*) – coordinate string in D:M:S format
- **delimiter** (*string*) – delimiter character in decString

Return type float

Returns coordinate in decimal degrees

astLib.astCoords.**hms2decimal**(*RAString*, *delimiter*)
Converts a delimited string of Hours:Minutes:Seconds format into decimal degrees.

Parameters

- **RAString** (*string*) – coordinate string in H:M:S format
- **delimiter** (*string*) – delimiter character in RAString

Return type float

Returns coordinate in decimal degrees

astLib.astCoords.**shiftRADec**(*ra1*, *dec1*, *deltaRA*, *deltaDec*)
Computes new right ascension and declination shifted from the original by some delta RA and delta DEC. Input position is decimal degrees. Shifts (deltaRA, deltaDec) are arcseconds, and output is decimal degrees. Based on an IDL routine of the same name.

Parameters

- **ra1** (*R.A. in decimal degrees*) – float
- **dec1** (*dec. in decimal degrees*) – float
- **deltaRA** (*shift in R.A. in arcseconds*) – float
- **deltaDec** (*shift in dec. in arcseconds*) – float

Return type float [newRA, newDec]

Returns shifted R.A. and dec.

2.3 astImages

Module for simple .fits image tasks (rotation, clipping out sections, making .pngs etc.).

(c) 2007-2018 Matt Hilton

Some routines in this module will fail if, e.g., asked to clip a section from a .fits image at a position not found within the image (as determined using the WCS). Where this occurs, the function will return None. An error message will be printed to the console when this happens if astImages.REPORT_ERRORS=True (the default). Testing if an astImages function returns None can be used to handle errors in scripts.

astLib.astImages.**clipImageSectionPix**(*imageData*, *XCoord*, *YCoord*, *clipSizePix*)
Clips a square or rectangular section from an image array at the given pixel coordinates.

Parameters

- **imageData** (*np array*) – image data array
- **XCoord** (*float*) – coordinate in pixels

- **YCoord** (*float*) – coordinate in pixels
- **clipSizePix** (*float or list in format [widthPix, heightPix]*) – if float, size of square clipped section in pixels; if list,

size of clipped section in pixels in x, y axes of output image respectively :rtype: np array :return: clipped image section

astLib.astImages.**clipImageSectionWCS**(*imageData*, *imageWCS*, *RADeg*, *decDeg*, *clipSizeDeg*, *returnWCS=True*)

Clips a square or rectangular section from an image array at the given celestial coordinates. An updated WCS for the clipped section is optionally returned, as well as the x, y pixel coordinates in the original image corresponding to the clipped section.

Note that the clip size is specified in degrees on the sky. For projections that have varying real pixel scale across the map (e.g. CEA), use *clipUsingRADecCoords* instead.

Similarly, this routine will not work for a WCS that has polynomial distortion coefficients in the header (e.g., CTYPE1 = 'RA—TAN-SIP' etc.) - again *clipUsingRADecCoords* can be used in such cases.

Parameters

- **imageData** (*np array*) – image data array
- **imageWCS** (astWCS.WCS) – astWCS.WCS object
- **RADeg** (*float*) – coordinate in decimal degrees
- **decDeg** (*float*) – coordinate in decimal degrees
- **clipSizeDeg** (*float or list in format [widthDeg, heightDeg]*) – if float, size of square clipped section in decimal degrees; if list,

size of clipped section in degrees in x, y axes of image respectively :type returnWCS: bool :param returnWCS: if True, return an updated WCS for the clipped section :rtype: dictionary :return: clipped image section (np array), updated astWCS WCS object for clipped image section, and coordinates of clipped section in imageData in format {'data', 'wcs', 'clippedSection'}.

astLib.astImages.**clipRotatedImageSectionWCS**(*imageData*, *imageWCS*, *RADeg*, *decDeg*, *clipSizeDeg*, *returnWCS=True*)

Clips a square or rectangular section from an image array at the given celestial coordinates. The resulting clip is rotated and/or flipped such that North is at the top, and East appears at the left. An updated WCS for the clipped section is also returned. Note that the alignment of the rotated WCS is currently not perfect - however, it is probably good enough in most cases for use with `ImagePlot` for plotting purposes.

Note that the clip size is specified in degrees on the sky. For projections that have varying real pixel scale across the map (e.g. CEA), use *clipUsingRADecCoords* instead.

Similarly, this routine will not work for a WCS that has polynomial distortion coefficients in the header (e.g., CTYPE1 = 'RA—TAN-SIP' etc.) - again *clipUsingRADecCoords* can be used in such cases.

Parameters

- **imageData** (*np array*) – image data array
- **imageWCS** (astWCS.WCS) – astWCS.WCS object
- **RADeg** (*float*) – coordinate in decimal degrees
- **decDeg** (*float*) – coordinate in decimal degrees
- **clipSizeDeg** (*float*) – if float, size of square clipped section in decimal degrees; if list,

size of clipped section in degrees in RA, dec. axes of output rotated image respectively :type returnWCS: bool :param returnWCS: if True, return an updated WCS for the clipped section :rtype: dictionary :return: clipped image section (np array), updated astWCS WCS object for clipped image section, in format {'data', 'wcs'}.

Note Returns 'None' if the requested position is not found within the image. If the image

WCS does not have keywords of the form CD1_1 etc., the output WCS will not be rotated.

astLib.astImages.**clipUsingRADecCoords**(*imageData*, *imageWCS*, *RAMin*, *RAMax*, *decMin*, *decMax*, *returnWCS=True*)

Clips a section from an image array at the pixel coordinates corresponding to the given celestial coordinates.

Parameters

- **imageData** (*np array*) – image data array
- **imageWCS** (astWCS.WCS) – astWCS.WCS object
- **RAMin** (*float*) – minimum RA coordinate in decimal degrees
- **RAMax** (*float*) – maximum RA coordinate in decimal degrees
- **decMin** (*float*) – minimum dec coordinate in decimal degrees
- **decMax** (*float*) – maximum dec coordinate in decimal degrees
- **returnWCS** (*bool*) – if True, return an updated WCS for the clipped section

Return type dictionary

Returns clipped image section (np array), updated astWCS WCS object for

clipped image section, and corresponding pixel coordinates in imageData in format {'data', 'wcs', 'clippedSection'}.

Note Returns 'None' if the requested position is not found within the image.

astLib.astImages.**generateContourOverlay**(*backgroundImageData*, *backgroundImageWCS*, *contourImageData*, *contourImageWCS*, *contourLevels*, *contourSmoothFactor=0*, *highAccuracy=False*)

Rescales an image array to be used as a contour overlay to have the same dimensions as the background image, and generates a set of contour levels. The image array from which the contours are to be generated will be resampled to the same dimensions as the background image data, and can be optionally smoothed using a Gaussian filter. The sigma of the Gaussian filter (contourSmoothFactor) is specified in arcsec.

Parameters

- **backgroundImageData** (*np array*) – background image data array
- **backgroundImageWCS** (astWCS.WCS) – astWCS.WCS object of the background image data array
- **contourImageData** (*np array*) – image data array from which contours are to be generated
- **contourImageWCS** (astWCS.WCS) – astWCS.WCS object corresponding to contourImageData
- **contourLevels** (*list*) – sets the contour levels - available options: - values: contourLevels=[list of values specifying each level] - linear spacing: contourLevels=['linear', min level value, max level value, number of levels] - can use "min", "max" to automatically set min, max levels from image data - log spacing: contourLevels=['log', min level value, max level value, number of levels] - can use "min", "max" to automatically set min, max levels from image data

- **contourSmoothFactor** (*float*) – standard deviation (in arcsec) of Gaussian filter for

pre-smoothing of contour image data (set to 0 for no smoothing) :type highAccuracy: bool :param highAccuracy: if True, sample every corresponding pixel in each image; otherwise, sample

every nth pixel, where n = the ratio of the image scales.

astLib.astImages.**histEq**(*inputArray*, *numBins*)
Performs histogram equalisation of the input np array.

Parameters

- **inputArray** (*np array*) – image data array
- **numBins** (*int*) – number of bins in which to perform the operation (e.g. 1024)

Return type np array

Returns image data array

astLib.astImages.**intensityCutImage**(*imageData*, *cutLevels*)
Creates a matplotlib.pylab plot of an image array with the specified cuts in intensity applied. This routine is used by *saveBitmap* and *saveContourOverlayBitmap*, which both produce output as .png, .jpg, etc. images.

Parameters

- **imageData** (*np array*) – image data array
- **cutLevels** (*list*) – sets the image scaling - available options: - pixel values: cutLevels=[low value, high value]. - histogram equalisation: cutLevels=["histEq", number of bins (e.g. 1024)] - relative: cutLevels=["relative", cut per cent level (e.g. 99.5)] - smart: cutLevels=["smart", cut per cent level (e.g. 99.5)]

["smart", 99.5] seems to provide good scaling over a range of different images. :rtype: dictionary :return: image section (np.array), matplotlib image normalisation (matplotlib.colors.Normalize), in the format {'image', 'norm'}.

Note If cutLevels[0] == "histEq", then only {'image'} is returned.

astLib.astImages.**normalise**(*inputArray*, *clipMinMax*)
Clips the inputArray in intensity and normalises the array such that minimum and maximum values are 0, 1. Clip in intensity is specified by clipMinMax, a list in the format [clipMin, clipMax]

Used for normalising image arrays so that they can be turned into RGB arrays that matplotlib can plot (see astPlots.ImagePlot).

Parameters

- **inputArray** (*np array*) – image data array
- **clipMinMax** (*list*) – [minimum value of clipped array, maximum value of clipped array]

Return type np array

Returns normalised array with minimum value 0, maximum value 1

astLib.astImages.**resampleToTanProjection**(*imageData*, *imageWCS*, *outputPixDimensions=[600, 600]*)
Resamples an image and WCS to a tangent plane projection. Purely for plotting purposes (e.g., ensuring RA, dec. coordinate axes perpendicular).

Parameters

- **imageData** (*np array*) – image data array
- **imageWCS** (*astWCS.WCS*) – astWCS.WCS object
- **outputPixDimensions** (*list*) – [width, height] of output image in pixels

Return type dictionary

Returns image data (np array), updated astWCS WCS object for image, in format {'data', 'wcs'}.

astLib.astImages.**resampleToWCS**(*im1Data*, *im1WCS*, *im2Data*, *im2WCS*, *highAccuracy=False*, *onlyOverlapping=True*)

Resamples data corresponding to second image (with data im2Data, WCS im2WCS) onto the WCS of the first image (im1Data, im1WCS). The output, resampled image is of the pixel same dimensions of the first image. This routine is for assisting in plotting - performing photometry on the output is not recommended.

Set highAccuracy == True to sample every corresponding pixel in each image; otherwise only every nth pixel (where n is the ratio of the image scales) will be sampled, with values in between being set using a linear interpolation (much faster).

Set onlyOverlapping == True to speed up resampling by only resampling the overlapping area defined by both image WCSs.

Parameters

- **im1Data** (*np array*) – image data array for first image
- **im1WCS** (astWCS.WCS) – astWCS.WCS object corresponding to im1Data
- **im2Data** (*np array*) – image data array for second image (to be resampled to match first image)
- **im2WCS** (astWCS.WCS) – astWCS.WCS object corresponding to im2Data
- **highAccuracy** (*bool*) – if True, sample every corresponding pixel in each image; otherwise, sample every nth pixel, where n = the ratio of the image scales.
- **onlyOverlapping** (*bool*) – if True, only consider the overlapping area defined by both image WCSs (speeds things up)

Return type dictionary

Returns np image data array and associated WCS in format {'data', 'wcs'}

astLib.astImages.**saveBitmap**(*outputFileName*, *imageData*, *cutLevels*, *size*, *colorMapName*)

Makes a bitmap image from an image array; the image format is specified by the filename extension. (e.g. ".jpg" =JPEG, ".png"=PNG).

Parameters

- **outputFileName** (*string*) – filename of output bitmap image
- **imageData** (*np array*) – image data array
- **cutLevels** (*list*) – sets the image scaling - available options: - pixel values: cutLevels=[low value, high value]. - histogram equalisation: cutLevels=["histEq", number of bins (e.g. 1024)] - relative: cutLevels=["relative", cut per cent level (e.g. 99.5)] - smart: cutLevels=["smart", cut per cent level (e.g. 99.5)]

["smart", 99.5] seems to provide good scaling over a range of different images. :type size: int :param size: size of output image in pixels :type colorMapName: string :param colorMapName: name of a standard matplotlib colormap, e.g. "hot", "cool", "gray" etc. (do "help(pylab.colormaps)" in the Python interpreter to see available options)

astLib.astImages.**saveContourOverlayBitmap**(*outputFileName*, *backgroundImageData*, *backgroundImageWCS*, *cutLevels*, *size*, *colorMapName*, *contourImageData*, *contourImageWCS*, *contourSmoothFactor*, *contourLevels*, *contourColor*, *contourWidth*)

Makes a bitmap image from an image array, with a set of contours generated from a second image array overlaid.

The image format is specified by the file extension (e.g. ".jpg"=JPEG, ".png"=PNG). The image array from which the contours are to be generated can optionally be pre-smoothed using a Gaussian filter.

Parameters

- **outputFileName** (*string*) – filename of output bitmap image
- **backgroundImageData** (*np array*) – background image data array
- **backgroundImageWCS** (astWCS.WCS) – astWCS.WCS object of the background image data array
- **cutLevels** (*list*) – sets the image scaling - available options: - pixel values: cutLevels=[low value, high value]. - histogram equalisation: cutLevels=["histEq", number of bins (e.g. 1024)] - relative: cutLevels=["relative", cut per cent level (e.g. 99.5)] - smart: cutLevels=["smart", cut per cent level (e.g. 99.5)]

["smart", 99.5] seems to provide good scaling over a range of different images. :type size: int :param size: size of output image in pixels :type colorMapName: string :param colorMapName: name of a standard matplotlib colormap, e.g. "hot", "cool", "gray" etc. (do "help(pylab.colormaps)" in the Python interpreter to see available options) :type contourImageData: np array :param contourImageData: image data array from which contours are to be generated :type contourImageWCS: astWCS.WCS :param contourImageWCS: astWCS.WCS object corresponding to contourImageData :type contourSmoothFactor: float :param contourSmoothFactor: standard deviation (in pixels) of Gaussian filter for pre-smoothing of contour image data (set to 0 for no smoothing) :type contourLevels: list :param contourLevels: sets the contour levels - available options:

- values: contourLevels=[list of values specifying each level]
- linear spacing: contourLevels=['linear', min level value, max level value, number

of levels] - can use "min", "max" to automatically set min, max levels from image data - log spacing: contourLevels=['log', min level value, max level value, number of levels] - can use "min", "max" to automatically set min, max levels from image data

Parameters **contourColor** (*string*) – color of the overlaid contours, specified by the name of a standard

matplotlib color, e.g., "black", "white", "cyan" etc. (do "help(pylab.colors)" in the Python interpreter to see available options) :type contourWidth: int :param contourWidth: width of the overlaid contours

astLib.astImages.**saveFITS**(*outputFileName*, *imageData*, *imageWCS=None*)

Writes an image array to a new .fits file.

Parameters

- **outputFileName** (*string*) – filename of output FITS image
- **imageData** (*np array*) – image data array
- **imageWCS** (*astWCS.WCS object*) – image WCS object

Note If imageWCS=None, the FITS image will be written with a rudimentary header containing

no meta data.

astLib.astImages.**scaleImage**(*imageData*, *imageWCS*, *scaleFactor*)

Scales image array and WCS by the given scale factor.

Parameters

- **imageData** (*np array*) – image data array
- **imageWCS** (astWCS.WCS) – astWCS.WCS object

- **scaleFactor** (*float or list or tuple*) – factor to resize image by - if tuple or list, in format [x scale factor, y scale factor]

Return type dictionary

Returns image data (np array), updated astWCS WCS object for image, in format {'data', 'wcs'}.

2.4 astPlots

Module for producing astronomical plots.

(c) 2007-2018 Matt Hilton

This module provides the matplotlib powered ImagePlot class, which is designed to be flexible. ImagePlots can have RA, Dec. coordinate axes, contour overlays, and have objects marked in them, using WCS coordinates. RGB plots are supported too.

astLib.astPlots.**DECIMAL_TICK_STEPS = [0.001, 0.0025, 0.005, 0.01, 0.025, 0.05, 0.1, 0.25, 0.5, 1.0, 2.0, 2.5, 5.0, 10.0, 30.0, 90.0]**

Defines the possible coordinate label steps on both coordinate axes in decimal degrees mode.

astLib.astPlots.**DEC_TICK_STEPS = [{'deg': 0.0002777777777777778, 'unit': 's'}, {'deg': 0.0005555555555555556, 'unit': 's'}, {'deg': 0.001388888888888887, 'unit': 's'}, {'deg': 0.002777777777777775, 'unit': 's'}, {'deg': 0.00833333333333333, 'unit': 's'}, {'deg': 0.016666666666666666, 'unit': 'm'}, {'deg': 0.03333333333333333, 'unit': 'm'}, {'deg': 0.08333333333333333, 'unit': 'm'}, {'deg': 0.25, 'unit': 'm'}, {'deg': 0.5, 'unit': 'm'}, {'deg': 1.0, 'unit': 'd'}, {'deg': 2.0, 'unit': 'd'}, {'deg': 4.0, 'unit': 'd'}, {'deg': 5.0, 'unit': 'd'}, {'deg': 10.0, 'unit': 'd'}, {'deg': 20.0, 'unit': 'd'}, {'deg': 30.0, 'unit': 'd'}]**

Defines the possible coordinate label steps on the delination axis in sexagesimal mode. Dictionary format: {'deg', 'unit'}

class astLib.astPlots.**ImagePlot**(*imageData*, *imageWCS*, *axes=[0.1, 0.1, 0.8, 0.8]*, *cutLevels=['smart', 99.5]*, *colorMapName='gray'*, *title=None*, *axesLabels='sexagesimal'*, *axesFontFamily='serif'*, *axesFontSize=12.0*, *RATickSteps='auto'*, *decTickSteps='auto'*, *colorBar=False*, *interpolation='bilinear'*)

This class describes a matplotlib image plot containing an astronomical image with an associated WCS.

Objects within the image boundaries can be marked by passing their WCS coordinates to *ImagePlot.addPlotObjects*.

Other images can be overlaid using *ImagePlot.addContourOverlay*.

For images rotated with North at the top, East at the left (as can be done using `astImages.clipRotatedImageSectionWCS` or `astImages.resampleToTanProjection`, WCS coordinate axes can be plotted, with tick marks set appropriately for the image size. Otherwise, a compass can be plotted showing the directions of North and East in the image.

RGB images are also supported.

The plot can of course be tweaked further after creation using matplotlib/pylab commands.

addCompass(*location*, *sizeArcSec*, *color='white'*, *fontSize=12*, *width=20.0*)

Adds a compass to the ImagePlot at the given location ('N', 'NE', 'E', 'SE', 'S', 'SW', 'W', or 'NW'). Note these aren't directions on the WCS coordinate grid, they are relative positions on the plot - so N is top centre, NE is top right, SW is bottom right etc.. Alternatively, pixel coordinates (x, y) in the image can be given.

Parameters

- **location** (*string or tuple*) – location in the plot where the compass is drawn: - string: N, NE, E, SE, S, SW, W or NW - tuple: (x, y)
- **sizeArcSec** (*float*) – length of the compass arrows on the plot in arc seconds
- **color** (*string*) – any valid matplotlib color string
- **fontSize** (*float*) – size of font used to label N and E, in points
- **width** (*float*) – width of arrows used to mark compass

addContourOverlay(*contourImageData*, *contourWCS*, *tag*, *levels=['linear', 'min', 'max', 5]*, *width=1*, *color='white'*, *smooth=0*, *highAccuracy=False*)

Adds image data to the ImagePlot as a contour overlay. The contours can be removed using *removeContourOverlay*. If a contour overlay already exists with this tag, it will be replaced.

Parameters

- **contourImageData** (*numpy array*) – image data array from which contours are to be generated
- **contourWCS** (astWCS.WCS) – astWCS.WCS object for the image to be contoured
- **tag** (*string*) – identifying tag for this set of contours
- **levels** (*list*) – sets the contour levels - available options: - values: contourLevels=[list of values specifying each level] - linear spacing: contourLevels=['linear', min level value, max level value, number of levels] - can use "min", "max" to automatically set min, max levels from image data - log spacing: contourLevels=['log', min level value, max level value, number of levels] - can use "min", "max" to automatically set min, max levels from image data
- **width** (*int*) – width of the overlaid contours
- **color** (*string*) – color of the overlaid contours, specified by the name of a standard matplotlib color, e.g., "black", "white", "cyan" etc. (do "help(pylab.colors)" in the Python interpreter to see available options)
- **smooth** (*float*) – standard deviation (in arcsec) of Gaussian filter for pre-smoothing of contour image data (set to 0 for no smoothing)
- **highAccuracy** (*bool*) – if True, sample every corresponding pixel in each image; otherwise, sample every nth pixel, where n = the ratio of the image scales.

addPlotObjects(*objRAs*, *objDecs*, *tag*, *symbol='circle'*, *size=4.0*, *width=1.0*, *color='yellow'*, *objLabels=None*, *objLabelSize=12.0*)

Add objects with RA, dec coords objRAs, objDecs to the ImagePlot. Only objects that fall within the image boundaries will be plotted.

symbol specifies the type of symbol with which to mark the object in the image. The following values are allowed:

- "circle"
- "box"
- "cross"
- "diamond"

size specifies the diameter in arcsec of the symbol (if plotSymbol == "circle"), or the width of the box in arcsec (if plotSymbol == "box")

width specifies the thickness of the symbol lines in pixels

color can be any valid matplotlib color (e.g. “red”, “green”, etc.)

The objects can be removed from the plot by using removePlotObjects(), and then calling draw(). If the ImagePlot already has a set of plotObjects with the same tag, they will be replaced.

Parameters

- **objRAs** (*numpy array or list*) – object RA coords in decimal degrees
- **objDecs** (*numpy array or list*) – corresponding object Dec. coords in decimal degrees
- **tag** (*string*) – identifying tag for this set of objects
- **symbol** (*string*) – either “circle”, “box”, “cross”, or “diamond”
- **size** (*float*) – size of symbols to plot (radius in arcsec, or width of box)
- **width** (*float*) – width of symbols in pixels
- **color** (*string*) – any valid matplotlib color string, e.g. “red”, “green” etc.
- **objLabels** (*list*) – text labels to plot next to objects in figure
- **objLabelSize** (*float*) – size of font used for object labels (in points)

addScaleBar(*location*, *sizeArcSec*, *color='white'*, *fontSize=12*, *width=20.0*, *label=None*, *style='whiskers'*)

Adds a scale bar to the ImagePlot at the given location (‘N’, ‘NE’, ‘E’, ‘SE’, ‘S’, ‘SW’, ‘W’, or ‘NW’). Note these aren’t directions on the WCS coordinate grid, they are relative positions on the plot - so N is top centre, NE is top right, SW is bottom right etc.. Alternatively, pixel coordinates (x, y) in the image can be given.

Parameters

- **location** (*string or tuple*) – location in the plot where the compass is drawn: - string: N, NE, E, SE, S, SW, W or NW - tuple: (x, y)
- **sizeArcSec** (*float*) – scale length to indicate on the plot in arc seconds
- **color** (*string*) – any valid matplotlib color string
- **fontSize** (*float*) – size of font used to label N and E, in points
- **width** (*float*) – width of arrow used to mark scale
- **label** (*string*) – overrides the displayed label if not None (if None, label is the angular size)
- **style** (*string*) – either “whiskers” or “arrows”

calcWCSAxisLabels(*axesLabels='decimal'*)

This function calculates the positions of coordinate labels for the RA and Dec axes of the ImagePlot. The tick steps are calculated automatically unless self.RATickSteps, self.decTickSteps are set to values other than “auto” (see `ImagePlot.__init__`).

The ImagePlot must be redrawn for changes to be applied.

Parameters **axesLabels** (*string*) – either “sexagesimal” (for H:M:S, D:M:S), “decimal” (for decimal degrees),

or None for no coordinate axes labels

draw()

Redraws the ImagePlot.

getTickSteps()
Chooses the appropriate WCS coordinate tick steps for the plot based on its size. Whether the ticks are decimal or sexagesimal is set by self.axesLabels.

Note: minor ticks not used at the moment.

Return type dictionary

Returns tick step sizes for major, minor plot ticks, in format {'major', 'minor'}

removeContourOverlay(*tag*)
Removes the contourOverlay from the ImagePlot corresponding to the tag.

Parameters **tag** (*string*) – tag for contour overlay in ImagePlot.contourOverlays to be removed

removePlotObjects(*tag*)
Removes the plotObjects from the ImagePlot corresponding to the tag. The plot must be redrawn for the change to take effect.

Parameters **tag** (*string*) – tag for set of objects in ImagePlot.plotObjects to be removed

save(*fileName*)
Saves the ImagePlot in any format that matplotlib can understand, as determined from the fileName extension.

Parameters **fileName** (*string*) – path where plot will be written

astLib.astPlots.**RA_TICK_STEPS = [{'deg': 0.0020833333333333333, 'unit': 's'}, {'deg': 0.004166666666666667, 'unit': 's'}, {'deg': 0.008333333333333333, 'unit': 's'}, {'deg': 0.016666666666666666, 'unit': 's'}, {'deg': 0.020833333333333332, 'unit': 's'}, {'deg': 0.041666666666666664, 'unit': 's'}, {'deg': 0.08333333333333333, 'unit': 's'}, {'deg': 0.125, 'unit': 's'}, {'deg': 0.25, 'unit': 'm'}, {'deg': 0.5, 'unit': 'm'}, {'deg': 1.25, 'unit': 'm'}, {'deg': 2.5, 'unit': 'm'}, {'deg': 5.0, 'unit': 'm'}, {'deg': 7.5, 'unit': 'm'}, {'deg': 15.0, 'unit': 'h'}, {'deg': 45.0, 'unit': 'h'}, {'deg': 90.0, 'unit': 'h'}, {'deg': 180.0, 'unit': 'h'}]**
Defines the possible coordinate label steps on the right ascension axis in sexagesimal mode. Dictionary format: {'deg', 'unit'}

2.5 astWCS

Module for handling World Coordinate Systems (WCS).

(c) 2007-2012 Matt Hilton

(c) 2013-2020 Matt Hilton & Steven Boada

This is a higher level interface to some of the routines in PyWCSTools (distributed with astLib). PyWCSTools is a simple SWIG wrapping of WCSTools by Jessica Mink (http://tdc-www.harvard.edu/software/wcstools/). It is intended is to make this interface complete enough such that direct use of PyWCSTools is unnecessary.

astLib.astWCS.**NUMPY_MODE = True**
If True (default), pixel coordinates accepted/returned by routines such as `astWCS.WCS.pix2wcs`, `astWCS.WCS.wcs2pix` have (0, 0) as the origin. Set to False to make these routines accept/return pixel coords with (1, 1) as the origin (i.e. to match the FITS convention, default behaviour prior to astLib version 0.3.0).

class astLib.astWCS.**WCS**(*headerSource*, *extensionName=0*, *mode='image'*, *zapKeywords=[]*, *useAstropyWCS=True*, *naxis=2*)
This class provides methods for accessing information from the World Coordinate System (WCS) contained in the header of a FITS image. Conversions between pixel and WCS coordinates can also be performed.

To create a WCS object from a FITS file called “test.fits”, simply:

WCS=astWCS.WCS(“test.fits”)

Likewise, to create a WCS object from the pyfits.header of “test.fits”:

img=pyfits.open(“test.fits”) header=img[0].header WCS=astWCS.WCS(header, mode = “pyfits”)

`coordsAreInImage`(*RADeg*, *decDeg*)
Returns True if the given RA, dec coordinate is within the image boundaries.

Return type bool

Returns True if coordinate within image, False if not.

`copy`()
Copies the WCS object to a new object.

Return type astWCS.WCS object

Returns WCS object

`getCentreWCSCoords`()
Returns the RA and dec coordinates (in decimal degrees) at the centre of the WCS.

Return type list

Returns coordinates in decimal degrees in format [RADeg, decDeg]

`getEpoch`()
Returns the epoch of the WCS.

Return type float

Returns epoch of the WCS

`getEquinox`()
Returns the equinox of the WCS.

Return type float

Returns equinox of the WCS

`getFullSizeSkyDeg`()
Returns the width, height of the image according to the WCS in decimal degrees on the sky (i.e., with the projection taken into account).

Return type list

Returns width and height of image in decimal degrees on the sky in format [width, height]

`getHalfSizeDeg`()
Returns the half-width, half-height of the image according to the WCS in RA and dec degrees.

Return type list

Returns half-width and half-height of image in R.A., dec. decimal degrees in format [half-width, half-height]

`getImageMinMaxWCSCoords`()
Returns the minimum, maximum WCS coords defined by the size of the parent image (as defined by the NAXIS keywords in the image header).

Return type list

Returns [minimum R.A., maximum R.A., minimum Dec., maximum Dec.]

getPixelSizeDeg()
Returns the pixel scale of the WCS. This is the average of the x, y pixel scales.

Return type float

Returns pixel size in decimal degrees

getRotationDeg()
Returns the rotation angle in degrees around the axis, North through East.

Return type float

Returns rotation angle in degrees

getXPixelSizeDeg()
Returns the pixel scale along the x-axis of the WCS in degrees.

Return type float

Returns pixel size in decimal degrees

getYPixelSizeDeg()
Returns the pixel scale along the y-axis of the WCS in degrees.

Return type float

Returns pixel size in decimal degrees

isFlipped()
Returns 1 if image is reflected around axis, otherwise returns 0.

Return type int

Returns 1 if image is flipped, 0 otherwise

pix2wcs(*x*, *y*)
Returns the WCS coordinates corresponding to the input pixel coordinates.

Return type list

Returns WCS coordinates in format [RADeg, decDeg]

updateFromHeader()
Updates the WCS object using information from WCS.header. This routine should be called whenever changes are made to WCS keywords in WCS.header.

wcs2pix(*RADeg*, *decDeg*)
Returns the pixel coordinates corresponding to the input WCS coordinates (given in decimal degrees). RADeg, decDeg can be single floats, or lists or np arrays.

Return type list

Returns pixel coordinates in format [x, y]

astLib.astWCS.**findWCSOverlap**(*wcs1*, *wcs2*)
Finds the minimum, maximum WCS coords that overlap between wcs1 and wcs2. Returns these coordinates, plus the corresponding pixel coordinates for each wcs. Useful for clipping overlapping region between two images.

Return type dictionary

Returns dictionary with keys 'overlapWCS' (min, max RA, dec of overlap between wcs1, wcs2) 'wcs1Pix', 'wcs2Pix' (pixel coords in each input WCS that correspond to 'overlapWCS' coords)

2.6 astSED

Module for performing calculations on Spectral Energy Distributions (SEDs).

(c) 2007-2013 Matt Hilton

This module provides classes for manipulating SEDs, in particular the Bruzual & Charlot 2003, Maraston 2005, and Percival et al 2009 stellar population synthesis models are currently supported. Functions are provided for calculating the evolution of colours and magnitudes in these models with redshift etc., and for fitting broadband photometry using these models.

astLib.astSED.**AB = <astLib.astSED.SED object>**
Flat spectrum SED, used for calculation of magnitudes on the AB system (*SED* object).

class astLib.astSED.**BC03Model**(*fileName*)
This class describes a Bruzual & Charlot 2003 stellar population model, extracted from a GALAXEV .ised file using the galaxevpl program that is included in GALAXEV. The file format is white space delimited, with wavelength in the first column. Subsequent columns contain the model fluxes for SEDs of different ages, as specified when running galaxevpl. The age corresponding to each flux column is taken from the comment line beginning "# Age (yr)", and is converted to Gyr.

For example, to load a tau = 0.1 Gyr burst of star formation, solar metallicity, Salpeter IMF model stored in a file (created by galaxevpl) called "csp_lr_solar_0p1Gyr.136":

bc03model = BC03Model("csp_lr_solar_0p1Gyr.136")

The wavelength units of SEDs from BC03 models are Angstroms. Flux is converted into units of erg/s/Angstrom (the units in the files output by galaxevpl are LSun/Angstrom).

astLib.astSED.**Jy2Mag**(*fluxJy*)
Converts flux density in Jy into AB magnitude

Parameters **fluxJy** (*float*) – flux density in Jy

Return type float

Returns AB magnitude

class astLib.astSED.**M05Model**(*fileName*, *fileType='csp'*)
This class describes a Maraston 2005 stellar population model. To load a composite stellar population model (CSP) for a tau = 0.1 Gyr burst of star formation, solar metallicity, Salpeter IMF:

m05csp = astSED.M05Model(M05_DIR+"/csp_e_0.10_z02_salp.sed_agb")

where M05_DIR is set to point to the directory where the Maraston 2005 models are stored on your system.

The file format of the Maraston 2005 simple stellar poulation (SSP) models is different to the file format used for the CSPs, and this needs to be specified using the fileType parameter. To load a SSP with solar metallicity, red horizontal branch morphology:

m05ssp = astSED.M05Model(M05_DIR+"/sed.ssz002.rhb", fileType = "ssp")

The wavelength units of SEDs from M05 models are Angstroms, with flux in units of erg/s/Angstrom.

class astLib.astSED.**P09Model**(*fileName*)
This class describes a Percival et al 2009 (BaSTI; http://albione.oa-teramo.inaf.it) stellar population model. We assume that the synthetic spectra for each model are unpacked under the directory pointed to by fileName.

The wavelength units of SEDs from P09 models are converted to Angstroms. Flux is converted into units of erg/s/Angstrom (the units in the BaSTI low-res spectra are 4.3607e-33 erg/s/m).

class astLib.astSED.**Passband**(*fileName*, *normalise=True*, *inputUnits='angstroms'*, *wavelengthColumn=0*, *transmissionColumn=1*)

This class describes a filter transmission curve. Passband objects are created by loading data from from text files containing wavelength in angstroms in the first column, relative transmission efficiency in the second column (whitespace delimited). For example, to create a Passband object for the 2MASS J filter:

passband=astSED.Passband("J_2MASS.res")

where "J_2MASS.res" is a file in the current working directory that describes the filter.

Wavelength units can be specified as 'angstroms', 'nanometres' or 'microns'; if either of the latter, they will be converted to angstroms.

asList()

Returns a two dimensional list of [wavelength, transmission], suitable for plotting by gnuplot.

Return type list

Returns list in format [wavelength, transmission]

effectiveWavelength()

Calculates effective wavelength for the passband. This is the same as equation (3) of Carter et al. 2009.

Return type float

Returns effective wavelength of the passband, in Angstroms

plot(*xmin='min'*, *xmax='max'*, *maxTransmission=None*)

Plots the passband, rescaling the maximum of the tranmission curve to maxTransmission if required.

Parameters

- **xmin** (*float or 'min'*) – minimum of the wavelength range of the plot
- **xmax** (*float or 'max'*) – maximum of the wavelength range of the plot
- **maxTransmission** (*float*) – maximum value of rescaled transmission curve

rescale(*maxTransmission*)

Rescales the passband so that maximum value of the transmission is equal to maxTransmission. Useful for plotting.

Parameters **maxTransmission** (*float*) – maximum value of rescaled transmission curve

class astLib.astSED.**SED**(*wavelength=[]*, *flux=[]*, *z=0.0*, *ageGyr=None*, *normalise=False*, *label=None*)

This class describes a Spectral Energy Distribution (SED).

To create a SED object, lists (or numpy arrays) of wavelength and relative flux must be provided. The SED can optionally be redshifted. The wavelength units of SEDs are assumed to be Angstroms - flux calculations using Passband and SED objects specified with different wavelength units will be incorrect.

The *StellarPopulation* class (and derivatives) can be used to extract SEDs for specified ages from e.g. the Bruzual & Charlot 2003 or Maraston 2005 models.

asList()

Returns a two dimensional list of [wavelength, flux], suitable for plotting by gnuplot.

Return type list

Returns list in format [wavelength, flux]

calcColour(*passband1*, *passband2*, *magType='Vega'*)

Calculates the colour passband1-passband2.

Parameters

- **passband1** (*Passband* object) – filter passband through which to calculate the first magnitude
- **passband1** – filter passband through which to calculate the second magnitude
- **magType** (*string*) – either "Vega" or "AB"

Return type float

Returns colour defined by passband1 - passband2 on the specified magnitude system

calcFlux(*passband*)

Calculates flux in the given passband.

Parameters **passband** (*Passband* object) – filter passband through which to calculate the flux from the SED

Return type float

Returns flux

calcMag(*passband*, *addDistanceModulus=True*, *magType='Vega'*)

Calculates magnitude in the given passband. If addDistanceModulus == True, then the distance modulus (5.0*log10*(dl*1e5), where dl is the luminosity distance in Mpc at the redshift of the *SED*) is added.

Parameters

- **passband** (*Passband* object) – filter passband through which to calculate the magnitude from the SED
- **addDistanceModulus** (*bool*) – if True, adds 5.0*log10*(dl*1e5) to the mag returned, where dl is the luminosity distance (Mpc) corresponding to the SED z
- **magType** (*string*) – either "Vega" or "AB"

Return type float

Returns magnitude through the given passband on the specified magnitude system

copy()

Copies the SED, returning a new SED object

Return type *SED* object

Returns SED

extinctionCalzetti(*EBMinusV*)

Applies the Calzetti et al. 2000 (ApJ, 533, 682) extinction law to the SED with the given E(B-V) amount of extinction. R_v' = 4.05 is assumed (see equation (5) of Calzetti et al.).

Parameters **EBMinusV** (*float*) – extinction E(B-V), in magnitudes

getSEDDict(*passbands*)

This is a convenience function for pulling out fluxes from a SED for a given set of passbands in the same format as made by *mags2SEDDict* - designed to make fitting code simpler.

Parameters **passbands** (list of *Passband* objects) – list of passbands through which fluxes will be calculated

integrate(*wavelengthMin='min'*, *wavelengthMax='max'*)

Calculates flux in SED within given wavelength range.

Parameters

- **wavelengthMin** (*float or 'min'*) – minimum of the wavelength range
- **wavelengthMax** (*float or 'max'*) – maximum of the wavelength range

Return type float

Returns relative flux

loadFromFile(*fileName*)
Loads SED from a white space delimited file in the format wavelength, flux. Lines beginning with # are ignored.

Parameters **fileName** (*string*) – path to file containing wavelength, flux data

matchFlux(*matchSED*, *minWavelength*, *maxWavelength*)
Matches the flux in the wavelength range given by minWavelength, maxWavelength to the flux in the same region in matchSED. Useful for plotting purposes.

Parameters

- **matchSED** (*SED* object) – SED to match flux to
- **minWavelength** (*float*) – minimum of range in which to match flux of current SED to matchSED
- **maxWavelength** (*float*) – maximum of range in which to match flux of current SED to matchSED

normalise(*minWavelength='min'*, *maxWavelength='max'*)
Normalises the SED such that the area under the specified wavelength range is equal to 1.

Parameters

- **minWavelength** (*float or 'min'*) – minimum wavelength of range over which to normalise SED
- **maxWavelength** (*float or 'max'*) – maximum wavelength of range over which to normalise SED

normaliseToMag(*ABMag*, *passband*)
Normalises the SED to match the flux equivalent to the given AB magnitude in the given passband.

Parameters

- **ABMag** (*float*) – AB magnitude to which the SED is to be normalised at the given passband
- **passband** (an *Passband* object) – passband at which normalisation to AB magnitude is calculated

plot(*xmin='min'*, *xmax='max'*)
Produces a simple (wavelength, flux) plot of the SED.

Parameters

- **xmin** (*float or 'min'*) – minimum of the wavelength range of the plot
- **xmax** (*float or 'max'*) – maximum of the wavelength range of the plot

redshift(*z*)
Redshifts the SED to redshift z.

Parameters **z** (*float*) – redshift

smooth(*smoothPix*)
Smooths SED.flux with a uniform (boxcar) filter of width smoothPix. Cannot be undone.

Parameters **smoothPix** (*int*) – size of uniform filter applied to SED, in pixels

writeToFile(*fileName*)
Writes SED to a white space delimited file in the format wavelength, flux.

Parameters **fileName** (*string*) – path to file

astLib.astSED.**SOL = <astLib.astSED.SED object>**
The SED of the Sun (*SED* object).

class astLib.astSED.**StellarPopulation**(*fileName*, *ageColumn=0*, *wavelengthColumn=1*, *fluxColumn=2*)
This class describes a stellar population model, either a Simple Stellar Population (SSP) or a Composite Stellar Population (CSP), such as the models of Bruzual & Charlot 2003 or Maraston 2005.

The constructor for this class can be used for generic SSPs or CSPs stored in white space delimited text files, containing columns for age, wavelength, and flux. Columns are counted from 0 . . . n. Lines starting with # are ignored.

The classes *M05Model* (for Maraston 2005 models), *BC03Model* (for Bruzual & Charlot 2003 models), and *P09Model* (for Percival et al. 2009 models) are derived from this class. The only difference between them is the code used to load in the model data.

calcEvolutionCorrection(*zFrom*, *zTo*, *zFormation*, *passband*, *magType='Vega'*)
Calculates the evolution correction in magnitudes in the rest frame through the passband from redshift zFrom to redshift zTo, where the stellarPopulation is assumed to be formed at redshift zFormation.

Parameters

- **zFormation** (*float*) – redshift to evolution correct from
- **zTo** (*float*) – redshift to evolution correct to
- **zFormation** – formation redshift of the StellarPopulation
- **passband** (*Passband* object) – filter passband through which to calculate magnitude
- **magType** (*string*) – either "Vega" or "AB"

Return type float

Returns evolution correction in magnitudes in the rest frame

getColourEvolution(*passband1*, *passband2*, *zFormation*, *zStepSize=0.05*, *magType='Vega'*)
Calculates the evolution of the colour observed through passband1 - passband2 for the StellarPopulation with redshift, from z = 0 to z = zFormation.

Parameters

- **passband1** (*Passband* object) – filter passband through which to calculate the first magnitude
- **passband2** (*Passband* object) – filter passband through which to calculate the second magnitude
- **zFormation** (*float*) – formation redshift of the StellarPopulation
- **zStepSize** (*float*) – size of interval in z at which to calculate model colours
- **magType** (*string*) – either "Vega" or "AB"

Return type dictionary

Returns dictionary of numpy.arrays in format {'z', 'colour'}

getMagEvolution(*passband*, *magNormalisation*, *zNormalisation*, *zFormation*, *zStepSize=0.05*, *onePlusZSteps=False*, *magType='Vega'*)
Calculates the evolution with redshift (from z = 0 to z = zFormation) of apparent magnitude in the observed frame through the passband for the StellarPopulation, normalised to magNormalisation (apparent) at z = zNormalisation.

Parameters

- **passband** (*Passband* object) – filter passband through which to calculate the magnitude
- **magNormalisation** (*float*) – sets the apparent magnitude of the SED at zNormalisation
- **zNormalisation** (*float*) – the redshift at which the magnitude normalisation is carried out
- **zFormation** (*float*) – formation redshift of the StellarPopulation
- **zStepSize** (*float*) – size of interval in z at which to calculate model magnitudes
- **onePlusZSteps** (*bool*) – if True, zSteps are (1+z)*zStepSize, otherwise zSteps are linear
- **magType** (*string*) – either "Vega" or "AB"

Return type dictionary

Returns dictionary of numpy.arrays in format {'z', 'mag'}

getSED(*ageGyr*, *z=0.0*, *normalise=False*, *label=None*)

Extract a SED for given age. Do linear interpolation between models if necessary.

Parameters

- **ageGyr** (*float*) – age of the SED in Gyr
- **z** (*float*) – redshift the SED from z = 0 to z = z
- **normalise** (*bool*) – normalise the SED to have area 1

Return type *SED* object

Returns SED

class astLib.astSED.**TopHatPassband**(*wavelengthMin*, *wavelengthMax*, *normalise=True*)

This class generates a passband with a top hat response between the given wavelengths.

astLib.astSED.**VEGA = <astLib.astSED.VegaSED object>**

The SED of Vega, used for calculation of magnitudes on the Vega system (*SED* object).

class astLib.astSED.**VegaSED**(*normalise=False*)

This class stores the SED of Vega, used for calculation of magnitudes on the Vega system.

The Vega SED used is taken from Bohlin 2007 (http://adsabs.harvard.edu/abs/2007ASPC..364..315B), and is available from the STScI CALSPEC library (http://www.stsci.edu/hst/observatory/cdbs/calspec.html).

astLib.astSED.**fitSEDDict**(*SEDDict*, *modelSEDDictList*)

Fits the given SED dictionary (made using *mags2SEDDict*) with the given list of model SED dictionaries. The latter should be made using *makeModelSEDDictList*, and entries for fluxes should correspond directly between the model and SEDDict.

Returns a dictionary with best fit values.

Parameters

- **SEDDict** (dictionary, in format of *mags2SEDDict*) – dictionary of observed fluxes and uncertainties, in format of *mags2SEDDict*
- **modelSEDDictList** (list of dictionaries, in format of *makeModelSEDDictList*) – list of dictionaries containing fluxes of models to be fitted to the observed fluxes listed in the SEDDict. This should be made using *makeModelSEDDictList*.

Return type dictionary

Returns results of the fitting - keys: - 'minChiSq': minimum chi squared value of best fit - 'chiSqContrib': corresponding contribution at each passband to the minimum chi squared value - 'ageGyr': the age in Gyr of the best fitting model - 'modelFileName': the file name of the stellar

population model corresponding to the best fit - 'modelListIndex': the index of the best fitting model in the input modelSEDDictList - 'norm': the normalisation that the best fit model should be multiplied by to match the SEDDict - 'z': the redshift of the best fit model - 'E(B-V)': the extinction, E(B-V), in magnitudes, of the best fit model

astLib.astSED.**flux2Mag**(*flux*, *fluxErr*, *passband*)

Converts given flux and uncertainty in erg/s/cm^2/Angstrom into AB magnitudes.

Parameters

- **flux** (*float*) – flux in erg/s/cm^2/Angstrom in passband
- **fluxErr** (*float*) – uncertainty in flux in passband, in erg/s/cm^2/Angstrom
- **passband** (*Passband* object) – *Passband* object at which ABMag was measured

Return type list

Returns [ABMag, ABMagError], in AB magnitudes

astLib.astSED.**mag2Flux**(*ABMag*, *ABMagErr*, *passband*)

Converts given AB magnitude and uncertainty into flux, in erg/s/cm^2/Angstrom.

Parameters

- **ABMag** (*float*) – magnitude on AB system in passband
- **ABMagErr** (*float*) – uncertainty in AB magnitude in passband
- **passband** (*Passband* object) – *Passband* object at which ABMag was measured

Return type list

Returns [flux, fluxError], in units of erg/s/cm^2/Angstrom

astLib.astSED.**mag2Jy**(*ABMag*)

Converts an AB magnitude into flux density in Jy

Parameters **ABMag** (*float*) – AB magnitude

Return type float

Returns flux density in Jy

astLib.astSED.**mags2SEDDict**(*ABMags*, *ABMagErrs*, *passbands*)

Takes a set of corresponding AB magnitudes, uncertainties, and passbands, and returns a dictionary with keys 'flux', 'fluxErr', 'wavelength'. Fluxes are in units of erg/s/cm^2/Angstrom, wavelength in Angstroms. These dictionaries are the staple diet of the *fitSEDDict* routine.

Parameters

- **ABMags** (*list or numpy array*) – AB magnitudes, specified in corresponding order to passbands and ABMagErrs
- **ABMagErrs** (*list or numpy array*) – AB magnitude errors, specified in corresponding order to passbands and ABMags
- **passbands** (list of *Passband* objects) – passband objects, specified in corresponding order to ABMags and ABMagErrs

Return type dictionary

Returns dictionary with keys {'flux', 'fluxErr', 'wavelength'}, suitable for input to *fitSEDDict*

astLib.astSED.**makeModelSEDDictList**(*modelList*, *z*, *passbandsList*, *labelsList=[]*, *EBMinusVList=[0.0]*, *forceYoungerThanUniverse=True*)

This routine makes a list of SEDDict dictionaries (see *mags2SEDDict*) for fitting using *fitSEDDict*. This speeds up the fitting as this allows us to calculate model SED magnitudes only once, if all objects to be fitted are at the same redshift. We add some meta data to the modelSEDDicts (e.g. the model file names).

The effect of extinction by dust (assuming the Calzetti et al. 2000 law) can be included by giving a list of E(B-V) values.

If forceYoungerThanUniverse == True, ages which are older than the universe at the given z will not be included.

Parameters

- **modelList** (list of *StellarPopulation* model objects) – list of StellarPopulation models to include
- **z** (*float*) – redshift to apply to all stellar population models in modelList
- **EBMinusVList** (*list*) – list of E(B-V) extinction values to apply to all models, in magnitudes
- **labelsList** (*list*) – optional list used for labelling passbands in output SEDDicts
- **forceYoungerThanUniverse** (*bool*) – if True, do not allow models that exceed the age of the universe at z

Return type list

Returns list of dictionaries containing model fluxes, to be used as input to *fitSEDDict*.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

INDEX

U

V

W