PyCASSO Documentation

Release 0.9.3

André Luiz de Amorim

December 03, 2013

CONTENTS

1 Introduction **1**

2 Getting started **3**
- 2.1 Requirements . . . 3
- 2.2 Stable version . . . 3
- 2.3 Development version . . . 4

3 Creating datasets **5**
- 3.1 STARLIGHT data . . . 5
- 3.2 The PyCASSO import tools . . . 5

4 Usage **7**
- 4.1 Listing synthesis runs . . . 7
- 4.2 Loading galaxy data . . . 7
- 4.3 Displaying property images . . . 8
- 4.4 Radial and azimuthal profiles . . . 10
- 4.5 Filling missing data . . . 19
- 4.6 Working with spectra . . . 21
- 4.7 Integrated properties . . . 22

5 Legacy API **25**
- 5.1 Loading from FITS files . . . 25
- 5.2 Using very old scripts . . . 25
- 5.3 Old API reference . . . 26

6 Galaxy property list **31**
- 6.1 Base . . . 31
- 6.2 qbick planes . . . 31
- 6.3 Population . . . 33
- 6.4 Physical properties . . . 35
- 6.5 Spectra . . . 39
- 6.6 Synthesis fit diagnostics . . . 41

7 Utility functions **43**
- 7.1 Function list . . . 43

8 Full data access API **49**
- 8.1 HDF5 query functions . . . 49
- 8.2 Synthesis data abstraction class . . . 49

Index **79**

CHAPTER

ONE

INTRODUCTION

PyCASSO stands for Python CALIFA Stellar Synthesis Organizer, and its main objective is to help handling stellar synthesis datacubes produced by STARLIGHT[1] from integrated field spectra (IFS) produced by the CALIFA survey[2].

This manual serves two purposes: it is a reference for the data cubes API, and a guide to the usage pattern of PyCASSO. The examples, when possible, contain real world data and perform common tasks when dealing with IFS data.

The CALIFA data cubes were preprocessed using a tool called QBICK to obtain spectra with a minimum signal-to-noise ratio using Voronoi binning. PyCASSO and QBICK may merge in the future.

[1] http://astro.ufsc.br/starlight/
[2] http://califa.caha.es/

CHAPTER

TWO

GETTING STARTED

2.1 Requirements

PyCASSO requires python 2.7. You also need these to use the data access modules.

- numpy/scipy
- pyfits
- h5py

These are probaby available at you python package manager. For example, in Mac OSX using macports, use these commands to install python 2.7 and the dependencies, and then select this as your default python installation.

```
$ sudo port install python27 py27-h5py py27-numpy py27-scipy py27-pyfits
$ sudo port select --set python python27
```

You will also need PySTARLIGHT[1]. The package is in the python package index[2], you may install it using easy_install or pip (*NOTE*: be sure that pip and easy_install are using the same python installation as you use for coding, or PySTARLIGHT will be installed at the wrong directory):

```
$ pip install PySTARLIGHT
```

or

```
$ easy_install PySTARLIGHT
```

If you want to create your own datasets, you'll also need atpy[3]. Refer to the site for installation instructions. In macports, install it using the command:

```
$ sudo port install py27-atpy
```

2.2 Stable version

The stable version of PyCASSO is 0.9.0[4]. Install using these commands:

```
$ tar xvzf PyCASSO-X.X.X.tar.gz
$ cd PyCASSO-X.X.X/
$ python setup.py install
```

[1] https://bitbucket.org/astro_ufsc/pystarlight
[2] https://pypi.python.org/pypi
[3] http://atpy.github.com/
[4] https://bitbucket.org/astro_ufsc/pycasso/get/PyCASSO-0.9.0.tar.gz

2.3 Development version

Development version is hosted in bitbucket. Note that the development version may be broken or have half-implemented features. If you are feeling brave, clone it from the mercurial repository:

```
$ hg clone https://bitbucket.org/astro_ufsc/pycasso
```

To install it, run the commands:

```
$ cd pycasso
$ python setup.py install
```

CHAPTER

THREE

CREATING DATASETS

3.1 STARLIGHT data

The main objective of PyCASSO is to organize the access to the stellar synthesis data provided by STARLIGHT. If you are reading this I assume you have the following files:

- QBICK segmentation FITS file (A.K.A. zone file)
- Input spectra for STARLIGHT
- STARLIGHT output

It is important to have the name of the files following the rules:

- Input spectra must have the format: `KNNNN_MMMM_suffix` where `KNNNN` is the CALIFA number and `MMMM` is the zone ID. The suffix can be anything and will be ignored.
- There can be only one input file starting with each pair `KNNNN_MMMM` in the same directory. Use different directories for different runs. This limitation will be fixed in later releases.
- STARLIGHT output files must have the format: `KNNNN_MMMM_runID`. These files can be gzipped or bzipped, and the `.gz` or `.bz2` suffixes will be ignored.

3.2 The PyCASSO import tools

The import process is similar whether you want to use FITS or HDF5 files. Keep in mind that most of the effort will be towards the HDF5 storage, and the FITS support may be dropped in the future.

The import tools are `h5pycassoImport.py` and `pycassoImport.py` to import data to HDF5 and FITS, respectively. These scripts are located in the `tools` directory. Here is an example of a `h5pycassoImport.py` run:

```
$ h5pycassoImport.py --synthesis-dir=./sl_out/ --spectra-dir=./sl_in/ \
> --base=Bgsd01 --base-description='Granada 01' \
> --qbick-run=q027 --qbick-description='COMBO convex voronoi zones ver. 0.2.7' \
> --run-id=eBR_v20_q027.d13c512.ps3b.k1.mC.CCM.Bgsd01.v01 \
> --run-description='q027/Bgsd01 with default parameters' \
> --zone-file=K0001_eBR_v20_q035.d13c512-planes.fits \
> --database=qalifa-synthesis.h5 K0001
```

The arguments are the following:

- `--synthesis-dir`: Path to the synthesis output directory.
- `--spectra-dir`: Path to the synthesis input directory.
- `--base`: Identifier of the base used in the synthesis.
- `--base-description`: Description of the base used in the synthesis.

- `--qbick-run`: Identifier of the qbick run used in the synthesis.
- `--qbick-description`: Description of the qbick run used in the synthesis.
- `--run-id`: Identifier of the whole synthesis run.
- `--run-description`: Description of the whole synthesis run.
- `--zone-file`: Path to the FITS file containing the qbick planes.
- `--database`: Path to the HDF5 database file.
- `CALIFAID`: Califa name of the galaxy.

It is recommended to use a single database for everything.

The arguments for the FITS version are similar. For a complete list, run `h5pycassoImport.py --help` or `pycassoImport.py --help`.

For more information on the database structure, see PyCASSO data model[1].

[1] https://docs.google.com/document/d/1s3D8Ma-whWlZauMb9cMH7lISRrfpuzTh5Tkhbm-zbbA

CHAPTER

FOUR

USAGE

4.1 Listing synthesis runs

This section only applies to HDF5 datasets. If you are using the FITS format, you have one FITS file per galaxy per run.

Start a python shell. To list the synthesis runs present in the database *qalifa-synthesis.h5*, use the function `listRuns()` (page 49):

```
>>> from pycasso.h5datacube import
>>> listRuns('qalifa-synthesis.h5')
eBR_v20_q027.d13c512.ps3b.k1.mC.CCM.Bgsd01.v01
eBR_v20_q036.d13c512.ps03.k2.mC.CCM.Bgsd61
```

In this case, we have two runs, `'eBR_v20_q027.d13c512.ps3b.k1.mC.CCM.Bgsd01.v01'` and `'eBR_v20_q036.d13c512.ps03.k2.mC.CCM.Bgsd61'`. The other tools for listing the contents of a database are `listBases()` (page 49) and `listQbickRuns()` (page 49).

```
>>> listBases('qalifa-synthesis.h5')
Bgsd01 Bgsd61
>>> listQbickRuns('qalifa-synthesis.h5')
q027 q036
```

Now, we can search the runs that have specific bases or qbick runs using the keyword arguments `baseId` and `qbickRunId`. One can specify any one or both of these filter arguments.

```
>>> listRuns('qalifa-synthesis.h5', baseId='q036', qbickRunId='Bgsd61')
eBR_v20_q036.d13c512.ps03.k2.mC.CCM.Bgsd61
```

4.2 Loading galaxy data

To load the galaxy `'K0001'` (IC5376), in the run `'eBR_v20_q036.d13c512.ps03.k2.mC.CCM.Bgsd61'` from the database, do

```
>>> from pycasso.h5datacube import
>>> K = h5Q3DataCube('qalifa-synthesis.h5', \
'eBR_v20_q036.d13c512.ps03.k2.mC.CCM.Bgsd61', 'K0001')
```

The object `K` is a `h5Q3DataCube` (page 49). which implements `IQ3DataCube` (page 59). The latter defines high level algorithms like zone to spatial conversions and radial profiles, while the former contains the I/O related stuff. If you are using ipython, entering `K` followed by two tab keystrokes, you should see all the properties of the galaxy and all methods available to operate on those.

If you have FITS files with your data, please refer to *Loading from FITS files* (page 25).

4.2.1 Zone smoothing

The data from the galaxies are are binned in spatial regions called voronoi bins. There is a procedure called "dezonification" which converts the data from voronoi bins back to spatial coordinates. When doing so, one can use the surface brightness of the image of the galaxy as a hint to smooth some of the properties.

This is enabled by default. To disable it, set the keyword argument `smooth` to `False` when loading the galaxy.

```
>>> K = h5Q3DataCube('qalifa-synthesis.h5', \
'eBR_v20_q036.d13c512.ps03.k2.mC.CCM.Bgsd61', 'K0001', smooth=False)
```

For more information, see `getDezonificationWeight()` (page 59).

4.3 Displaying property images

In this section we explain how to obtain images of properties of the galaxy. Most of of the properties of interest are already calculated, see the section *Galaxy property list* (page 31).

Here we assume the galaxy is loaded as `K`. Now, all the data is stored as voronoi zones, but we really want to see the spatial distribution of the galaxy properties. `K` has a method for converting data from voronoi zones to images, the method `zoneToYX()` (page 59).

Let's see the property `popx` (page 54). It is a 3-dimensional array, containing the light fraction for every triplet (*age*, *metallicity*, *zone*).

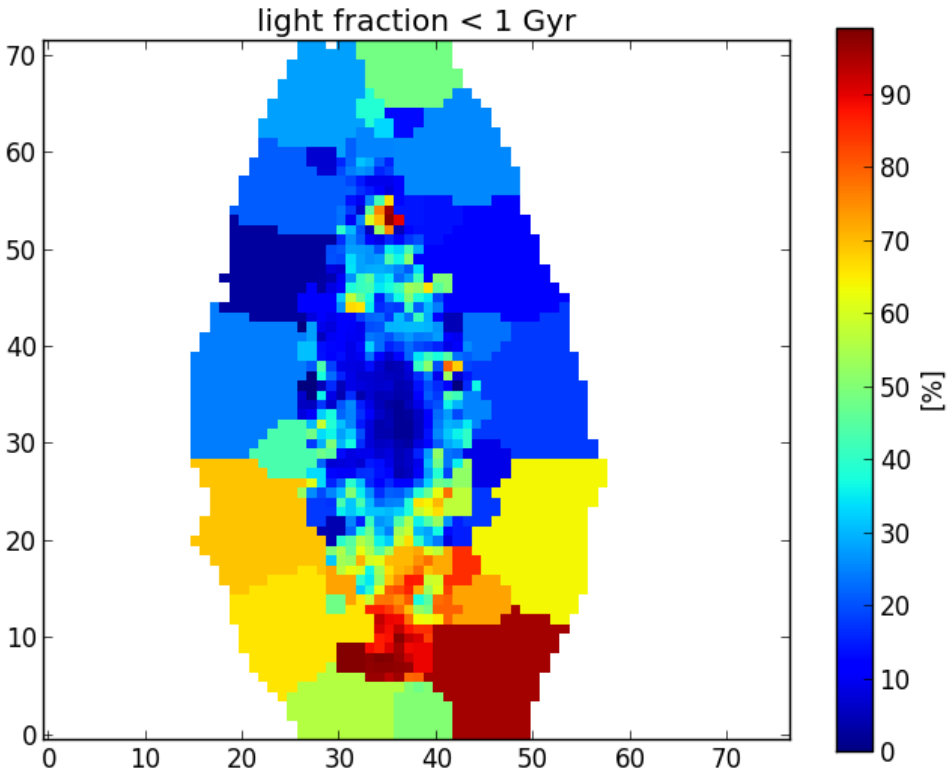
```
>>> K.popx.shape (39, 4, 649) >>> K.N_age, K.N_met, K.N_zone (39, 4, 649)
```

Now, we want to see how this is distributed in the galaxy. The fraction of light in each population is the same in every pixel of a zone. So, we call `popx` an **intensive** property. To inform `zoneToYX()` (page 59) that the property is intensive, we set `extensive=False`.

```
>>> popx = K.zoneToYX(K.popx, extensive=False) >>> popx.shape (39, 4, 72, 77)
```

In this case, the resulting `popx` is 4-dimensional, where the zone dimension has been replaced by the (x,y) plane. Let's plot the young population (summing only the fractions for ages smaller than 1Gyr) as a function of the position in the galaxy plane.

```
>>> import matplotlib.pyplot as plt
>>> popY = popx[K.ageBase < 1e9].sum(axis=1).sum(axis=0)
>>> plt.imshow(popY, origin='lower', interpolation='nearest')
>>> cb = plt.colorbar()
>>> cb.set_label('[%]')
>>> plt.title('light fraction < 1 Gyr')
```



4.3.1 Masked pixels

The white pixels do not belong to any zone (*masked pixels*), and by default are set to `numpy.nan`. One might change the fill value for the masked pixels using the keyword `fill_value`. For example, to fill with zeros,

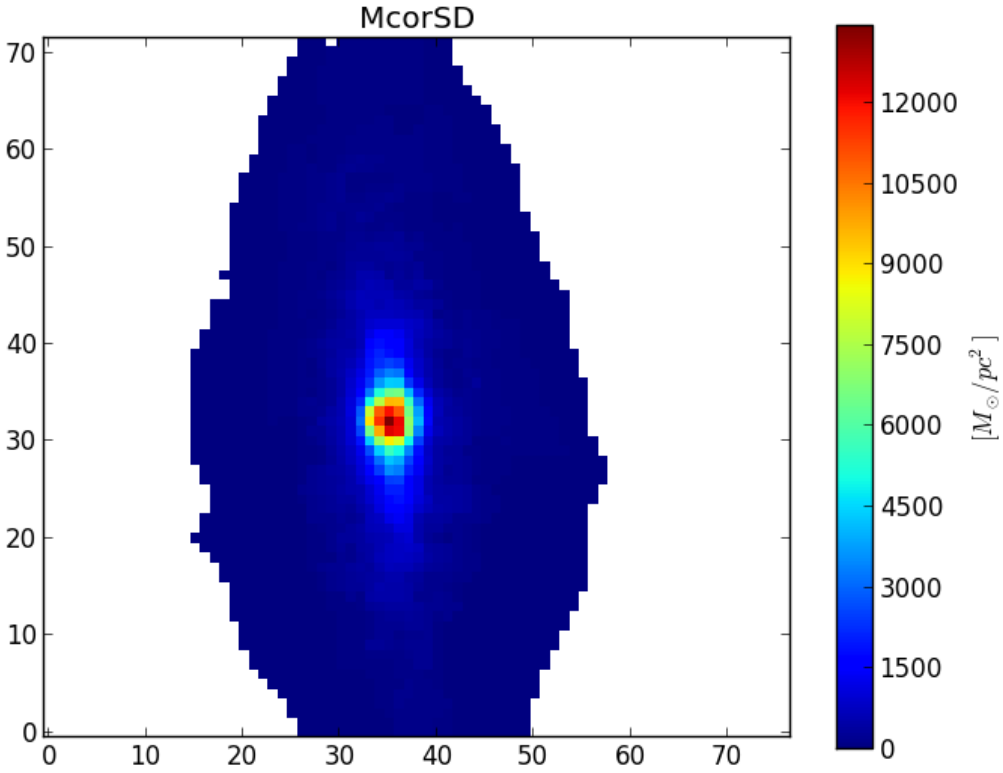
```
>>> popx_fill_zeros = K.zoneToYX(K.popx, extensive=False, fill_value=0.0)
```

It is recommended to use the fill values as `nan`, this way you know when you make a mistake and use values from regions where there's no data (`nan` is contagious, any operation containing a `nan` will yield `nan`).

4.3.2 Surface densities

When the property is extensive, such as mass or flux, it is more proper to work with surface densities of these properties, when working with images. The most easily acknowledged extensive properties are light and mass. The properties `Lobn__z` (page 73), `Mini__z` (page 74) and `Mcor__z` (page 74) are the total light and mass for the voronoi zones. When we convert these to images, setting `extensive=True`, the resulting image contains the surface density of these properties (for example, in units of $M_\odot/pc^2$).

```
>>> McorSD = K.zoneToYX(K.Mcor__z, extensive=True)
>>> plt.imshow(McorSD, origin='lower', interpolation='nearest')
>>> cb = plt.colorbar()
>>> cb.set_label('$[M_\odot / pc^2]$')
>>> plt.title('McorSD')
```



This is the same property as `McorSD__yx` (page 74), see the section below.

4.3.3 Predefined properties

The conversion from zones to images is perhaps the most common thing you wil ever use from PyCASSO. To make things easier (and less verbose), most of the properties already have been converted to images. These predefined properties take care of the extensiveness or intensiveness nature of the physical property, and return a surface density when suitable. The full list of propeties can be seen in the section *Galaxy property list* (page 31).

4.4 Radial and azimuthal profiles

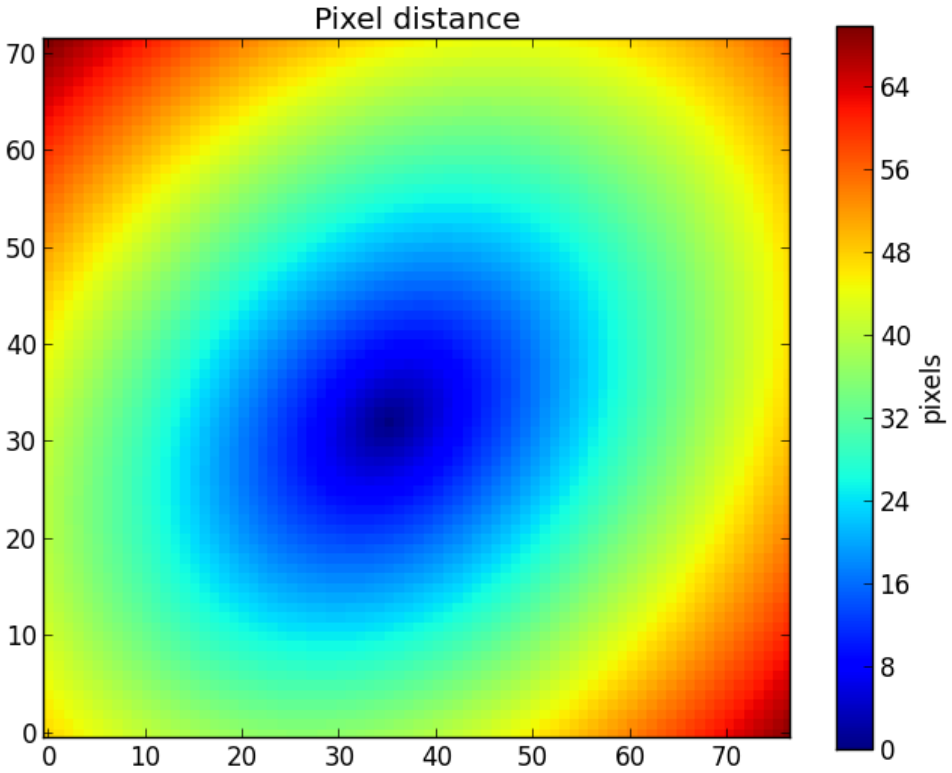
Galaxy images are nice and all, but in the end you want to compare spatial information between galaxies, and it can not be done in a per pixel basis. One can think in polar coordinates and analyze only the radial variations of the properties. For this we have to think about what galaxy distances in these images really mean.

4.4.1 Image geometry

One fair approximation is to assume the galaxy has an axis-symmetric geometry. Therefore, if we see the galaxy as an elliptic shape, that is because of its inclination with respect to the line of sight. If we get all the points at a given distance `a` from the center of the galaxy, this will appear as an ellipse of semimajor axis equal to `a`.

To know the distace of each pixel to the center of the galaxy, we need to know the *ellipticity* (`ba` (page 51)) of the galaxy, defined as $\frac{b}{a}$ (ratio between semiminor and semimajor axes), and the *position angle* (`pa` (page 51), in radians, counter-clockwise from the X-axis) of the ellipse. These can be set using the method `setGeometry()` (page 60). The distance of each pixel is obtained using the property `pixelDistance__yx` (page 51).

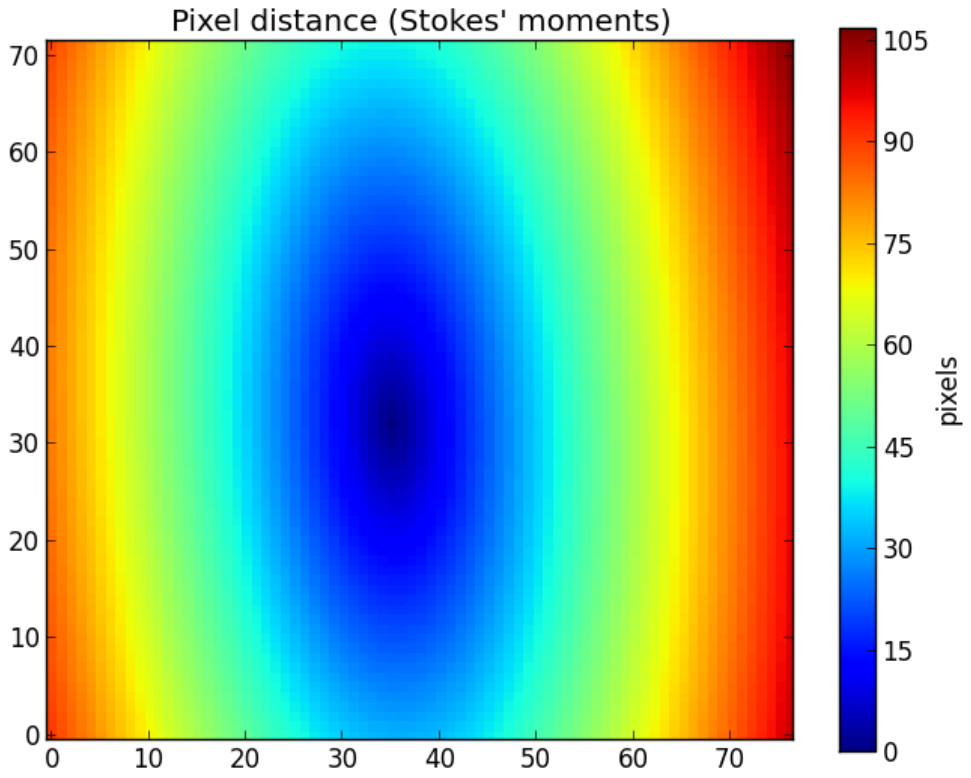
```
>>> K.setGeometry(pa=45*np.pi/180.0, ba=0.75)
>>> dist = K.pixelDistance__yx
>>> plt.imshow(dist, origin='lower', interpolation='nearest')
>>> cb = plt.colorbar()
>>> cb.set_label('pixels')
>>> plt.title('Pixel distance')
```



This example geometry clearly does not correspond to geometry of the galaxy, as seen in the previous figures. If we lack the values of `ba` and `pa` from other sources, we can use the method `getEllipseParams()` (page 68) to estimate these ellipse parameters using "Stokes' moments" of the flux (actually, we use `qSignal` (page 52)) as explained in the paper Sloan Digital Sky Survey: Early Data Release[1].

```
>>> pa, ba = K.getEllipseParams()
>>> K.setGeometry(pa, ba)
```

[1] http://adsabs.harvard.edu/abs/2002AJ....123..485S



Now the distances match the geometry of the galaxy assuming it has an elliptical desitribution. Other parameters thar may be set by `setGeometry()` (page 60) are the *Half Light Radius* (see below) and the center of the galaxy, by using respectively the keyword parameters `HLR_pix` and `center` (a tuple containing x and y).

4.4.2 Galaxy scales

We also need to take into account that galaxies exist in all sizes and distances. This means that using pixels as a distance scale is not satisfactory. Thus we have to convert distances to a "natural scale" to compare the radial profiles of distinct galaxies.

We define the *Half Light Radius* (`HLR_pix` (page 50)) as the semimajor axis of the ellipse centered at the galaxy center containing half of the flux in a certain band. In PyCASSO, we use `qSignal` (page 52), which is the flux in the normalization window (5635).

The HLR is updated whenever `setGeometry()` (page 60) is called. One can explicitly set the HLR using the `HLR_pix` parameter, or let it be determined automatically. Note that the HLR depends heavily on the ellipse parameters.

- Circular

```
>>> K.setGeometry(pa=0.0, ba=1.0)
>>> K.HLR_pix
9.334798311419563
```

- Ellipse

```
>>> pa, ba = K.getEllipseParams()
>>> K.setGeometry(pa, ba)
>>> K.HLR_pix
12.878480006876336
```

```
>>> K.HLR_pc
2545.9657503507078
```

In the last line we show the HLR in parsecs (HLR_pc (page 50)), which may be handy sometimes. It is based on HLR_pix (page 50) and parsecPerPixel (page 50).

Alternatively, one can think in terms of other property as a half-scale. This is achieved using the method getHalfRadius() (page 69). One alternate scale commonly used is the mass. To calculate the Half Mass Radius (HMR) use the following:

```
>>> K.getHalfRadius(K.McorSD__yx)
7.612629351561292
```

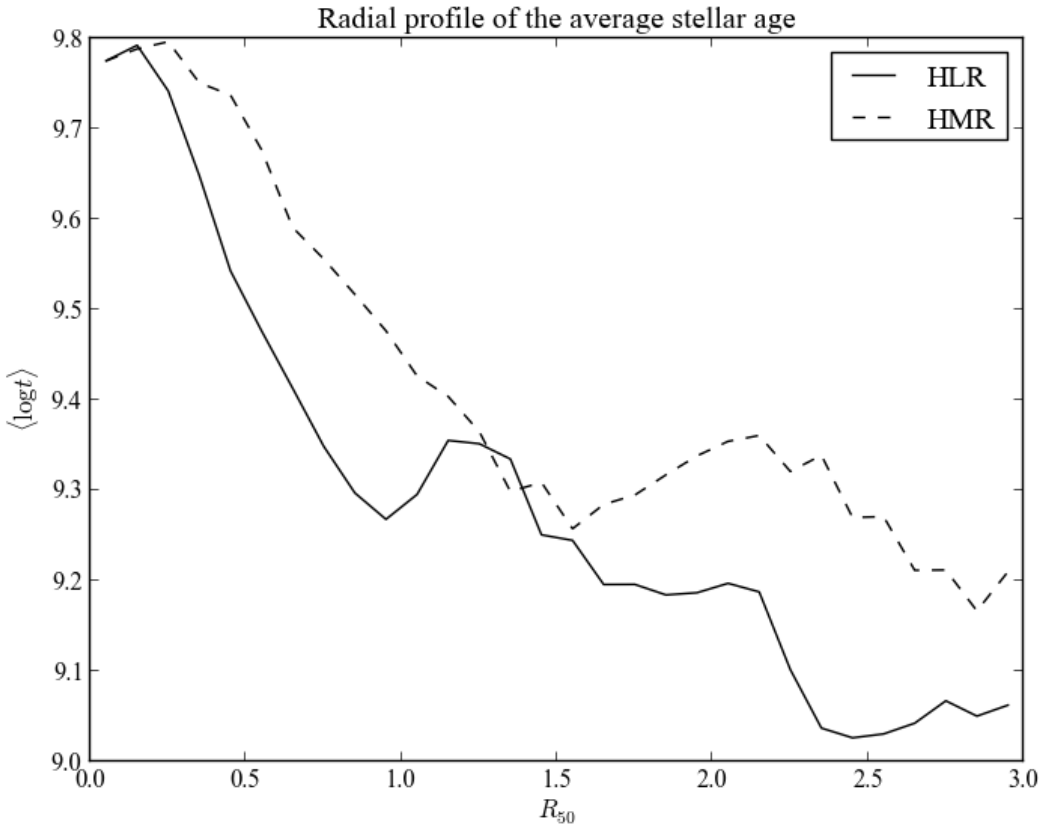
The input for this method must be a 2-D image.

Note: This uses the same computation as the HLR.

4.4.3 Calculating the radial profile of a 2-D image

Now that we have a scale to measure things in radial distance, it is useful to have a radial profile of things. First, let's assume we have a 2-D image we want to get the radial profile, for instance, at_flux__yx (page 75). The method radialProfile() (page 63) receives a 2-D image of a given property, a set of radial bins, an optional radial scale (it uses HLR by default) and returns the average values of the pixels inside each bin. Think of it as a fancy histogram.

```
>>> HMR = K.getHalfRadius(K.McorSD__yx)
>>> bin_r = np.arange(0, 3+0.1, 0.1)
>>> bin_center = (bin_r[1:] + bin_r[:-1]) / 2.0
>>> at_flux__HLR = K.radialProfile(K.at_flux__yx, bin_r)
>>> at_flux__HMR = K.radialProfile(K.at_flux__yx, bin_r, rad_scale=HMR)
>>> plt.plot(bin_center, at_flux__HLR, '-k', label='HLR')
>>> plt.plot(bin_center, at_flux__HMR, '--k', label='HMR')
>>> plt.ylabel(r'$\langle \log t\rangle$')
>>> plt.xlabel(r'$R_{50}$')
>>> plt.title('Radial profile of the mean stellar age')
>>> plt.legend()
```



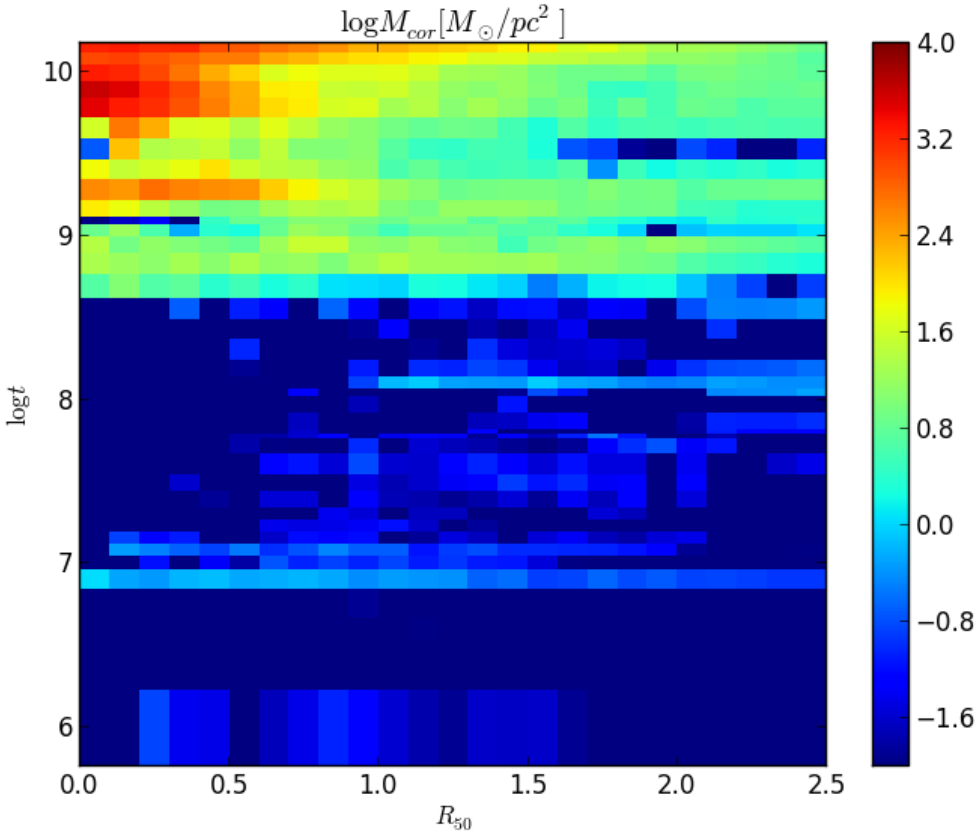
Now let's explain the procedure. `bin_r` represent the bin boundaries in which we want to calculate the mean value of the property. It is not suitable for plotting, as it contains one extra value (`len(bin_r) == len(at_flux__HLR) + 1`). So, we calculate the central points of the bins, `bin_center`. Then, we proceed to calculate the radial profile of `at_flux__yx`, which is the average log. of the age, weighted by flux. We do so using HLR and HMR as a distance scale. The mass is usually more concentrated in the center of the galaxy, so that in general `HMR < K.HLR_pix` and therefore the plots look similar, but stretched.

4.4.4 Calculating the radial profile of a N-D image

In the example above we compute the radial profile of a 2-D image. Now, the process is the same for higher-dimensional images. The presentation can be tricky, though. Let's see an example.

We have the surface distribution of mass in ages, `McorSD__tyx`, which is a 3-D array (age, y, x). One may be tempted to do a `for` loop for each age, but the method `radialProfile()` takes care of this, as it accepts properties with more than 2 dimensions, as long as the spatial dimensions are the rightmost (`[..., y, x]` in numpy slice notation). In this example, we want to plot a 2-D map of the mass as a function of radius and age. We will use the Half Light Radius as a radial scale, so we don't have to set the argument `rad_scale` as we did previously. We also don't need the bin centers, as the plot method used is fit for dealing with histogram-like data.

```
>>> bin_r = np.arange(0, 2.5+0.1, 0.1)
>>> McorSD__tyx = K.McorSD__tZyx.sum(axis=1)
>>> logMcorSD__tr = np.log10(K.radialProfile(McorSD__tyx, bin_r, mode='mean'))
>>> vmin = -2
>>> vmax = 4
>>> plt.pcolormesh(bin_r, K.logAgeBaseBins, logMcorSD__tr, vmin=-2, vmax=4)
>>> plt.colorbar()
>>> plt.ylim(K.logAgeBaseBins.min(), K.logAgeBaseBins.max())
>>> plt.ylabel(r'$\log t$')
>>> plt.xlabel(r'$R_{50}$')
>>> plt.title(r'$\log M_{cor} [M_\odot / pc^2]$')
```



As we said, the :meth:pseudocolor method needs the bin edges, so we have to use the radial bin edges (provided by `bin_r`), and the age bin edges. The property called `logAgeBaseBins` returns the age bin edges in log space, see it's documentation for more details.

One can also compute the sum of the values (or the mean or median) inside a ring, using a pair of values as bins:

```
>>> Mcor_tot_ring = K.radialProfile(K.McorSD__yx, \
(0.5, 1.0), mode='sum') * K.parsecPerPixel**2
```

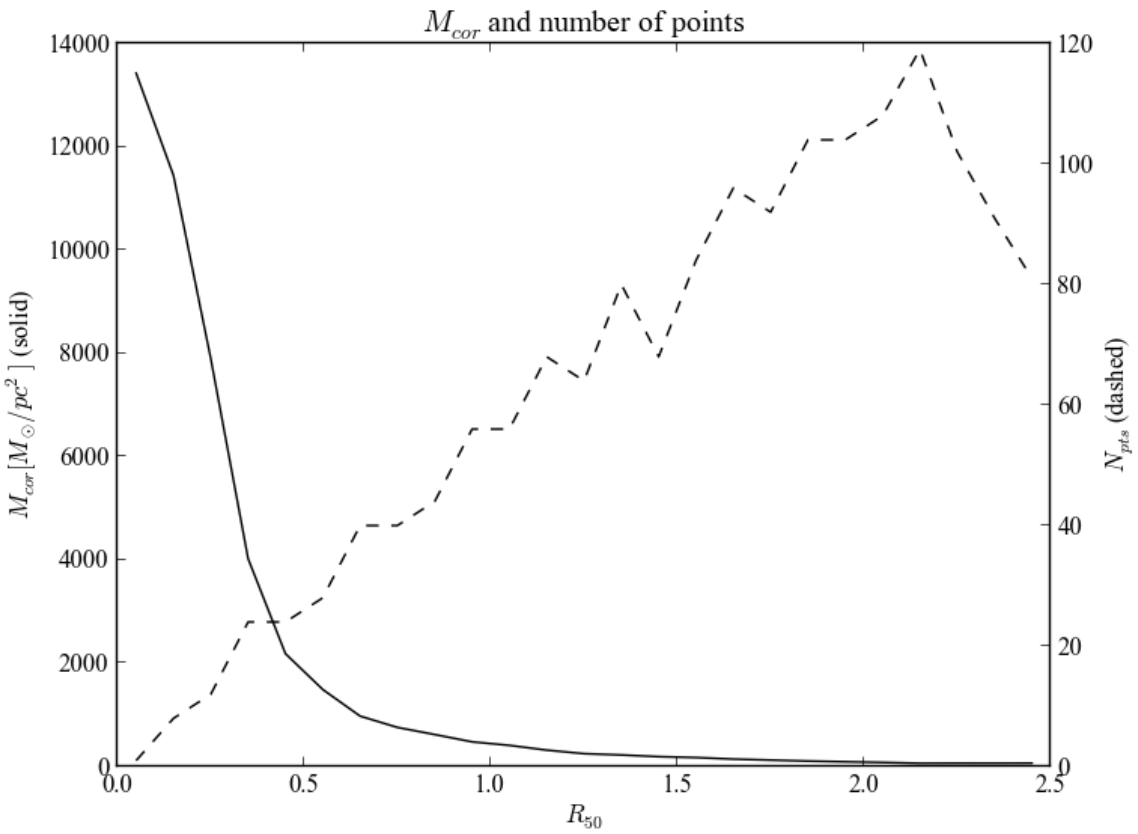
The rightmost term is the pixel area, to convert from surface density to absolute mass. The `mode` keyword indicates what to do inside the bins. By default, the mean is computed. The valid values for `mode` are:

- `'mean'`
- `'mean_exact'` (see below)
- `'sum'`
- `'median'`

If the number of pixels inside each radial bin is needed, the keyword argument `return_npts` may be set. In this case, the return value changes to a tuple containing the radial binnned property and the number of points in each bin.

```
>>> bin_r = np.arange(0, 2.5+0.1, 0.1)
>>> bin_center = (bin_r[1:] + bin_r[:-1]) / 2.0
>>> McorSD__r, npts = K.radialProfile(K.McorSD__yx, bin_r, return_npts=True)
>>> fig = plt.figure()
>>> plt.title('$M_{cor}$ and number of points')
>>> ax1 = fig.add_subplot(111)
>>> ax1.plot(bin_center, McorSD__r, '-k')
>>> ax1.set_xlabel(r'$R_{50}$')
>>> ax1.set_ylabel(r'$M_{cor} [M_\odot / pc^2]$ (solid)')
>>> ax2 = ax1.twinx()
```

```
>>> ax2.plot(bin_center, npts, '--k')
>>> ax2.set_ylabel(r'$N_{pts}$ (dashed)')
```



4.4.5 Radial profiles using exact apertures

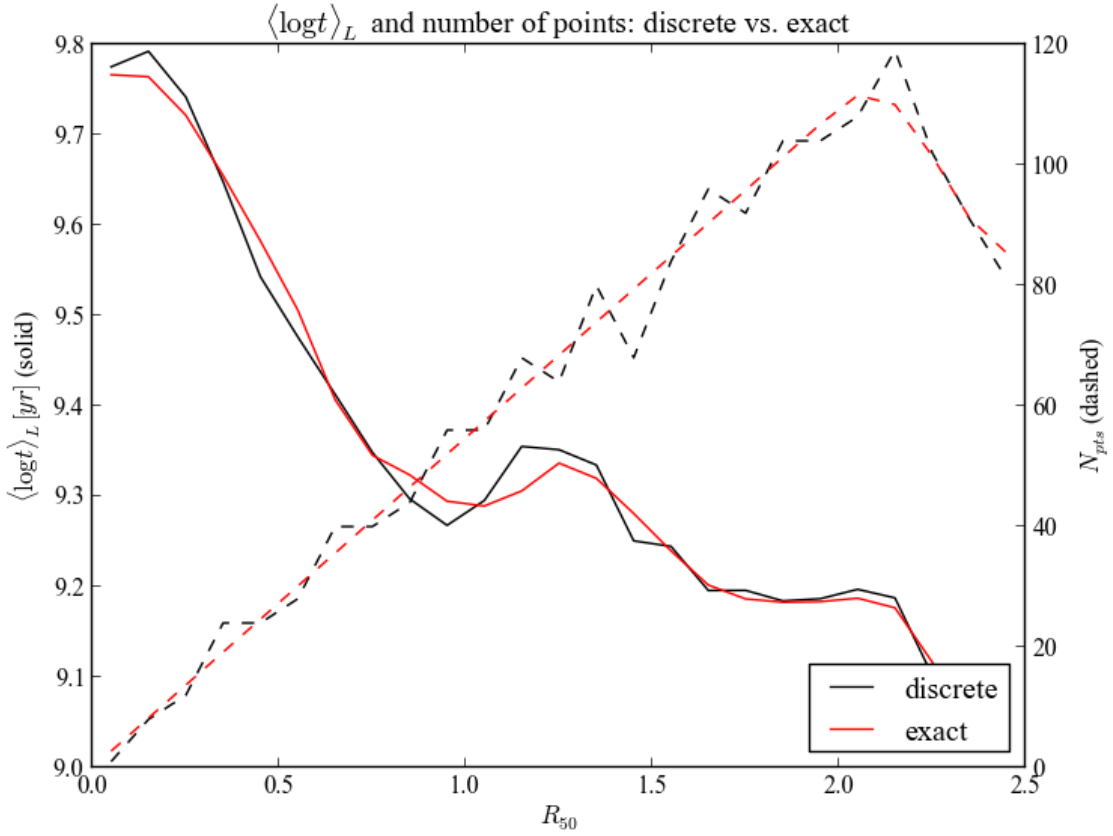
All the radial profiles calculated above suffer from a "jagginess" caused by the discrete method of assigning pixels to the radial bins (see the dashed line from the image above). When pixels fall in the boundaries of the bins, the whole pixel gets assigned to one or the other bin, based onthe pixel center. Often this is harmless, but for small radii (or conversely, large pixels) this effect can be a problem. The solution in this case is to "split" the pixel between the bins, calculating the intersection of the bin boundaries and each pixel. Needless to say, this can be very slow. To use the exact apertures algorithm (shamelessly "stolen" from photutils[2]), change the `mode` parameter to `'mean_exact'`.

Here we can see a comparison between the exact and the discrete versions of the radial profile of the luminosity weighted mean stellar age.

```
>>> bin_r = np.arange(0, 2.5+0.1, 0.1)
>>> bin_center = (bin_r[1:] + bin_r[:-1]) / 2.0
>>> atflux, npts = K.radialProfile(K.at_flux__yx, bin_r, \
mode='mean', return_npts=True)
>>> atflux_ex, npts_ex = K.radialProfile(K.at_flux__yx, bin_r, \
mode='mean_exact', return_npts=True)
>>> fig = plt.figure()
>>> plt.title(r'$\langle \log t \rangle_L$ and number of points: discrete vs. exact')
>>> ax1 = fig.add_subplot(111)
>>> ax1.plot(bin_center, atflux, '-k', label='discrete')
>>> ax1.plot(bin_center, atflux_ex, '-r', label='exact')
>>> ax1.set_xlabel(r'$R_{50}$')
```

[2] https://github.com/astropy/photutils

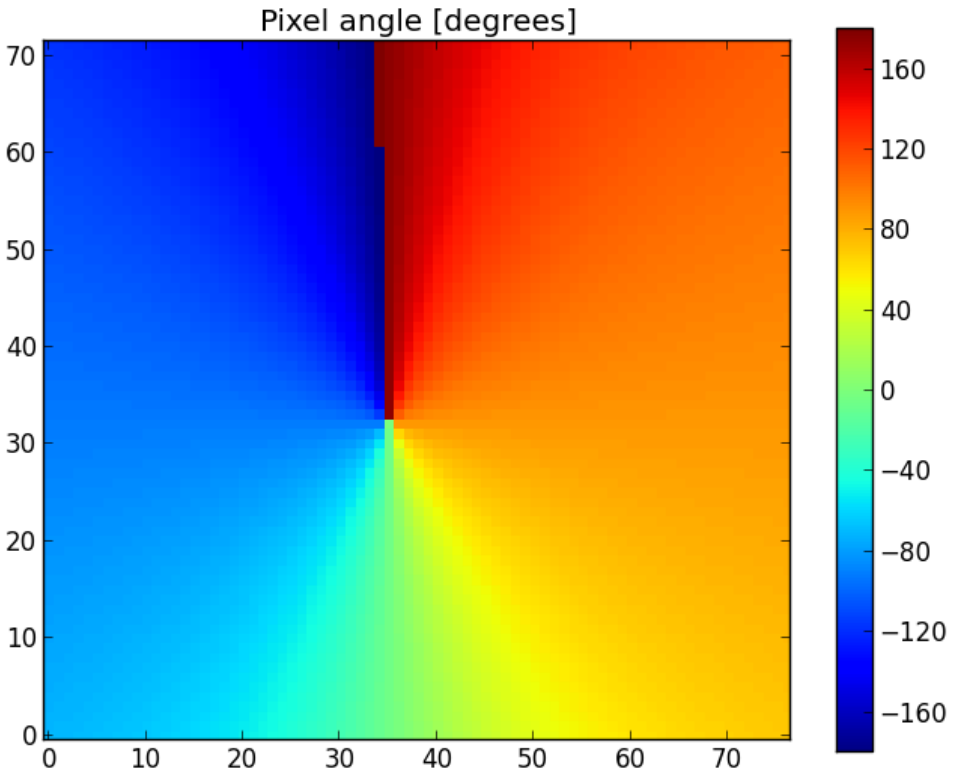
```
>>> ax1.set_ylabel(r'$\langle \log t \rangle_L [yr]$ (solid)')
>>> plt.legend(loc='lower right')
>>> ax2.plot(bin_center, npts_ex, '--r')
>>> ax2 = ax1.twinx()
>>> ax2.plot(bin_center, npts, '--k')
>>> ax2.set_ylabel(r'$N_{pts}$ (dashed)')
```



The bin area (dashed lines, right vertical axis) is much smoother when using the `'mean_exact'` algorithm (in red), compared to the discrete `'mean'` version (in black). As you can see, the age radial profile (solid lines, left vertical axis) becomes less noisy.

4.4.6 Azimuthal and radial profile of a N-D image

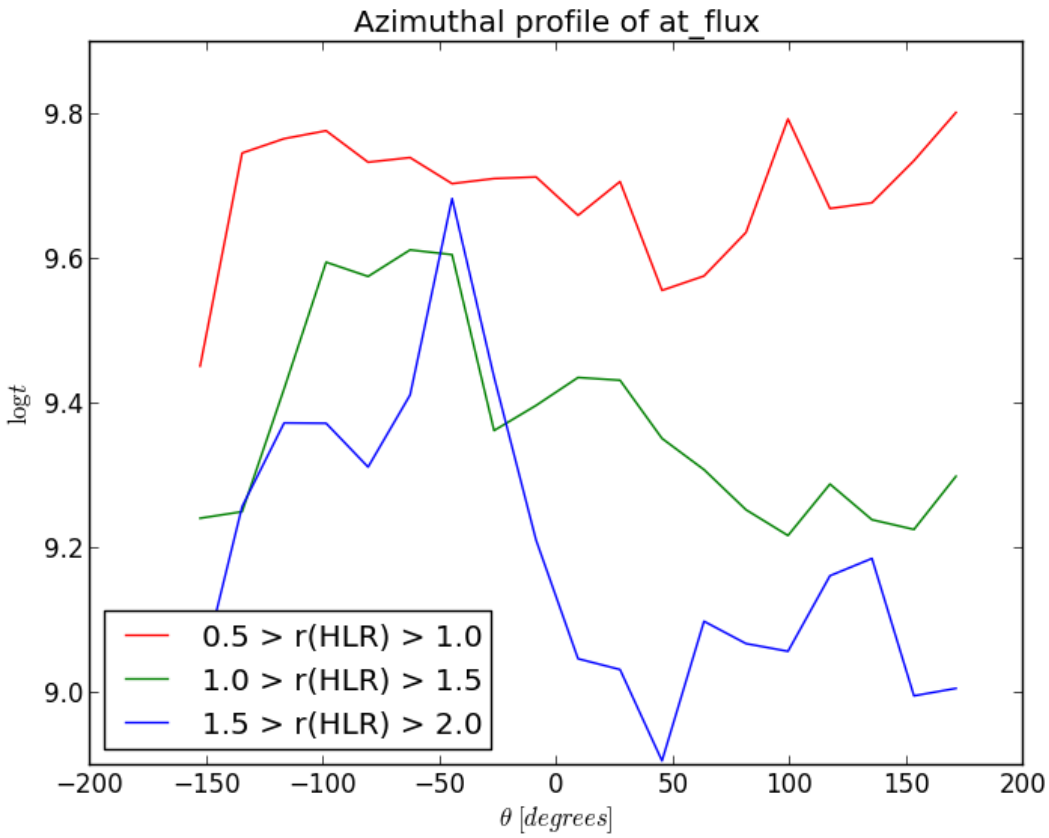
Still more generic than `radialProfile()` (page 63) is `azimuthalProfileNd()` (page 66). It adds the capability of azimuthal profiles, based on the angles associated with each pixel. This angle is accessible using `pixelAngle__yx` (page 51), in radians. The method `getPixelAngle()` (page 69) allows one to get the angle in degrees, or to get an arbitraty ellipse geometry, much like `getPixelDistance()` (page 68).



In PyCASSO the pixel angle is defined as the counter-clockwise angle between the pixel and the semimajor axis, taking the ellipse distortion into account. Please look at `getPixelAngle()` (page 69) for more details.

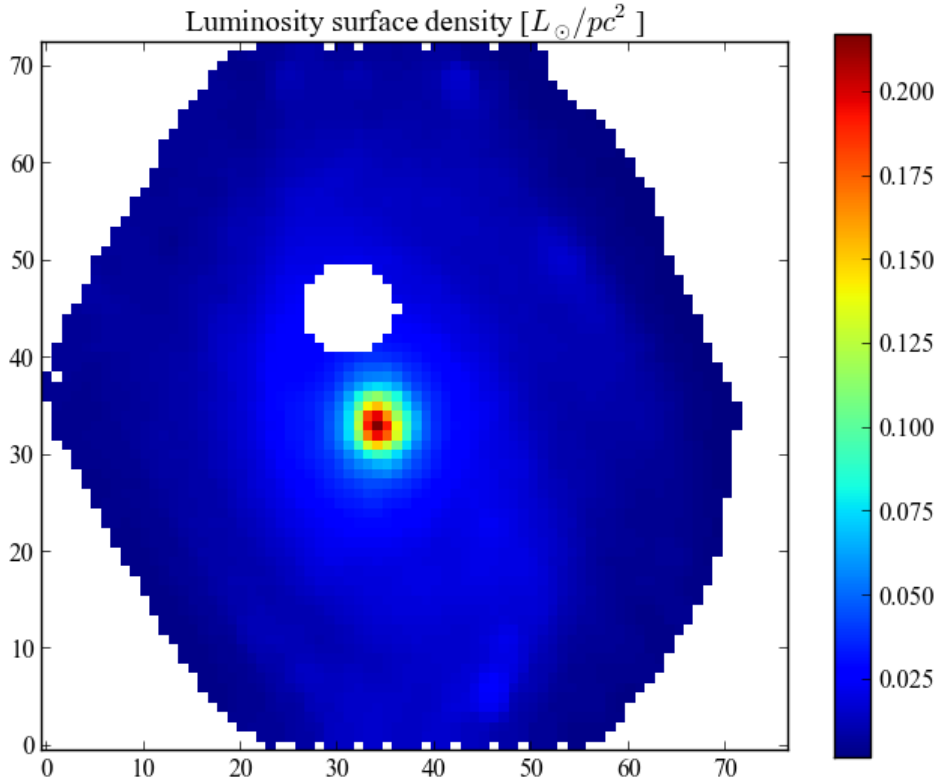
In the example below, we calculate the azimuthal profile of the log. age, averaged in flux, inside three rings between 0.5, 1.0, 1.5 and 2.0 HLR.

```
>>> bin_a = np.linspace(-np.pi, np.pi, 21)
>>> bin_a_center_deg = (bin_a[1:] + bin_a[:-1]) / 2.0 * 180 / np.pi
>>> bin_r = np.array((0.5, 1.0, 1.5, 2.0))
>>> at_flux__ar = K.azimuthalProfileNd(K.at_flux__yx, bin_a, bin_r)
>>> plt.plot(bin_a_center_deg, at_flux__ar[:,0], '-r', label='0.5 > r(HLR) > 1.0')
>>> plt.plot(bin_a_center_deg, at_flux__ar[:,1], '-g', label='1.0 > r(HLR) > 1.5')
>>> plt.plot(bin_a_center_deg, at_flux__ar[:,2], '-b', label='1.5 > r(HLR) > 2.0')
>>> plt.title('Azimuthal profile of mean stellar age')
>>> plt.ylabel(r'$\langle \log t \rangle_L$')
>>> plt.xlabel(r'$\theta\ [degrees]$')
>>> plt.legend()
```

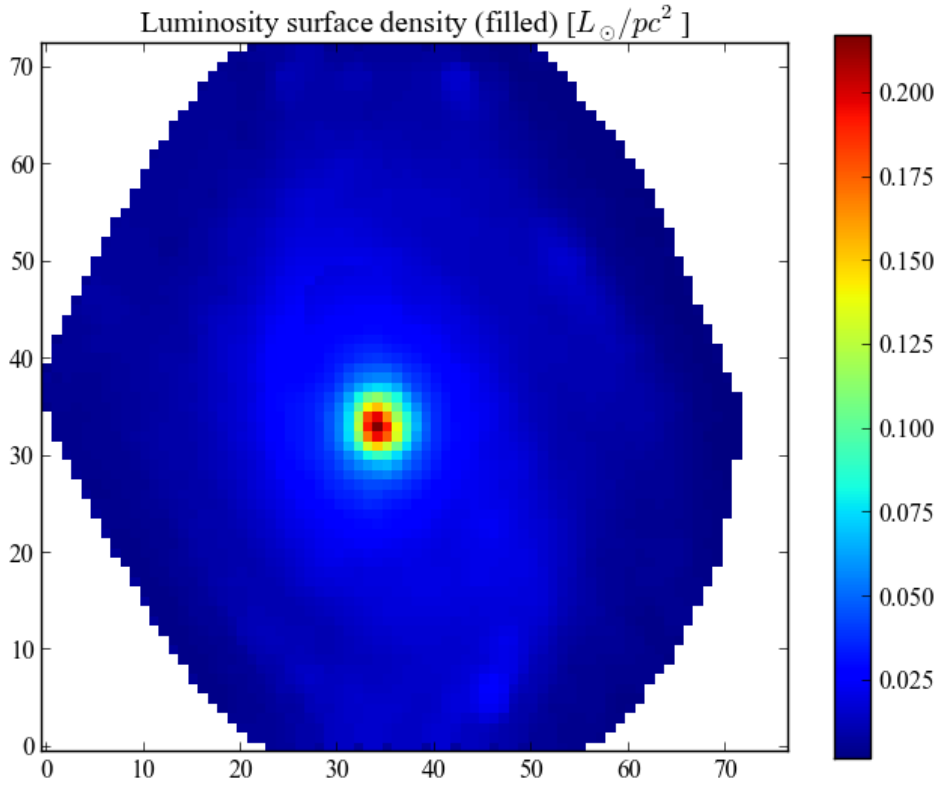


4.5 Filling missing data

The example galaxy used above does not have any masked foreground or background objects. This is not always the case. For example, galaxy K0277 (NGC2916) has a foreground star near it's center, which was masked in the analysis pipeline.



This impacts some measurements like the total mass of the galaxy or the cumulative radial curves. The latter can have a big effect in properties like the half light radius. To take the hollow areas into account, we can use the method `fillImage()` (page 67). By default, it takes a property as input, calculates its radial profile and interpolates the missing data from the radial profile. This assumes the user already called `setGeometry()` (page 60) with proper values. Alternatively, one can calculate a radial profile and use it as parameter when filling, see the documentation. Also, there's two modes of filling: `'hollow'` and `'convex'`. The first one only fills the hollow pixels, while the second fills the complete convex hull of the image (that is, fills the edges making the image "roundy"). Here's the same image as above, filling the convex hull.



Note the barely visible artifact in the filled areas. As an example, let's compare the values of the half light radius with and without filling. We will calculate the filling by hand to illustrate the usage. The methods `getHalfRadius()` (page 69) and `getHLR_pix()` (page 70) have a `fill` parameter, which when set to True, will perform the filling automatically.

```
>>> pa, ba = K.getEllipseParams()
>>> K.setGeometry(pa, ba)
>>> LobnSD_fill, mask = K.fillImage(K.LobnSD__yx, mode='hollow')
>>> K.getHalfRadius(LobnSD_fill, mask=mask)
19.43191121094409
>>> K.getHalfRadius(K.LobnSD__yx)
19.991108029801655
```

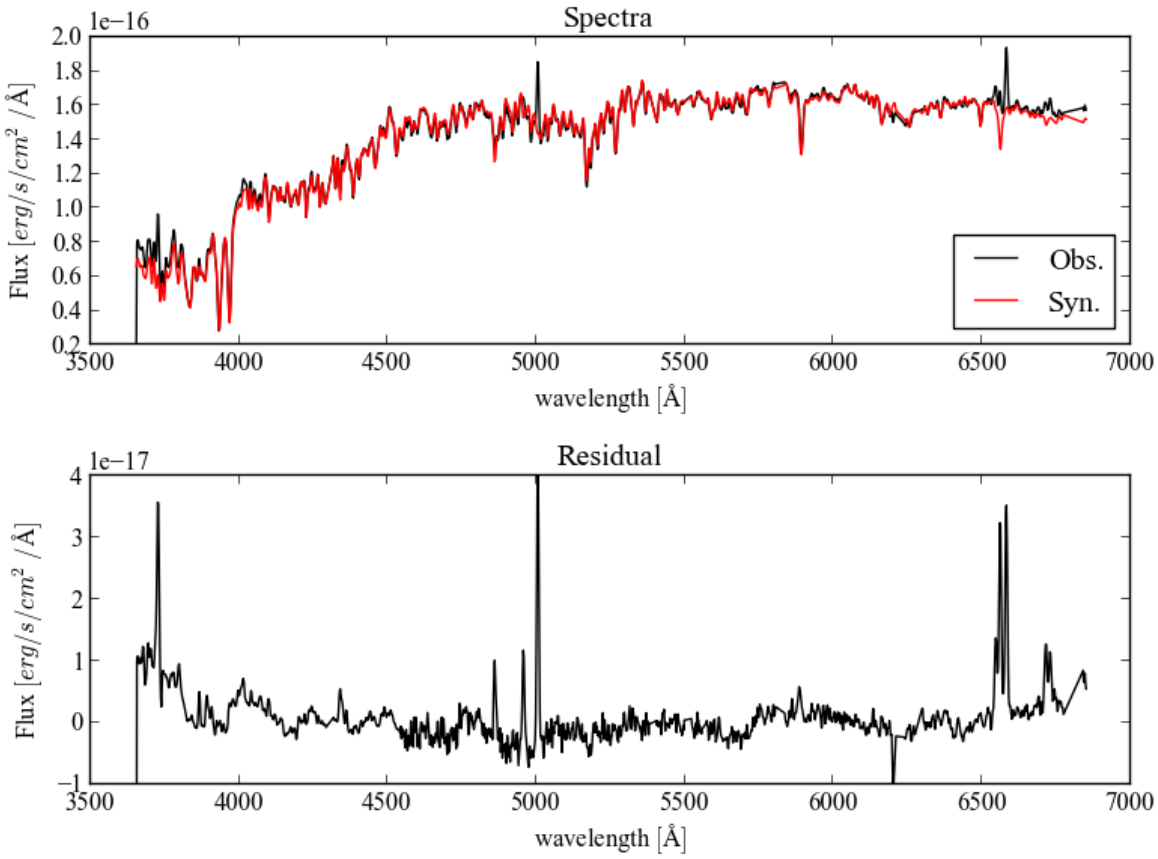
If one does not fill the hollow areas, the computed half light radius will be overestimated (in this case, by about 0.5 pixel, but it can be worse). The same applies to the total mass or luminosity of the galaxy.

4.6 Working with spectra

PyCASSO handles the stellar synthesis results from IFS data. One of the products of the stellar synthesis is a synthetic spetrum of each pixel (`f_syn__lyx` (page 77)), containing only the stellar contribution to the light. The observed spectra (`f_obs__lyx` (page 76)), errors (`f_err__lyx` (page 77)) and flags (`f_flag__lyx` (page 76)) are also available. Using these, one can easily, for example, obtain the residual spectra and measure emission lines.

```
>>> f_syn_nuc = K.f_syn__lyx[:, K.y0, K.x0]
>>> f_obs_nuc = K.f_obs__lyx[:, K.y0, K.x0]
>>> f_flag_nuc = K.f_flag__lyx[:, K.y0, K.x0]
>>> mask = f_flag_nuc < 1.0
>>> ax1 = plt.subplot(211)
```

```
>>> ax1.plot(K.l_obs[mask], f_obs_nuc[mask], '-k', label='Obs.')
>>> ax1.plot(K.l_obs[mask], f_syn_nuc[mask], '-r', label='Syn.')
>>> ax1.set_xlabel(r'wavelength $[\AA]$')
>>> ax1.set_ylabel(r'Flux $[erg / s / cm^2 / \AA]$')
>>> ax1.set_title('Spectra')
>>> ax1.legend()
>>> ax2 = plt.subplot(212)
>>> ax2.plot(K.l_obs[mask], f_obs_nuc[mask] - f_syn_nuc[mask], '-k')
>>> ax2.set_xlabel(r'wavelength $[\AA]$')
>>> ax2.set_ylabel(r'Flux $[erg / s / cm^2 / \AA]$')
>>> ax2.set_title('Residual')
```



Notice the emission lines are more evident in the residual. Thus using PyCASSO it is possible to analyze the spatially resolved spectral features like the $D_n(4000)$ and Lick indices.

4.7 Integrated properties

In addition to the spatially stellar synthesis, we performed the stellar synthesis in the integrated spectra of the galaxies. This was one can check if the sum of the parts is equal to the whole. In general the properties in integrated form contain the prefix `integrated_`. Please take a look at the section *Galaxy property list* (page 31). Let's take a look in the mass of the galaxy.

```
>>> K.integrated_Mcor / 1e10
6.4588941491889695
>>> (K.McorSD__yx * K.parsecPerPixel**2).sum() / 1e10
6.7011924815290183
>>> K.integrated_Mcor / (K.McorSD__yx * K.parsecPerPixel**2).sum()
0.9638425051947227
```

That is, the total mass of the galaxy, calculated using the stellar synthesis of the integrated spectra

(`integrated_Mcor`) is 4% smaller than the sum of the masses of the pixels, calculated in the same way.

CHAPTER

FIVE

LEGACY API

5.1 Loading from FITS files

The process of loading data for a galaxy in a FITS file is similar to the HDF5 one. The main difference is that you only have to specify the path to the file. One should use sensible file names in this case. Using the same example, let's load the data from the file `'K0001_synthesis_eBR_v20_q036.d13c512.ps03.k2.mC.CCM.Bgsd61.fits'`.

```
>>> from pycasso.fitsdatacube import fitsQ3DataCube
>>> K = fitsQ3DataCube('K0001_synthesis_eBR_v20_q036.d13c512.ps03.k2.mC.CCM.Bgsd61.fits')
```

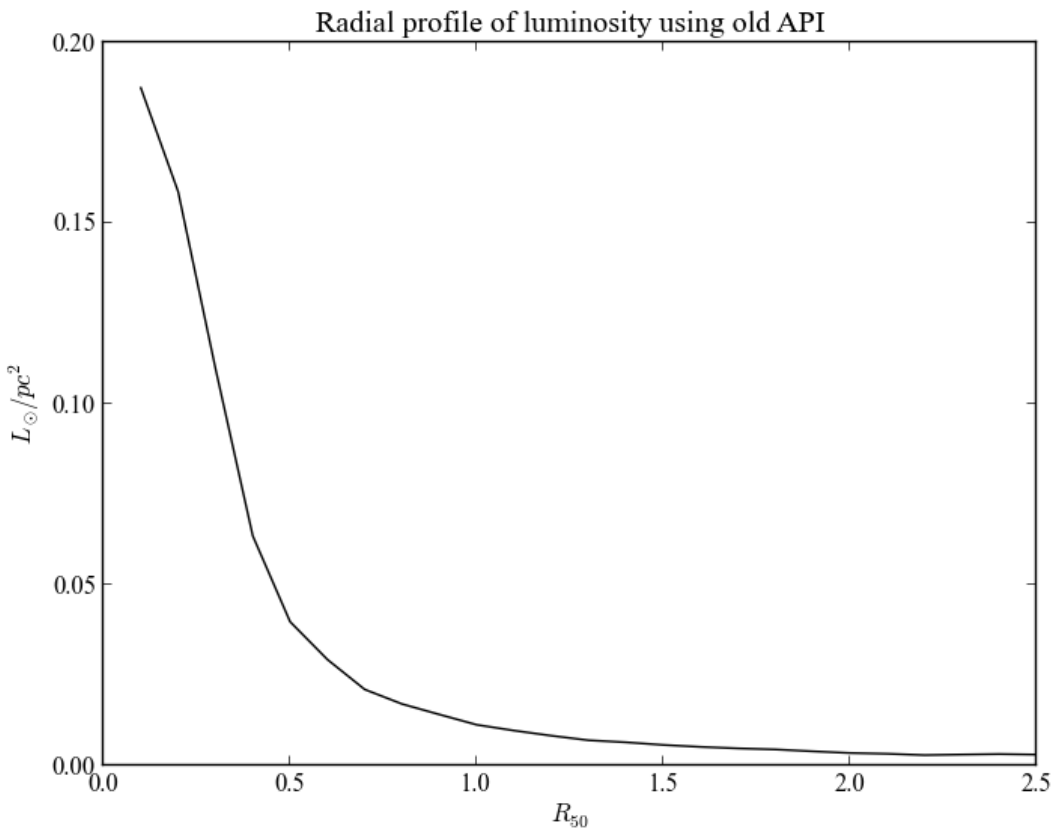
The object `K` is a `fitsQ3DataCube`, whick quacks almost exactly like a `h5Q3DataCube` (page 49), except for the dynamic loading of data present in the HDF5 implementation. Your script should work just like before.

5.2 Using very old scripts

There were several prototype versions of PyCASSO containing features which were abandoned, or renamed, or had changes in functionality. The class `Q3DataCube` (page 26) is meant to be a backwards compatibility layer for older scripts that used these features.

The main use for the old API is to convert from zone to XY notation, and produce radial profiles using the older dialect.

```
>>> from califa.Q3DataCube import Q3DataCube
>>> K = Q3DataCube('K0001_synthesis_eBR_v20_q036.d13c512.ps03.k2.mC.CCM.Bgsd61.fits')
>>> zyx = K.getZoneToYXTensor(extensive=True, dezonification=True)
>>> LobnSD__yx = np.tensordot(K.Lobn__z, zyx, (0, 0))
>>> ryx, bin_r, bin_area = K.getYXToRadialBinsTensor(d_r=0.1, \
rad_bin_ini=0.0, rad_bin_fin=2.5)
>>> LobnSD__r = np.tensordot(LobnSD__yx, ryx, [(0, 1), (1, 2)])
>>> plt.plot(bin_r, LobnSD__r, '-k')
>>> plt.xlim(0, 2.5)
>>> plt.xlabel(r'$R_{50}$')
>>> plt.ylabel(r'$L_\odot / pc^2$')
>>> plt.title('Radial profile of luminosity using old API')
```



All that could be easily attained using the methods `zoneToYX()` (page 59) and `radialProfile()` (page 63) without the hassle intermediary matrices and the order of the dimensions in tensordot.

5.3 Old API reference

class `califa.Q3DataCube.`**Q3DataCube**(*synthesisFile*, *circularProfile=True*, *runId=None*, *califaId=None*, *smoothDezonification=True*)

getZoneToYXTensor(*extensive=True*, *dezonification=True*)

Returns a tensor to transform from zone notation to spatial notation, with shape (zone, y, x). The normalization of the pixels can take into account the area of the zones or the luminosity weigth, using the following parameters:

Parameters **extensive** : boolean

A.K.A. "normalize by zone area". Use surface density when calculating the transformation from zone to spatial coordinates.

dezonification : boolean

Use image at 5650AA to weight pixels in the zones. This is used only when `extensive=True`.

Returns **zoneToYX** : 3-D array

The transform tensor, shape (nZones, y, x).

See Also:

`getYXToRadialBinsTensor` (page 27), `getZoneToRadialBinsTensor` (page 27)

Examples

To transform a cube named mycube, with axes (age, zone), into a cube with axes (age, y, x), one can perform the operation

```
>>> K = Q3DataCube(filename)
>>> zyx = K.getZoneToYXTensor()
>>> numpy.tensordot(mycube, zyx, (1, 0))
```

where 1 is the number of the zone axis in `mycube`, and 0 is the zone axis number in `zyx`.

getYXToRadialBinsTensor(*d_r*, *rad_bin_ini=0*, *rad_bin_fin=None*, *use_HLR_units=True*, *normalize_area=True*)

Compute a tensor that transforms an array from zone binning notation to radial binning notation. Given the radial profile for this galaxy, this method calculates a tensor similar to the `zonesToYXTensor`, it transforms an image from spatial coordinates (x,y) to a radial profile.

Parameters **d_r** : float

Radial step of the rings, in units of HLR.

rad_bin_ini : float, defaults to 0.0

Initial value for rad bins, in units of HLR.

rad_bin_fin : float, defaults to infinity

Final value for rad bins, in units of HLR.

use_HLR_units : boolean

If True, use the bins in HLR units. If False, use pixels.

normalize_area : boolean

If True, divide the tensor by the bin area.

Returns **YXToRad** : array, shape (rad. bins, y, x).

The transform tensor.

bin_R : array of float

Radial bin outer edges.

bin_area : array of float

Radial bin area, in pixels.

See Also:

`getZoneToRadialBinsTensor` (page 27), `getZoneToYXTensor` (page 26)

Examples

```
>>> c = Q3DataCube(file)
>>> yxToRad, bins, unused = c.getYXToRadialBinsTensor(d_r=0.2)
```

`c.LobnSD__yx` is the luminosity surface desity in each pixel. Compute the radial profile of the average luminosity surface density.

```
>>> radProf__R = np.tensordot(Lobn_tot__yx, yxToRad, [(0,1),(1,2)])
>>> plot(bins, radProf__R)
```

getZoneToRadialBinsTensor(*d_r*, *rad_bin_ini=0*, *rad_bin_fin=None*, *use_HLR_units=True*, *extensive=True*, *dezonification=False*)

Return a tensor that transforms an array from zone binning notation to radial binning notation. Given the radial profile for this galaxy, this method calculates a tensor similar to the `zonesToYXTensor`, it transforms an image from spatial coordinates (x,y) to a radial profile.

Parameters **d_r** : float

Radial step of the rings, in units of HLR_pix.

rad_bin_ini : float, defaults to 0.0

Initial value for rad bins, in units of HLR_pix.

rad_bin_fin : float, defaults to infinity

Final value for rad bins, in units of HLR_pix.

use_HLR_units : boolean

If True, use the bins in HLR units. If False, use pixels.

extensive : boolean

Use surface density when calculating the transformation from zone to spatial coordinates.

dezonification : boolean

Use image at 5650AA to weight pixels in the zones.

Returns **YXToRad** : array, shape (y, x, zone)

The transform tensor.

bin_R : array of float

Radial bin outer edges.

bin_area : array of float

Radial bin area, in pixels.

See Also:

`getYXToRadialBinsTensor` (page 27), `getZoneToYXTensor` (page 26)

Examples

```
>>> c = Q3DataCube(file)
>>> zoneToRad, bins, unused = c.getZoneToRadialBinsTensor(d_r=0.2)
```

`c.popx` is luminosity fraction per zone. `c.popx.shape` is (ageBase, metBase, zone). Get the flux per unit area (?) for each zone.

```
>>> Lobn = c.popx / 100.0 * c.Lobs_norm
```

Total luminosity in each zone.

```
>>> Lobn_tot = Lobn.sum(axis=1).sum(axis=0)
>>> radProf = np.tensordot(Lobn_tot, zoneToRad, (1,1))
>>> plot(bins, radProf)
```

findHLR_pix_CID()

TODO: Deprecated findHLR_pix_CID(). Find the half light radius using the image at 5650 Angstrom. Using radial bins of 1 pixel, calculate the cumulative sum of luminosity. The HLR is the radius where the cum. sum reaches 50% of its peak value.

Returns **HLR** : float

The half light radius, in pixels.

See Also:

`setGeometry`

Notes

This value should be close to $dfrac{HLR_{circular}}{sqrt{b/a}}$.

CHAPTER

SIX

GALAXY PROPERTY LIST

These are most of the attributes for a galaxy. Refer to section *Full data access API* (page 49) for a complete (and very long) list.

Some properties have a suffix like `__tZyx`. This is a mnemonic to the dimensions of the property. The dimensions are as following:

- `t`: age - `N_age` (page 50)
- `Z` (capital): metallicity - `N_met` (page 50)
- `z` (small): zone - `N_zone` (page 50)
- `y`: Y-position - `N_y` (page 50)
- `x`: X-position - `N_x` (page 50)
- `l`: wavelength - `Nl_obs` (page 50)

6.1 Base

`h5Q3DataCube.`**`ageBase`**
Ages of the base.

•Units: $[Yr]$

•Shape: `(N_zone)`

`h5Q3DataCube.`**`metBase`**
Metalicities of the base.

•Units: dimensionless

•Shape: `(N_zone)`

`h5Q3DataCube.`**`Mstars`**
Fraction of the initial stellar mass for a given population that is still trapped inside stars.

•Units: dimensionless

•Shape: `(N_age, N_met)`

`h5Q3DataCube.`**`fbase_norm`**
TODO: Add description of cubes.

6.2 qbick planes

`h5Q3DataCube.`**`qSignal`**
Signal at wavelength WINDOWSN.

•Units: $[erg/s/cm^2/\mathring{A}]$

•Shape: (N_y, N_x)

h5Q3DataCube.**qNoise**
TODO: Add description of cubes.

h5Q3DataCube.**qSn**
S/N at wavelength WINDOWSN.

•Units: dimensionless

•Shape: (N_y, N_x)

h5Q3DataCube.**qSignalUnmasked**
Image at wavelength WINDOWSN (quick, not masked).

•Units: $[erg/s/cm^2/\mathring{A}]$

•Shape: (N_y, N_x)

h5Q3DataCube.**qNoiseUnmasked**
TODO: Add description of qNoiseUnmasked.

h5Q3DataCube.**qPipeNoise**
TODO: Add description of qPipeNoise.

h5Q3DataCube.**qMask**
Boolean image mask used for data.

•Units: bool

•Shape: (N_y, N_x)

h5Q3DataCube.**qZones**
Voronoi/segmentation zones (bins).

•Units: index

•Shape: (N_y, N_x)

h5Q3DataCube.**qZonesSn**
S/N in Voronoi zones.

•Units: dimensionless

•Shape: (N_y, N_x)

h5Q3DataCube.**qZonesNoise**
Noise (RMS) in Voronoi zones.

•Units: $[erg/s/cm^2/\mathring{A}]$

•Shape: (N_y, N_x)

h5Q3DataCube.**qFlagRatio**
Ratio of flags in window at wavelength WINDOWSN

•Units: $[\%]$

•Shape: (N_y, N_x)

h5Q3DataCube.**qZonesSnOrig**
S/N in Voronoi zones (unresampled, beta).

•Units: dimensionless

•Shape: (N_y, N_x)

h5Q3DataCube.**qZonesNoiseOrig**
Noise (RMS) in Voronoi zones (unresampled, beta).

- Units: $[erg/s/cm^2/\mathring{A}]$
- Shape: `(N_y, N_x)`

`h5Q3DataCube.`**`qSegmentationSn`**
Voronoi program output S/N.

- Units: dimensionless
- Shape: `(N_y, N_x)`

`h5Q3DataCube.`**`qPipeNoiseOrig`**
Noise image derived from formal errors (unresampled).

- Units: $[erg/s/cm^2/\mathring{A}]$
- Shape: `(N_y, N_x)`

`h5Q3DataCube.`**`qPipeZonesNoiseOrig`**
Noise image derived from formal errors in Voronoi zones (unresampled).

- Units: $[erg/s/cm^2/\mathring{A}]$
- Shape: `(N_y, N_x)`

`h5Q3DataCube.`**`qSpatialMask`**
TODO: Add description of qSpatialMask.

`h5Q3DataCube.`**`qSnMask`**
TODO: Add description of qSnMask.

`h5Q3DataCube.`**`qFilledMask`**
TODO: Add description of qFilledMask.

`h5Q3DataCube.`**`qConvexHull`**
Convex hull of data mask.

- Units: bool
- Shape: `(N-y, N_x)`

`h5Q3DataCube.`**`qHollowPixels`**
Masked (bad) pixels inside the data mask.

- Units: bool
- Shape: `(N-y, N_x)`

6.3 Population

6.3.1 Zone

`h5Q3DataCube.`**`popx`**
Light fractions for each population, in voronoi zones.

- –Units: $[\%]$
- Shape: `(N_age, N_met, N_zone)`

`h5Q3DataCube.`**`popmu_ini`**
Initial mass fractions for each population, in voronoi zones.

- Units: $[\%]$
- Shape: `(N_age, N_met, N_zone)`

`h5Q3DataCube.`**`popmu_cor`**
Current mass fractions for each population, in voronoi zones.

•Units: $[\%]$

•Shape: `(N_age, N_met, N_zone)`

`h5Q3DataCube.`**`popAV_tot`**
Extinction for each population, in voronoi zones.

•Units: $[mag]$

•Shape: `(N_age, N_met, N_zone)`

`h5Q3DataCube.`**`Lobs_norm`**
Luminosity density in norm window, in voronoi zones.

•Units: $[L_\odot/\mathring{A}]$

•Shape: `(N_zone)`

`h5Q3DataCube.`**`Mini_tot`**
Initial mass for each population, in voronoi zones.

•Units: $[M_\odot]$

•Shape: `(N_zone)`

`h5Q3DataCube.`**`Mcor_tot`**
Current mass for each population, in voronoi zones.

•Units: $[M_\odot]$

•Shape: `(N_zone)`

6.3.2 Spatially resolved

`h5Q3DataCube.`**`popx__tZyx`**
Spatially resolved light fractions for each population.

•Units: $[\%]$

•Shape: `(N_age, N_met, N_y, N_x)`

`h5Q3DataCube.`**`popmu_ini__tZyx`**
Spatially resolved initial mass fractions for each population.

•Units: $[\%]$

•Shape: `(N_age, N_met, N_y, N_x)`

`h5Q3DataCube.`**`popmu_cor__tZyx`**
Spatially resolved corrected mass fractions for each population.

•Units: $[\%]$

•Shape: `(N_age, N_met, N_y, N_x)`

`h5Q3DataCube.`**`popAV_tot__tZyx`**
Spatially resolved extinction by dust for each population.

•Units: $[mag]$

•Shape: `(N_age, N_met, N_y, N_x)`

6.3.3 Integrated

`h5Q3DataCube.`**`integrated_popmu_ini`**
Current mass fractions for each population, in integrated spectrum.

•Units: $[\%]$

•Shape: (N_age, N_met)

h5Q3DataCube.**integrated_popmu_cor**
Current mass fractions for each population, in integrated spectrum.

•Units: $[\%]$

•Shape: (N_age, N_met)

h5Q3DataCube.**integrated_popAV_tot**
Extinction by dust for each population, in integrated spectrum.

•Units: $[mag]$

•Shape: (N_age, N_met)

6.4 Physical properties

See also HLR_pix (page 50), HLR_pc (page 50) and q_norm (page 50).

6.4.1 Zone

h5Q3DataCube.**Mini__tZz**
Initial mass of each population, in voronoi zones.

•Units: $[M_\odot]$

•Shape: (N_age, N_met, N_zone)

h5Q3DataCube.**Mcor__tZz**
Current mass of each population, in voronoi zones.

•Units: $[M_\odot]$

•Shape: (N_age, N_met, N_zone)

h5Q3DataCube.**Lobn__tZz**
Luminosity of each population in normalization window, in voronoi zones.

•Units: $[L_\odot]$

•Shape: (N_age, N_met, N_zone)

h5Q3DataCube.**DeRed_Lobn__tZz**
"Dereddened" luminosity of each population in normalization window, in voronoi zones.

•Units: $[L_\odot]$

•Shape: (N_age, N_met, N_zone)

h5Q3DataCube.**Mini__z**
Initial mass, in voronoi zones.

•Units: $[M_\odot]$

•Shape: (N_age, N_met, N_zone)

h5Q3DataCube.**Mcor__z**
Current mass, in voronoi zones.

•Units: $[M_\odot]$

•Shape: (N_age, N_met, N_zone)

h5Q3DataCube.**Lobn__z**
Luminosity in normalization window, in voronoi zones.

•Units: $[L_\odot]$

•Shape: (N_zone)

h5Q3DataCube.**DeRed_Lobn__z**
"Dereddened" luminosity in normalization window, in voronoi zones.

•Units: $[L_\odot]$

•Shape: (N_zone)

h5Q3DataCube.**A_V**
Extinction by dust, in voronoi zones.

•Units: $[mag]$

•Shape: (N_zone)

h5Q3DataCube.**v_0**
Velocity displacement, in voronoi zones.

•Units: $[km/s]$

•Shape: (N_zone)

h5Q3DataCube.**v_d**
Velocity dispersion, in voronoi zones.

•Units: $[km/s]$

•Shape: (N_zone)

6.4.2 Spatially resolved

h5Q3DataCube.**MiniSD__tZyx**
Spatially resolved initial mass surface density of each population.

•Units: $[M_\odot/pc^2]$

•Shape: (N_age, N_met, N_y, N_x)

h5Q3DataCube.**McorSD__tZyx**
Spatially resolved current mass surface density of each population.

•Units: $[M_\odot/pc^2]$

•Shape: (N_age, N_met, N_y, N_x)

h5Q3DataCube.**LobnSD__tZyx**
Spatially resolved luminosity surface density of each population in normalization window.

•Units: $[L_\odot/pc^2]$

•Shape: (N_age, N_met, N_y, N_x)

h5Q3DataCube.**DeRed_LobnSD__tZyx**
Spatially resolved "dereddened" luminosity surface density of each population in normalization window.

•Units: $[L_\odot/pc^2]$

•Shape: (N_age, N_met, N_y, N_x)

h5Q3DataCube.**MiniSD__yx**
Spatially resolved initial mass surface density.

•Units: $[M_\odot/pc^2]$

•Shape: (N_y, N_x)

h5Q3DataCube.**McorSD__yx**
Spatially resolved current mass surface density.

•Units: $[M_\odot/pc^2]$

- Shape: (N_y, N_x)

h5Q3DataCube.**LobnSD__yx**
Luminosity surface density of each population in normalization window.

- Units: $[L_\odot/pc^2]$
- Shape: (N_y, N_x)

h5Q3DataCube.**DeRed_LobnSD__yx**
"Dereddened" luminosity surface density of each population in normalization window.

- Units: $[L_\odot/pc^2]$
- Shape: (N_y, N_x)

h5Q3DataCube.**M2L__yx**
Spatially resolved mass to light ratio.

- Units: $[M_\odot/L_\odot]$
- Shape: (N_y, N_x)

h5Q3DataCube.**DeRed_M2L__yx**
Spatially resolved "dereddened" mass to light ratio.

- Units: $[M_\odot/L_\odot]$
- Shape: (N_y, N_x)

h5Q3DataCube.**A_V__yx**
Spatially resolved extinction by dust.

- Units: $[mag]$
- Shape: (N_y, N_x)

h5Q3DataCube.**v_0__yx**
Spatially resolved velocity displacement.

- Units: $[km/s]$
- Shape: (N_y, N_x)

h5Q3DataCube.**v_d__yx**
Spatially resolved velocity dispersion.

- Units: $[km/s]$
- Shape: (N_y, N_x)

h5Q3DataCube.**at_flux__yx**
Spatially resolved, flux-weighted average log. age.

- Units: $[\log Gyr]$
- Shape: (N_y, N_x)

h5Q3DataCube.**at_mass__yx**
Spatially resolved, mass-weighted average log. age.

- Units: $[\log Gyr]$
- Shape: (N_y, N_x)

h5Q3DataCube.**aZ_flux__yx**
Spatially resolved, flux-weighted average metallicity.

- Units: dimensionless
- Shape: (N_y, N_x)

h5Q3DataCube.**aZ_mass__yx**
Spatially resolved, mass-weighted average metallicity.

•Units: dimensionless

•Shape: `(N_y, N_x)`

6.4.3 Integrated

`h5Q3DataCube.`**`integrated_Lobn`**
Luminosity in normalization window of the integrated spectrum.

•Units: $[L_\odot]$

•Type: float

`h5Q3DataCube.`**`integrated_Lobn__tZ`**
Luminosity of each population in normalization window of the integrated spectrum.

•Units: $[L_\odot]$

•Shape: `(N_age, N_met)`

`h5Q3DataCube.`**`integrated_DeRed_Lobn`**
"Dereddened" luminosity in normalization window of the integrated spectrum.

•Units: $[L_\odot]$

•Type: float

`h5Q3DataCube.`**`integrated_DeRed_Lobn__tZ`**

"Dereddened" luminosity of each population in normalization window of the integrated spectrum.

- Units: $[L_\odot]$
- Shape: `(N_age, N_met)`

`h5Q3DataCube.`**`integrated_Mcor`**
Current mass of the integrated spectrum.

•Units: $[M_\odot]$

•Type: float

`h5Q3DataCube.`**`integrated_Mcor__tZ`**
Current mass of each population of the integrated spectrum.

•Units: $[M_\odot]$

•Shape: `(N_age, N_met)`

`h5Q3DataCube.`**`integrated_Mini`**
Initial mass of the integrated spectrum.

•Units: $[M_\odot]$

•Type: float

`h5Q3DataCube.`**`integrated_Mini__tZ`**
Initial mass of each population of the integrated spectrum.

•Units: $[M_\odot]$

•Shape: `(N_age, N_met)`

`h5Q3DataCube.`**`integrated_M2L`**
Mass to light ratio of the integrated spectrum.

•Units: $[M_\odot/L_\odot]$

•Type: float

`h5Q3DataCube.`**`integrated_DeRed_M2L`**
"Dereddened" mass to light ratio of the integrated spectrum.

•Units: $[M_\odot/L_\odot]$

•Type: float

h5Q3DataCube.**integrated_at_flux**
Flux-weighted average log. age of the integrated spectrum.

•Units: $[\log Gyr]$

•Type: float

h5Q3DataCube.**integrated_at_mass**
Mass-weighted average log. age of the integrated spectrum.

•Units: $[\log Gyr]$

•Type: float

h5Q3DataCube.**integrated_aZ_flux**
Flux-weighted average metallicity of the integrated spectrum.

•Units: dimensionless

•Type: float

h5Q3DataCube.**integrated_aZ_mass**
Mass-weighted average metallicity of the integrated spectrum.

•Units: dimensionless

•Type: float

6.5 Spectra

6.5.1 Zone

h5Q3DataCube.**l_obs**
Wavelength array for the spectral.

•Units: $[\mathring{A}]$

•Shape: `(Nl_obs)`

h5Q3DataCube.**f_obs**
Observed flux (input spectra for the synthesis), in voronoi zones.

•Units: $[erg/s/cm^2/\mathring{A}]$

•Shape: `(Nl_obs, N_zone)`

h5Q3DataCube.**f_err**
Error in observed spetra, in voronoi zones.

•Units: $[erg/s/cm^2/\mathring{A}]$

•Shape: `(Nl_obs, N_zone)`

h5Q3DataCube.**f_flag**
Flagged spaxels, in voronoi zones.

FIXME: describe flags.

•Units: dimensionless

•Shape: `(Nl_obs, N_zone)`

h5Q3DataCube.**f_syn**
Synthetic spectra, in voronoi zones.

•Units: $[erg/s/cm^2/\overset{\circ}{A}]$

•Shape: `(Nl_obs, N_zone)`

h5Q3DataCube.**f_wei**
Weight of the spaxels in the input spectra. This is the weight actually used by the synthesis, after clipping, etc. Values for voronoi zones.

FIXME: describe flags and weights.

•Units: dimensionless

•Shape: `(Nl_obs, N_zone)`

6.5.2 Spatially resolved

h5Q3DataCube.**f_obs__lyx**
Spatially resolved observed flux (input spectra for the synthesis).

•Units: $[erg/s/cm^2/\overset{\circ}{A}]$

•Shape: `(Nl_obs, N_y, N_x)`

h5Q3DataCube.**f_err__lyx**
Spatially resolved error in observed spetra.

•Units: $[erg/s/cm^2/\overset{\circ}{A}]$

•Shape: `(Nl_obs, N_y, N_x)`

h5Q3DataCube.**f_flag__lyx**
Spatially resolved flagged spaxels.

FIXME: describe flags.

•Units: dimensionless

•Shape: `(Nl_obs, N_y, N_x)`

h5Q3DataCube.**f_syn__lyx**
Spatially resolved synthetic spectra.

•Units: $[erg/s/cm^2/\overset{\circ}{A}]$

•Shape: `(Nl_obs, N_y, N_x)`

h5Q3DataCube.**f_wei__lyx**
Spatially resolved weight of the spaxels in the input spectra. This is the weight actually used by the synthesis, after clipping, etc.

FIXME: describe flags and weights.

•Units: dimensionless

•Shape: `(Nl_obs, N_y, N_x)`

6.5.3 Integrated

h5Q3DataCube.**integrated_f_obs**
Observed flux (input spectra for the synthesis), in integrated spectrum.

•Units: $[erg/s/cm^2/\overset{\circ}{A}]$

•Shape: `(Nl_obs)`

h5Q3DataCube.**integrated_f_err**
Error in integrated observed spetrum.

•Units: $[erg/s/cm^2/\mathring{A}]$

•Shape: `(Nl_obs)`

`h5Q3DataCube.`**`integrated_f_flag`**
Flagged spaxels, in integrated spectrum.

FIXME: describe flags.

•Units: dimensionless

•Shape: `(Nl_obs)`

`h5Q3DataCube.`**`integrated_f_syn`**
Synthetic integrated spectrum.

•Units: $[erg/s/cm^2/\mathring{A}]$

•Shape: `(Nl_obs)`

`h5Q3DataCube.`**`integrated_f_wei`**
Weight of the spaxels in the integrated input spectra. This is the weight actually used by the synthesis, after clipping, etc.

FIXME: describe flags and weights.

•Units: dimensionless

•Shape: `(Nl_obs, N_y, N_x)`

6.6 Synthesis fit diagnostics

`h5Q3DataCube.`**`adev`**
Mean absolute relative deviation, in percent, only for the `Nl_eff` points actually used in the synthesis.

•Units: $[\%]$

•Shape: `(N_zone)`

`h5Q3DataCube.`**`adevS`**
From Cid @ 26/05/2012: Here's my request for pycasso reader: That it defines a new figure of merit analogous to `adev`, but which uses the synthetic flux in the denominator instead of the observed one. This is the `adevS__z` thing defined below in awful python. Why? Well, despite all our care there are still some non-flagged pixels with very low fluxes, which screws up adev, and this alternative definition fixes it.

Original code:

```
>>> adevS__z = np.zeros((self.nZones))
>>> for i_z in np.arange(self.nZones):
>>>     _a = np.abs(self.f_obs[:,i_z] - self.f_syn[:,i_z]) / self.f_syn[:,i_z]
>>>     _b = _a[self.f_wei[:,i_z] > 0]
>>>     adevS__z[i_z] = 100.0 * _b.mean()
>>> return adevS__z
```

Returns **aDevS** : array of length (nZones)

`h5Q3DataCube.`**`NOl_eff`**
Number of OK wavelengths in input spectrum, in voronoi zones.

•Units: dimensionless

•Shape: `(N_zone)`

`h5Q3DataCube.`**`Nglobal_steps`**
Number of steps in spectral fitting, in voronoi zones.

•Units: dimensionless

- Shape: `(N_zone)`

`h5Q3DataCube.`**`chi2`**
χ^2/Nl_{eff} of the fit, in voronoi zones.

- Units: dimensionless
- Shape: `(N_zone)`

CHAPTER

SEVEN

UTILITY FUNCTIONS

Here we have a few functions that are used by internally by PyCASSO but may be useful anyway.

7.1 Function list

Created on Oct 25, 2012

@author: andre

`pycasso.util.`**`arrayAsRowMatrix`**(*a*, *ndim*)

Reshape `a` to a "row matrix" of bigger rank, `ndim`.

Parameters **a** : array

Array to reshape.

ndim : integer

Number of dimensions to reshape.

Returns **a_reshaped** : array

The same array as `a`, reshaped to `ndim` dimensions.

`pycasso.util.`**`gen_rebin`**(*a*, *e*, *bin_e*, *mean=True*)

Rebinning function. Given the value array *a*, with generic positions *e* such that *e.shape == a.shape*, return the sum or the mean values of *a* inside the bins defined by *bin_e*.

Parameters **a** : array like

The array values to be rebinned.

e : array like

Generic positions of the values in *a*.

bin_e : array like

Bins of *e* to be used in the rebinning.

mean : boolean

Divide the sum by the number of points inside the bin. Defaults to *True*.

Returns **a_e** : array

An array of length *len(bin_e)-1*, containing the sum or mean values of *a* inside each bin.

Examples

TODO: Add examples for gen_rebin.

pycasso.util.**getGenHalfRadius**(*X*, *r*, *r_max=None*)

Evaluate radius where the cumulative value of *X* reaches half of its value.

Parameters **X** : array like

The property whose half radius will be evaluated.

r : array like

Radius associated to each value of *X*. Must be the same shape as *X*.

r_max : int

Integrate up to *r_max*. Defaults to *np.max(r)*.

Returns **HXR** : float

The "half X radius."

Examples

Find the radius containing half of the volume of a gaussian.

```
>>> import numpy as np
>>> xx, yy = np.indices((100, 100))
>>> x0, y0, A, a = 50.0, 50.0, 1.0, 20.0
>>> z = A * np.exp(-((xx-x0)**2 + (yy-y0)**2)/a**2)
>>> r = np.sqrt((xx - 50)**2 + (yy-50)**2)
>>> getGenHalfRadius(z, r)
16.786338066912215
```

pycasso.util.**getAverageFilledImage**(*X*, *to_fill*, *r__yx*, *X__r*, *bin_r*)

TODO: getAverageFilledImage documentation.

pycasso.util.**getApertureMask**(*pa*, *ba*, *x0*, *y0*, *N_x*, *N_y*, *apertures*)

TODO: Documentation for apertures.

pycasso.util.**convexHullMask**(*mask*)

Compute the convex hull of a boolean image mask.

Parameters **mask** : array

2-D image where the `True` pixels mark the data.

Returns **convex_mask** : array

2-D image of same shape and type as `mask`, where the convex hull of `mask` is marked as `True` values.

pycasso.util.**getEllipseParams**(*image*, *x0=0.0*, *y0=0.0*, *mask=None*)

Estimate ellipticity and orientation of the galaxy using the "Stokes parameters", as described in: http://adsabs.harvard.edu/abs/2002AJ....123..485S The image used is `qSignal`.

Parameters **image** : array

Image to use when calculating the ellipse parameters.

x0 : float

X coordinate of the origin. Defaults to `0.0`.

y0 : float

Y coordinate of the origin. Defaults to `0.0`.

mask : array, optional

Mask containing the pixels to take into account.

Returns **pa** : float

Position angle in radians, counter-clockwise relative to the positive X axis.

ba : float

Ellipticity, defined as the ratio between the semiminor axis and the semimajor axis (b/a).

pycasso.util.**getDistance**(*x*, *y*, *x0=0.0*, *y0=0.0*, *pa=0.0*, *ba=1.0*)

Return an image (numpy.ndarray[1]) of the distance from the center `(x0, y0)` in pixels, assuming a projected disk.

Parameters **x** : array

X coordinates to get the pixel distances.

y : array

y coordinates to get the pixel distances.

x0 : float, optional

X coordinate of the origin. Defaults to `0.0`.

y0 : float, optional

Y coordinate of the origin. Defaults to `0.0`.

pa : float, optional

Position angle in radians, counter-clockwise relative to the positive X axis.

ba : float, optional

Ellipticity, defined as the ratio between the semiminor axis and the semimajor axis (b/a).

Returns **pixelDistance** : array

Pixel distances.

See Also:

getImageDistance (page 45)

pycasso.util.**getImageDistance**(*shape*, *x0=0.0*, *y0=0.0*, *pa=0.0*, *ba=1.0*)

Return an image (numpy.ndarray[2]) of the distance from the center `(x0, y0)` in pixels, assuming a projected disk.

Parameters **shape** : (float, float)

Shape of the image to get the pixel distances.

x0 : float, optional

X coordinate of the origin. Defaults to `0.0`.

y0 : float, optional

Y coordinate of the origin. Defaults to `0.0`.

pa : float, optional

Position angle in radians, counter-clockwise relative to the positive X axis. Defaults to `0.0`.

ba : float, optional

[1] http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray
[2] http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray

Ellipticity, defined as the ratio between the semiminor axis and the semimajor axis (b/a). Defaults to `1.0`.

Returns **pixelDistance** : 2-D array

Image containing the distances.

See Also:

`getDistance` (page 45), `getEllipseParams` (page 44)

`pycasso.util.`**`getAngle`**(*x*, *y*, *x0=0.0*, *y0=0.0*, *pa=0.0*, *ba=1.0*)

Return an image (`numpy.ndarray`[3] of same shape as :attr'qSignal') of the angle in radians of each pixel, relative from the axis of the position angle `pa`. The projection is fixed assuming the galaxy is a disk, throught the ellipticity parameter `ba`.

The angle is obtained "de-rotating" the pixel positions, stretching the y-coordinates to account for the perspective, and then calculating the arc tangent of the resulting y/x.

Parameters **x** : array

X coordinates to calculate the angle.

y : array

Y coordinates to calculate the angle.

x0 : float, optional

X coordinate of the origin. Defaults to `0.0`.

y0 : float, optional

Y coordinate of the origin. Defaults to `0.0`.

pa : float, optional

Position angle in radians, counter-clockwise relative to the positive X axis. Defaults to `0`.

ba : float, optional

Ellipticity, defined as the ratio between the semiminor axis and the semimajor axis (b/a). This controls the corretion of the projection of the galaxy. Set to `1.0` (default) to disable the correction.

Returns **pixelAngle** : 2-D array

Image containing the angles in radians.

See Also:

`getPixelDistance`

`pycasso.util.`**`getImageAngle`**(*shape*, *x0=0.0*, *y0=0.0*, *pa=0.0*, *ba=1.0*)

Return an image (`numpy.ndarray`[4] of same shape as :attr'qSignal') of the angle in radians of each pixel, relative from the axis of the position angle `pa`. The projection is fixed assuming the galaxy is a disk, throught the ellipticity parameter `ba`.

The angle is obtained "de-rotating" the pixel positions, stretching the y-coordinates to account for the perspective, and then calculating the arc tangent of the resulting y/x.

Parameters **shape** : (float, float)

Shape of the image to get the angles.

x0 : float, optional

X coordinate of the origin. Defaults to `0.0`.

[3] http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray
[4] http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray

y0 : float, optional

Y coordinate of the origin. Defaults to `0.0`.

pa : float, optional

Position angle in radians, counter-clockwise relative to the positive X axis. Defaults to `0`.

ba : float, optional

Ellipticity, defined as the ratio between the semiminor axis and the semimajor axis (b/a). This controls the corretion of the projection of the galaxy. Set to `1.0` (default) to disable the correction.

Returns **pixelAngle** : 2-D array

Image containing the angles in radians.

See Also:

`getPixelDistance`

pycasso.util.**radialProfile**(*prop*, *r__yx*, *bin_r*, *rad_scale=1.0*, *mask=None*, *mode='mean'*, *return_npts=False*)

Calculate the radial profile of an N-D image.

Parameters **prop** : array

Image of property to calculate the radial profile.

r__yx : array

Distance of each pixel to the galaxy center, in pixels.

bin_r : array

Semimajor axis bin boundaries in units of `rad_scale`.

rad_scale : float, optional

Scale of the bins, in pixels. Defaults to 1.0.

mask : array, optional

Mask containing the pixels to use in the radial profile. Must have the shape `(N_y, N_x)`. Defaults to `qMask`.

mode : string, optional

One of:

- `'mean'`: Compute the mean inside the radial bins (default).
- `'median'`: Compute the median inside the radial bins.
- `'sum'`: Compute the sum inside the radial bins.
- `'var'`: Compute the variance inside the radial bins.

return_npts : bool, optional

If set to `True`, also return the number of points inside each bin. Defaults to `False`.

Returns **radProf** : [masked] array

Array containing the radial profile as the last dimension. Note that `radProf.shape[-1] == (len(bin_r) - 1)` If `prop` is a masked aray, this and `npts` will be a masked array as well.

npts : [masked] array, optional

The number of points inside each bin, only if `return_npts` is set to `True`.

See Also:

`getPixelDistance`

CHAPTER

EIGHT

FULL DATA ACCESS API

8.1 HDF5 query functions

pycasso.h5datacube.**listRuns**(*synthesisFile*, *baseId=None*, *qbickRunId=None*)
Prints STARLIGHT runs contained in synthesis dataset.

Parameters **synthesisFile** : string

Path to an HDF5 file containing the synthesis.

baseId [string] Show results which use base given by `baseId`.

qbickRunId [string] Show results which use qbick run given by `qbickRunId`.

See Also:

`listBases` (page 49), `listQbickRuns` (page 49)

pycasso.h5datacube.**listBases**(*synthesisFile*)
Prints bases used in STARLIGHT runs contained in synthesis dataset..

Parameters **synthesisFile** : string

Path to an HDF5 file containing the synthesis.

pycasso.h5datacube.**listQbickRuns**(*synthesisFile*)
Prints qbick runs contained in synthesis dataset.

Parameters **synthesisFile** : string

Path to an HDF5 file containing the synthesis.

8.2 Synthesis data abstraction class

The class `h5Q3DataCube` (page 49) manages the HDF5 interface and implements `IQ3DataCube` (page 59), which contains the storage-agnostic methods for data handling.

class pycasso.h5datacube.**h5Q3DataCube**(*synthesisFile*, *runId*, *califaId*, *smooth=True*)
Interface to CALIFA HDF5 synthesis dataset.

baseId = None
Identifier of the base used in the synthesis.

qbickRunId = None
Identifier of the qbick run used in the synthesis.

pycassoVersion = None
Version of PyCASSO used to import the synthesis.

qVersion = None
Version of qbick.

galaxyName = None
NED name of the galaxy.

califaID = None
CALIFA name of the galaxy.

N_age = None
Number of ages in base.

N_met = None
Number of metallicities in base.

N_base = None
Number of elements in base.

N_x = None
Number of pixels in X direction.

N_y = None
Number of pixels in Y direction.

N_zone = None
Number of voronoi zones.

Nl_obs = None
Number of wavelengths.

x0 = None
X position of the galaxy center.

y0 = None
Y position of the galaxy center.

HLR_pix = None
Half light radius in pixels.

HLR_pc = None
Half light radius in parsecs.

distance_Mpc = None
Distance to galaxy in megaparsecs.

parsecPerPixel = None
Pixel scale in parsecs.

PIXSIZE = None
Pixel scale in arcseconds.

flux_unit = None
Half light radius in pixels.

q_norm = None
A_λ/A_V. Used in reddening calculations.

l_ini = None
Initial wavelength of spectra, in angstroms.

l_fin = None
Final wavelength of spectra, in angstroms.

dl = None
Wavelength step, in angstroms.

masterListData = None
Dictionary containing the masterlist data for this galaxy.

keywords = None
Dictionary containing all synthesis, base and qbick keywords. Keys are uppercase.

header = None
Alias for keywords (page 50). For backwards compatibility.

integrated_keywords = None
Dictionary containing additional synthesis data for integrated spectra. Keys are uppercase.

pa = None
Position angle of the ellipse used in radial profiles.

ba = None
Ellipticity b/a of the ellipse used in radial profiles.

pixelDistance__yx = None
Distance of each pixel to the galaxy center.

- Units: [pixel]
- Shape: `(N_y, N_x)`

pixelAngle__yx = None
Angle of each pixel relative to the semimajor axis, corrected for a disk perspective. See pa (page 51) and ba (page 51). Ranges from $-\pi$ to $+\pi$.

- Units: [radians]
- Shape: `(N_y, N_x)`

zoneArea_pix = None
Area in pixels of each zone.

- Units: [pixel]
- Shape: `(N_Zone)`

zoneArea_pc2 = None
Area in square parsecs of each zone.

- Units: $[pc^2]$
- Shape: `(N_y, N_x)`

fill_value = None
Fill value used when building images from zones.

loadGalaxy(*runId*, *califaId*, *smooth=True*)
Load a galaxy from a run, and update the metadata.

Parameters **runId** : string

Id of the synthesis run.

califaId [string] CALIFA identifier of the galaxy. Ex.: K0001.

TODO: test LoadGalaxy() :

ageBase
Ages of the base.

- Units: $[Yr]$
- Shape: `(N_zone)`

metBase
Metalicities of the base.

- Units: dimensionless
- Shape: `(N_zone)`

zonePos
A recarray (x, y) containing the position of the center of each zone.

- Units: pixel
- Shape: `(N_zone)`

qSignal
Signal at wavelength WINDOWSN.

- Units: $[erg/s/cm^2/\mathring{A}]$
- Shape: `(N_y, N_x)`

qNoise
TODO: Add description of cubes.

qSn
S/N at wavelength WINDOWSN.

- Units: dimensionless
- Shape: `(N_y, N_x)`

qSignalUnmasked
Image at wavelength WINDOWSN (quick, not masked).

- Units: $[erg/s/cm^2/\mathring{A}]$
- Shape: `(N_y, N_x)`

qNoiseUnmasked
TODO: Add description of qNoiseUnmasked.

qPipeNoise
TODO: Add description of qPipeNoise.

qZones
Voronoi/segmentation zones (bins).

- Units: index
- Shape: `(N_y, N_x)`

qZonesSn
S/N in Voronoi zones.

- Units: dimensionless
- Shape: `(N_y, N_x)`

qZonesNoise
Noise (RMS) in Voronoi zones.

- Units: $[erg/s/cm^2/\mathring{A}]$
- Shape: `(N_y, N_x)`

qFlagRatio
Ratio of flags in window at wavelength WINDOWSN

- Units: $[\%]$
- Shape: `(N_y, N_x)`

qZonesSnOrig
S/N in Voronoi zones (unresampled, beta).

- Units: dimensionless
- Shape: `(N_y, N_x)`

qZonesNoiseOrig
Noise (RMS) in Voronoi zones (unresampled, beta).

- Units: $[erg/s/cm^2/\overset{\circ}{A}]$
- Shape: `(N_y, N_x)`

qSegmentationSn
Voronoi program output S/N.

- Units: dimensionless
- Shape: `(N_y, N_x)`

qPipeNoiseOrig
Noise image derived from formal errors (unresampled).

- Units: $[erg/s/cm^2/\overset{\circ}{A}]$
- Shape: `(N_y, N_x)`

qPipeZonesNoiseOrig
Noise image derived from formal errors in Voronoi zones (unresampled).

- Units: $[erg/s/cm^2/\overset{\circ}{A}]$
- Shape: `(N_y, N_x)`

qSpatialMask
TODO: Add description of qSpatialMask.

qSnMask
TODO: Add description of qSnMask.

qFilledMask
TODO: Add description of qFilledMask.

qMask
Boolean image mask used for data.

- Units: bool
- Shape: `(N_y, N_x)`

fobs_norm
Flux in norm window for output spectra, in voronoi zones.

- Units: $[erg/s/cm^2/\overset{\circ}{A}]$
- Shape: `(Nl_obs, N_zone)`

f_obs
Observed flux (input spectra for the synthesis), in voronoi zones.

- Units: $[erg/s/cm^2/\overset{\circ}{A}]$
- Shape: `(Nl_obs, N_zone)`

f_err
Error in observed spetra, in voronoi zones.

- Units: $[erg/s/cm^2/\overset{\circ}{A}]$
- Shape: `(Nl_obs, N_zone)`

f_flag
Flagged spaxels, in voronoi zones.

FIXME: describe flags.

- Units: dimensionless
- Shape: `(Nl_obs, N_zone)`

integrated_f_obs
Observed flux (input spectra for the synthesis), in integrated spectrum.

- Units: $[erg/s/cm^2/\mathring{A}]$
- Shape: `(Nl_obs)`

integrated_f_err
Error in integrated observed spetrum.

- Units: $[erg/s/cm^2/\mathring{A}]$
- Shape: `(Nl_obs)`

integrated_f_flag
Flagged spaxels, in integrated spectrum.

FIXME: describe flags.

- Units: dimensionless
- Shape: `(Nl_obs)`

Mstars
Fraction of the initial stellar mass for a given population that is still trapped inside stars.

- Units: dimensionless
- Shape: `(N_age, N_met)`

fbase_norm
TODO: Add description of cubes.

popx
Light fractions for each population, in voronoi zones.

- –Units: $[\%]$
- Shape: `(N_age, N_met, N_zone)`

popmu_cor
Current mass fractions for each population, in voronoi zones.

- Units: $[\%]$
- Shape: `(N_age, N_met, N_zone)`

popmu_ini
Initial mass fractions for each population, in voronoi zones.

- Units: $[\%]$
- Shape: `(N_age, N_met, N_zone)`

popAV_tot
Extinction for each population, in voronoi zones.

- Units: $[mag]$
- Shape: `(N_age, N_met, N_zone)`

popexAV_flag
TODO: Add description of popexAV_flag.

SSP_chi2r
TODO: Add description of cubes.

SSP_adev
TODO: Add description of cubes.

SSP_AV
TODO: Add description of cubes.

SSP_x
TODO: Add description of cubes.

Lobs_norm
Luminosity density in norm window, in voronoi zones.

- Units: $[L_\odot/\mathring{A}]$
- Shape: `(N_zone)`

Mini_tot
Initial mass for each population, in voronoi zones.

- Units: $[M_\odot]$
- Shape: `(N_zone)`

Mcor_tot
Current mass for each population, in voronoi zones.

- Units: $[M_\odot]$
- Shape: `(N_zone)`

A_V
Extinction by dust, in voronoi zones.

- Units: $[mag]$
- Shape: `(N_zone)`

v_0
Velocity displacement, in voronoi zones.

- Units: $[km/s]$
- Shape: `(N_zone)`

v_d
Velocity dispersion, in voronoi zones.

- Units: $[km/s]$
- Shape: `(N_zone)`

adev
Mean absolute relative deviation, in percent, only for the `Nl_eff` points actually used in the synthesis.

- Units: $[\%]$
- Shape: `(N_zone)`

index_Best_SSP
Best single SSP fit, to use in `SSP_*`, in voronoi zones.

FIXME: index_Best_SSP is j or (age,met)?

- Units: index
- Shape: `(N_zone)`

NOl_eff
Number of OK wavelengths in input spectrum, in voronoi zones.

- Units: dimensionless
- Shape: `(N_zone)`

Nl_eff
Number of wavelengths actually used in spectral fit, in voronoi zones.

- Units: dimensionless
- Shape: `(N_zone)`

Ntot_clipped
Number of wavelengths clipped, in voronoi zones.

- Units: dimensionless
- Shape: `(N_zone)`

Nglobal_steps
Number of steps in spectral fitting, in voronoi zones.

- Units: dimensionless
- Shape: `(N_zone)`

chi2
χ^2/Nl_{eff} of the fit, in voronoi zones.

- Units: dimensionless
- Shape: `(N_zone)`

chi2_TOT
Total χ^2 of the fit, in voronoi zones.

- Units: dimensionless
- Shape: `(N_zone)`

chi2_Opt
χ^2 of the optical fit, in voronoi zones.

- Units: dimensionless
- Shape: `(N_zone)`

chi2_FIR
χ^2 of the far-IR fit, in voronoi zones.

- Units: dimensionless
- Shape: `(N_zone)`

chi2_QHR
χ^2 of the QH-related fit in voronoi zones.

- Units: dimensionless
- Shape: `(N_zone)`

chi2_PHO
χ^2 of the photometric fit, in voronoi zones.

- Units: dimensionless
- Shape: `(N_zone)`

f_syn
Synthetic spectra, in voronoi zones.

- Units: $[erg/s/cm^2/\mathring{A}]$
- Shape: `(Nl_obs, N_zone)`

f_wei
Weight of the spaxels in the input spectra. This is the weight actually used by the synthesis, after clipping, etc. Values for voronoi zones.

FIXME: describe flags and weights.

- Units: dimensionless
- Shape: `(Nl_obs, N_zone)`

chains_best_par
TODO: Add description of chains_best_par.

chains_ave_par
TODO: Add description of chains_ave_par.

chains_par
TODO: Add description of chains_par.

chains_best_LAx
TODO: Add description of chains_best_LAx.

chains_ave_LAx
TODO: Add description of chains_ave_LAx.

chains_LAx
TODO: Add description of chains_LAx.

chains_best_mu_cor
TODO: Add description of chains_best_mu_cor.

chains_ave_mu_cor
TODO: Add description of chains_ave_mu_cor.

chains_mu_cor
TODO: Add description of chains_mu_cor.

best_chi2
Best χ^2 (in chains), in voronoi zones.

- Units: dimensionless
- Shape: `(N_zone)`

ave_chi2
Average χ^2 (in chains), in voronoi zones.

- Units: dimensionless
- Shape: `(N_zone)`

cha_chi2
TODO: Add description of cubes.

best_Mcor
Best Mcor (in chains), in voronoi zones.

- Units: $[M_\odot]$
- Shape: `(N_zone)`

ave_Mcor
Average Mcor (in chains), in voronoi zones.

- Units: $[M_\odot]$
- Shape: `(N_zone)`

cha_Mcor
TODO: Add description of cha_Mcor.

integrated_popx
Light fractions for each population, in integrated spectrum.

- Units: $[\%]$
- Shape: `(N_age, N_met)`

integrated_popmu_cor
Current mass fractions for each population, in integrated spectrum.

- Units: $[\%]$
- Shape: `(N_age, N_met)`

integrated_popmu_ini
Current mass fractions for each population, in integrated spectrum.

- Units: $[\%]$
- Shape: `(N_age, N_met)`

integrated_popAV_tot
Extinction by dust for each population, in integrated spectrum.

- Units: $[mag]$
- Shape: `(N_age, N_met)`

integrated_popexAV_flag
TODO: Add description of popexAV_flag.

integrated_SSP_chi2r
TODO: Add description of integrated_SSP_chi2r.

integrated_SSP_adev
TODO: Add description of integrated_SSP_adev.

integrated_SSP_AV
TODO: Add description of integrated_SSP_AV.

integrated_SSP_x
TODO: Add description of integrated_SSP_x.

integrated_f_syn
Synthetic integrated spectrum.

- Units: $[erg/s/cm^2/\mathring{A}]$
- Shape: `(Nl_obs)`

integrated_f_wei
Weight of the spaxels in the integrated input spectra. This is the weight actually used by the synthesis, after clipping, etc.

FIXME: describe flags and weights.

- Units: dimensionless
- Shape: `(Nl_obs, N_y, N_x)`

integrated_chains_best_par
TODO: Add description of integrated_chains_best_par.

integrated_chains_ave_par
TODO: Add description of integrated_chains_ave_par.

integrated_chains_par
TODO: Add description of integrated_chains_par.

integrated_chains_best_LAx
TODO: Add description of integrated_chains_best_LAx.

integrated_chains_ave_LAx
TODO: Add description of integrated_chains_ave_LAx.

integrated_chains_LAx
TODO: Add description of integrated_chains_LAx.

integrated_chains_best_mu_cor
TODO: Add description of integrated_chains_best_mu_cor.

integrated_chains_ave_mu_cor
TODO: Add description of integrated_chains_ave_mu_cor.

integrated_chains_mu_cor
TODO: Add description of integrated_chains_mu_cor.

l_obs
Wavelength array for the spectral.

- Units: $[\mathring{A}]$
- Shape: `(Nl_obs)`

class pycasso.q3datacube_intf.**IQ3DataCube**
Abstract class for Q3 datacubes manipulation.

This class defines the high level operations on the data, such as conversion from zones to spatial coordinates and radial profiles.

Do not use this class directly, use one of the implementations instead.

See Also:

`h5Q3DataCube`, `fitsQ3DataCube`

loadGalaxy()
Abstract method used to load galaxy data if allowed by the underlying infrastructure.

setSmoothDezonification(*smooth=True*)
Enable or disable smooth dezonification. If smooth is True, use `qSignal` image to weight the pixels in each zone. Otherwise use the zone area.

Parameters **smooth** : boolean, optional

Enable or disable smooth dezonification. Defaults to `True`.

getDezonificationWeight(*smooth*, *prop=None*)
Create the weight image for dezonification. If smooth is True, use `prop` image to weight the pixels in each zone. Otherwise use the zone area. If `prop` is not set, use `qSignal`.

Here we use a scheme similar to `zoneToYX()` (page 59), when using smooth dezonification, except that we use `numpy.histogram()`[1] to calculate the weight of the pixels.

Parameters **smooth** : boolean

Enable or disable smooth dezonification.

prop : array, optional

Image to use as dezonification weights if `smooth` is `True`. If set to `None`, use `qSignal`.

zoneToYX(*prop*, *extensive=True*, *surface_density=True*, *fill_value=None*)
Convert a zone array to an image.

This scheme takes advantage of the qZones image, which has, for every pixel (x, y) the index of the corresponding zone. Using this array as a "smart index" for `prop`, we get to reconstruct the image.

Parameters **prop** : array

[1] http://docs.scipy.org/doc/numpy/reference/generated/numpy.histogram.html#numpy.histogram

Property to be converted to an image. The zone dimension must be the rightmost dimension.

extensive : boolean, optional

If `True`, `prop` is extensive, use dezonification weights. Defaults to `True`.

surface_density : boolean, optional

If `True`, and `extensive` is `True`, divide the return value by the area in parsec^2 of the pixel. Defaults to `True`.

fill_value : float, optional

Fill value for masked pixels. Defaults to `numpy.nan`.

Returns **prop__yx** : masked array

The `prop` array converted to image. All dimensions are kept the same, except for the rightmost one, which is replaced by y and x.

See Also:

`fillImage` (page 67), `getDezonificationWeight` (page 59), `setSmoothDezonification` (page 59)

Examples

Load the dataset:

```
>>> from pycasso.h5datacube import h5Q3DataCube
>>> K = h5Q3DataCube('qalifa-synthesis.002.h5', 'run001', 'K0277')
```

The `A_V` attribute contains the extincion for each zone.

```
>>> K.A_V.shape
(1638,)
```

Convert `A_V` to spatial coordinates. Note that the extinction is not extensive.

```
>>> A_V__yx = K.zoneToYX(K.A_V, extensive=False)
>>> A_V__yx.shape
(73, 77)
```

Plot the image.

```
>>> import matplotlib.pyplot as plt
>>> plt.imshow(A_V__yx)
```

setGeometry(*pa*, *ba*, *HLR_pix=None*, *center=None*)

Change the geometry of the rings used when calculating radial profiles.

Parameters **pa** : float

Position angle in radians, counter-clockwise relative to the positive X axis.

ba : float

Ellipticity, defined as the ratio between the semiminor axis and the semimajor axis (b/a).

HLR_pix : float, optional

Effective radius

center : (float, float), optional

A tuple containing the x and y coodinates of the center of the galaxy, in pixels.

See Also:

getEllipseParams (page 68), getPixelDistance (page 68)

Examples

Load the dataset:

```
>>> from pycasso.h5datacube import h5Q3DataCube
>>> K = h5Q3DataCube('qalifa-synthesis.002.h5', 'run001', 'K0277')
```

Find the ellipse parameters.

```
>>> pa, ba = K.getEllipseParams()
```

Set the geometry, using a predefined value for HLR_pix.

```
>>> K.setGeometry(pa, ba, HLR_pix=10.5)
```

Get the distance for each pixel from the galaxy center.

```
>>> dist__yx = K.getPixelDistance()
```

Plot the distance image. Note that its shape should resemble the ellipticity of the galaxy (if it is well-behaved).

```
>>> import matplotlib.pyplot as plt
>>> plt.imshow(dist__yx)
```

getYXToRadialBinsTensorExact (*bin_r*, *rad_scale=None*, *mask=None*)

Generate an operator for calculating the radial profile using exact elliptic apertures. See the examples below for the usage.

Parameters **bin_r** : array

Semimajor axis bin boundaries in units of rad_scale.

rad_scale : float, optional

Scale of the bins, in pixels. Defaults to HLR_pix.

mask : array, optional

Mask containing the pixels to use in the radial profile. Must have the shape (N_y, N_x). Defaults to qMask.

Returns **ryx** : masked array

Operator for calculating the radial profile.

area_pix : masked array

The number of points inside each bin.

Examples

Load the dataset:

```
>>> from pycasso.h5datacube import h5Q3DataCube
>>> K = h5Q3DataCube('qalifa-synthesis.002.h5', 'run001', 'K0277')
```

Create the bins from 0.0 to 3.0 in 0.1 steps, using rad_scale units.

```
>>> import numpy as np
>>> bin_r = np.arange(0.0, 3.0 + 0.1, 0.1)
```

Create the radial profile operator.

```
>>> ryx, area = K.getYXToRadialBinsTensorExact(bin_r)
```

Calculate the radial profile for some properties. Note the `np.tensordot()` index convention. In general, one has to match the XY indices in both the `ryx` array (1, 2) and the property (the last two indices). The order of the arguments is important.

```
>>>
```

radialProfileExact (*prop*, *bin_r*, *rad_scale=None*, *mask=None*, *return_npts=False*)

Calculate the radial profile of a property, using exact elliptic apertures. This is suited for a one-shot radial profile. See `getYXToRadialBinsTensorExact()` (page 61) for a more efficient approach.

Parameters **prop** : array

Image of property to calculate the radial profile. The last two dimensions of `prop` must be of length `N_y` and `N_x`.

bin_r : array

Semimajor axis bin boundaries in units of `rad_scale`.

rad_scale : float, optional

Scale of the bins, in pixels. Defaults to `HLR_pix`.

mask : array, optional

Mask containing the pixels to use in the radial profile. Must have the shape `(N_y, N_x)`. Defaults to `qMask`.

return_npts : bool, optional

If set to `True`, also return the number of points inside each bin. Defaults to `False`.

Returns **radProf** : masked array

Array containing the radial profile as the last dimension. Note that `radProf.shape[-1] == (len(bin_r) - 1)`

npts : masked array, optional

The number of points inside each bin, only if `return_npts` is set to `True`.

See Also:

`azimuthalProfileNd` (page 66), `radialProfile` (page 63), `zoneToRad` (page 65), `zoneToYX` (page 59), `getYXToRadialBinsTensorExact` (page 61)

Examples

Load the dataset:

```
>>> from pycasso.h5datacube import h5Q3DataCube
>>> K = h5Q3DataCube('qalifa-synthesis.002.h5', 'run001', 'K0277')
```

Create the bins from 0.0 to 3.0 in 0.1 steps, using `rad_scale` units.

```
>>> import numpy as np
>>> bin_r = np.arange(0.0, 3.0 + 0.1, 0.1)
```

Calculate the radial profile of the time resolved mass surface density, using a radial scale of 10.5 pixels.

```
>>> McorSD__tyx = K.McorSD__tZyx.sum(axis=1)
>>> McorSD__tr = K.radialProfileExact(McorSD__tyx, bin_r, rad_scale=10.5)
```

Plot the Radial profile.

```
>>> import matplotlib.pyplot as plt
>>> plt.pcolormesh(np.log10(K.ageBase), bin_r, McorSD__r)
```

radialProfile(*prop*, *bin_r*, *rad_scale=None*, *mask=None*, *r__yx=None*, *mode='mean'*, *return_npts=False*)

Calculate the radial profile of a property. The last two dimensions of `prop` must be of length `N_y` and `N_x`.

Parameters **prop** : array

Image of property to calculate the radial profile.

bin_r : array

Semimajor axis bin boundaries in units of `rad_scale`.

rad_scale : float, optional

Scale of the bins, in pixels. Defaults to `HLR_pix`.

mask : array, optional

Mask containing the pixels to use in the radial profile. Must have the shape `(N_y, N_x)`. Defaults to `qMask`.

r__yx : array, optional

Distance of each pixel to the galaxy center, in pixels. Must have the shape `(N_y, N_x)`. Defaults to `pixelDistance__yx`. Not used when `mode='mean_exact'`.

mode : {'mean', mean_exact', 'median', 'sum'}, optional

The operation to perform inside the bin. Default is `'mean'`. The mode 'mean_exact' computes the intersection of an ellipse and the pixels, this avoids pixelated bins but is very slow.

return_npts : bool, optional

If set to `True`, also return the number of points inside each bin. Defaults to `False`.

Returns **radProf** : array

Array containing the radial profile as the last dimension. Note that `radProf.shape[-1] == (len(bin_r) - 1)`

npts : array, optional

The number of points inside each bin, only if `return_npts` is set to `True`.

See Also:

`azimuthalProfileNd` (page 66), `radialProfile` (page 63), `zoneToRad` (page 65), `zoneToYX` (page 59)

Examples

Load the dataset:

```
>>> from pycasso.h5datacube import h5Q3DataCube
>>> K = h5Q3DataCube('qalifa-synthesis.002.h5', 'run001', 'K0277')
```

Create the bins from 0.0 to 3.0 in 0.1 steps, using `rad_scale` units.

```
>>> import numpy as np
>>> bin_r = np.arange(0.0, 3.0 + 0.1, 0.1)
```

Calculate the radial profile of the time resolved mass surface density, using a radial scale of 10.5 pixels.

```
>>> McorSD__tyx = K.McorSD__tZyx.sum(axis=1)
>>> McorSD__tZr = K.radialProfile(McorSD__tyx, bin_r, rad_scale=10.5)
```

Plot the Radial profile.

```
>>> import matplotlib.pyplot as plt
>>> plt.pcolormesh(np.log10(K.ageBase), bin_r, McorSD__r)
```

radialProfileNd(*prop*, *bin_r*, *rad_scale=None*, *mask=None*, *r__yx=None*, *mode='mean'*, *return_npts=False*)

Calculate the radial profile of a property. The last two dimensions of `prop` must be of length `N_y` and `N_x`.

Parameters **prop** : array

Image of property to calculate the radial profile.

bin_r : array

Semimajor axis bin boundaries in units of `rad_scale`.

rad_scale : float, optional

Scale of the bins, in pixels. Defaults to `HLR_pix`.

mask : array, optional

Mask containing the pixels to use in the radial profile. Must have the shape `(N_y, N_x)`. Defaults to `qMask`.

r__yx : array, optional

Distance of each pixel to the galaxy center, in pixels. Must have the shape `(N_y, N_x)`. Defaults to `pixelDistance__yx`.

mode : {'mean', 'median', 'sum'}, optional

The operation to perform inside the bin. Default is `'mean'`.

return_npts : bool, optional

If set to `True`, also return the number of points inside each bin. Defaults to `False`.

Returns **radProf** : [masked] array

Array containing the radial profile as the last dimension. Note that `radProf.shape[-1] == (len(bin_r) - 1)`. If `prop` is a masked aray, this and `npts` will be a masked array as well.

npts : [masked] array, optional

The number of points inside each bin, only if `return_npts` is set to `True`.

See Also:

`azimuthalProfileNd` (page 66), `radialProfile` (page 63), `zoneToRad` (page 65), `zoneToYX` (page 59)

Examples

Load the dataset:

```
>>> from pycasso.h5datacube import h5Q3DataCube
>>> K = h5Q3DataCube('qalifa-synthesis.002.h5', 'run001', 'K0277')
```

Create the bins from 0.0 to 3.0 in 0.1 steps, using `rad_scale` units.

```
>>> import numpy as np
>>> bin_r = np.arange(0.0, 3.0 + 0.1, 0.1)
```

Calculate the radial profile of the time resolved mass surface density, using a radial scale of 10.5 pixels.

```
>>> McorSD__tyx = K.McorSD__tZyx.sum(axis=1)
>>> McorSD__tZr = K.radialProfileNd(McorSD__tyx, bin_r, rad_scale=10.5)
```

Plot the Radial profile.

```
>>> import matplotlib.pyplot as plt
>>> plt.pcolormesh(np.log10(K.ageBase), bin_r, McorSD__r)
```

zoneToRad(*prop*, *bin_r*, *rad_scale=None*, *mask=None*, *r__yx=None*, *mode='mean'*, *extensive=True*, *surface_density=True*)

Calculates the radial profile from a zone array. This method is a wrapper for `zoneToYX()` (page 59) and `radialProfile()` (page 63).

Parameters **prop** : array

Image of property to calculate the radial profile.

bin_r : array

Semimajor axis bin boundaries in units of `rad_scale`.

rad_scale : float, optional

Scale of the bins, in pixels. Defaults to `HLR_pix`.

mask : array, optional

Mask containing the pixels to use in the radial profile. Must have the same dimensions as `prop`. Defaults to `qMask`.

r__yx : array, optional

Distance of each pixel to the galaxy center, in pixels. Must have the same dimensions as `prop`. Defaults to `pixelDistance__yx`.

mode : {'mean', mean_exact', 'median', 'sum'}, optional

The operation to perform inside the bin. Default is `'mean'`. The mode 'mean_exact' computes the intersection of an ellipse and the pixels, this avoids pixelated bins but is very slow.

extensive : boolean, optional

If `True`, `prop` is extensive, use dezonification weights. Defaults to `True`.

surface_density : boolean, optional

If `True`, and `extensive` is `True`, divide the return value by the area in parsec^2 of the pixel. Defaults to `True`.

Returns **radProf** : array

Array containing the radial profile. Note that `len(radProf) == (len(bin_r) - 1)`

See Also:

`radialProfile` (page 63), `radialProfileNd` (page 64), `zoneToYX` (page 59)

Examples

Load the dataset:

```
>>> from pycasso.h5datacube import h5Q3DataCube
>>> K = h5Q3DataCube('qalifa-synthesis.002.h5', 'run001', 'K0277')
```

Create the bins from 0.0 to 3.0 in 0.1 steps, using `rad_scale` units.

```
>>> import numpy as np
>>> bin_r = np.arange(0.0, 3.0 + 0.1, 0.1)
```

Calculate the radial profile of mass surface density, using a radial scale of 10.5 pixels. The `extensive` option is the key for "surface density".

```
>>> McorSD__r = K.zoneToRad(K.Mcor__z, bin_r, rad_scale=10.5, extensive=True)
```

Note that `bin_r` is the bin boundaries, it is not fit for plotting along with `McorSD__r`. We need the bin centers.

```
>>> bin_center = (bin_r[:-1] + bin_r[1:]) / 2.0
```

Plot the Radial profile.

```
>>> import matplotlib.pyplot as plt
>>> plt.plot(bin_center, McorSD__r)
```

azimuthalProfileNd(*prop*, *bin_a*, *bin_r*, *rad_scale=None*, *mask=None*, *a__yx=None*, *r__yx=None*, *mode='mean'*, *return_npts=False*)

Calculate the azimuthal profile of a property. The last two dimensions of `prop` must be of legth `N_y` and `N_x`.

Parameters **prop** : array

Image of property to calculate the radial profile.

bin_a : array

Angular bin boundaries in radians.

bin_r : array

Semimajor axis bin boundaries in units of `rad_scale`.

rad_scale : float, optional

Scale of the bins, in pixels. Defaults to `HLR_pix`.

mask : array, optional

Mask containing the pixels to use in the radial profile. Must have the shape `(N_y, N_x)`. Defaults to `qMask`.

r__yx : array, optional

Distance of each pixel to the galaxy center, in pixels. Must have the shape `(N_y, N_x)`. Defaults to `pixelDistance__yx`.

a__yx : array, optional

Angle associated with each pixel. Must have the shape `(N_y, N_x)`. Defaults to `pixelAngle__yx`.

mode : {'mean', 'median', 'sum'}, optional

The operation to perform inside the bin. Default is `'mean'`.

return_npts : bool, optional

If set to `True`, also return the number of points inside each bin. Defaults to `False`.

Returns **azProf** : array

Array containing the radial and azimuthal profiles as the last dimensions. Note that `radProf.shape[-2] == (len(bin_a) - 1)` and `radProf.shape[-1] == (len(bin_r) - 1)`.

npts : array, optional

The number of points inside each bin, only if `return_npts` is set to `True`.

See Also:

`radialProfileNd` (page 64), `radialProfile` (page 63), `zoneToRad` (page 65), `zoneToYX` (page 59)

Examples

Load the dataset:

```
>>> from pycasso.h5datacube import h5Q3DataCube
>>> K = h5Q3DataCube('qalifa-synthesis.002.h5', 'run001', 'K0277')
```

Create the radial bins from 0.0 to 3.0 in 0.5 steps, using `rad_scale` units, and the angular bins as 21 boundaries between -180 and 180 degrees, converted to radians.

```
>>> import numpy as np
>>> bin_r = np.arange(0.0, 3.0 + 0.5, 0.5)
>>> bin_a = np.linspace(-180.0, 180.0, 21) / 180.0 * np.pi
```

Calculate the azimuthal profile of the time resolved mass surface density, using a radial scale of 10.5 pixels.

```
>>> McorSD__tyx = K.McorSD__tZyx.sum(axis=1)
>>> McorSD__taR = K.azimuthalProfileNd(McorSD__tyx, bin_a, bin_r, rad_scale=10.5)
```

Note that `bin_a` contains the bin boundaries, it is not fit for plotting along with `McorSD__tZaR`. We will use the bin centers.

```
>>> bin_a_center = (bin_a[:-1] + bin_a[1:]) / 2.0
```

Plot the azimuthal profile for the first and last radial bins, summing in all ages. Note that McorSD__taR.shape is `(N_ages, len(bin_a) - 1, len(bin_r) - 1)`.

```
>>> import matplotlib.pyplot as plt
>>> plt.plot(bin_a_center, McorSD__taR[...,0].sum(axis=0)
>>> plt.plot(bin_a_center, McorSD__taR[...,-1].sum(axis=0)
```

fillImage(*prop*, *prop__r=None*, *r=None*, *r__yx=None*, *mode='convex'*)

Fill a 2-D the masked pixels of an image of a property using the values of a radial profile. The pixels to fill are chosen by `mode`.

Parameters **prop** : array

Property (2-D) to fill the hollow pixels.

prop__r : the radial profile to use to fill the data.

If not set, calculate from prop.

r : array

The radial distances for `prop__r` values. If not set (or `prop__r` is not set), use a 1 pixel step.

r__yx : array

A 2-D image containing the geometry of the image. If not set, use `pixelDistance__yx`.

mode : {'convex', 'hollow'}, string

If mode is `'convex'`, fill entire convex hull. If mode is `'hollow'`, fill only hollow pixels. Default is `'convex'`.

Returns **prop_fill** : array

A 2-D image of `prop`, with the missing pixels filled.

mask : array

The effective mask for the filled image.

getEllipseParams(*prop=None*, *mask=None*)

Estimate ellipticity and orientation of the galaxy using the "Stokes parameters", as described in: http://adsabs.harvard.edu/abs/2002AJ....123..485S The image used is `qSignal`.

Parameters **prop** : array, optional

Image to use when calculating the ellipse parameters. If not set, `qSignal` will be used.

mask : array, optional

Mask containing the pixels to take into account. If not set, `qMask` will be used.

Returns **pa** : float

Position angle in radians, counter-clockwise relative to the positive X axis.

ba : float

Ellipticity, defined as the ratio between the semiminor axis and the semimajor axis (b/a).

getPixelDistance(*use_HLR_units=True*, *pixel_scale=None*, *x=None*, *y=None*, *pa=None*, *ba=None*)

Return an image (`numpy.ndarray`[2] of same shape as :attr'qSignal') of the distance from the center of the galaxy `(x0, y0)` in HLR units (default), assuming a projected disk.

Parameters **use_HLR_units** : boolean, optional

Whether to use units of half light radius or pixels.

pixel_scale : float, optional

Pixel distance scale, used if `use_HLR_units` is `False`. If not set, do not scale the distance.

x : array, optional

X coordinates to calculate the distance. If not set, the coordinates of the core images will be used.

y : array, optional

Y coordinates to calculate the distance. Must have the same length as `x`. If not set, the coordinates of the core images will be used.

pa : float, optional

Position angle in radians, counter-clockwise relative to the positive X axis.

ba : float, optional

Ellipticity, defined as the ratio between the semiminor axis and the semimajor axis (b/a).

Returns **pixelDistance** : array

Array (or image) containing the pixel distances.

[2] http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray

See Also:

`getPixelAngle` (page 69)

getPixelAngle(*units='radians'*, *x=None*, *y=None*, *pa=None*, *ba=None*)

Return an image (`numpy.ndarray`[3] of same shape as :attr'qSignal') of the angle in radians (default) of each pixel, relative from the axis of the position angle `pa`. The projection is fixed assuming the galaxy is a disk, throught the ellipticity parameter `ba`.

Parameters **units** : {'radians', 'degrees'}, optional

If `'radians'`, angles are in radians, from $-\pi$ to $+\pi$ (default). If `'degrees'`, angles are in degrees, from -180.0 to $+180.0$.

x : array, optional

X coordinates to calculate the distance. If not set, the coordinates of the core images will be used.

y : array, optional

Y coordinates to calculate the distance. Must have the same length as `x`. If not set, the coordinates of the core images will be used.

pa : float, optional

Position angle in radians, counter-clockwise relative to the positive X axis.

ba : float, optional

Ellipticity, defined as the ratio between the semiminor axis and the semimajor axis (b/a).

Returns **pixelDistance** : 2-D array

Image containing the pixel distances.

getHalfRadius(*prop*, *fill=False*, *mask=None*)

Find the half radius of the desired property. Using radial bins of 1 pixel, calculate the cumulative sum of `prop`. The "half prop radius" is the radius where the cumulative sum reaches 50% of its peak value.

Parameters **prop** : array

Image to get the half radius.

fill : boolean, optional

Whether to fill the hollow areas before the calculations. The filling is done using the average value in the respective radial distance of the missing pixels. Default is `False`.

mask : array(bool), optional

Boolean array with the valid data. Defaults to `qMask`.

Returns **HXR** : float

The half `prop` radius, in pixels.

See Also:

`setGeometry` (page 60), `radialProfile` (page 63)

Notes

This value should be close to $dfrac{HLR_{circular}}{sqrt{b/a}}$ if the isophotes of the galaxy are all ellipses with parameters p.a. and b/a.

[3] http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray

Examples

Load the dataset:

```
>>> from pycasso.h5datacube import h5Q3DataCube
>>> K = h5Q3DataCube('qalifa-synthesis.002.h5', 'run001', 'K0277')
```

Calculate the half mass radius, in pixels.

```
>>> HMR_pix = K.getHalfRadius(K.McorSD)
```

Calculate the radial profile of extinction by dust, using `HMR_pix` as radial scale, in bins from 0.0 to 3.0, in 0.1 steps.

```
>>> import numpy as np
>>> bin_r = np.arange(0.0, 3.0 + 0.1, 0.1)
>>> A_V__r = K.radialProfile(K.A_V__yx, bin_r, rad_scale=HMR_pix)
```

Note that `bin_r` is the bin boundaries, it is not fit for plotting along with `A_V__r`. We need the bin centers.

```
>>> bin_center = bin_center = (bin_r[:-1] + bin_r[1:]) / 2.0
```

Plot the Radial profile.

```
>>> import matplotlib.pyplot as plt
>>> plt.plot(bin_center, A_V__r)
```

getHLR_pix (*pa=None*, *ba=None*, *fill=False*)

Find the half light radius using the image at the normalization window (`qSignal`). Using radial bins of 1 pixel, calculate the cumulative sum of luminosity. The HLR is the radius where the cum. sum reaches 50% of its peak value.

Parameters **fill** : boolean

Whether to fill the hollow areas before calculating the HLR.

Returns **HLR** : float

The half light radius, in pixels.

See Also:

`getHalfRadius` (page 69), `setGeometry` (page 60), `fillImage` (page 67)

Notes

This value should be close to $dfrac{HLR_{circular}}{sqrt{b/a}}$ if the isophotes of the galaxy are all ellipses with parameters p.a. and b/a.

growthInAge (*prop*, *mask=None*, *relative=False*)

TODO: move to utils?

Calculate the cumulative growth in age space. In age = 0, we have the sum of `prop` in age, and as age increases the sum of `prop` decreases.

Parameters **prop** : array

A property having the age as first dimension.

mask : array

Boolean mask for all dimensions except the first (age). If set, the values will be summed, that is, the return array will only have the age dimension.

relative : bool

If True, normalize by the maximum value of the cumulative sum of `prop`.

Returns **cs** : array

The growth in age space of `prop`.

Examples

Load the dataset:

```
>>> from pycasso.h5datacube import h5Q3DataCube
>>> K = h5Q3DataCube('qalifa-synthesis.002.h5', 'run001', 'K0277')
```

Get the spatially resolved mass surface density for each age.

```
>>> MiniSD__tyx = K.MiniSD__tZyx.sum(axis=1)
```

Calculate the mass build up for the whole galaxy (notice the "all data" mask, `K.qMask`). The mask could be any selection of data pixels.

```
>>> MtotGrow__t = K.growthInAge(MiniSD__tyx, mask=K.qMask)
```

`MtotGrow__t` is a surface mass density, in $M_\odot/pc^2$. Converting to absolute mass.

```
>>> MtotGrow__t *= K.parsecPerPixel ** 2
```

Plot the mass buildup as a function of log. time.

```
>>> ages = np.log10(K.ageBase)
>>> import matplotlib.pyplot as plt
>>> plt.plot(ages, MtotGrow__t)
```

getSFR(*logtc_ini=None*, *logtc_fin=None*, *logtc_step=0.05*, *logtc_FWHM=0.5*)
Calculate the star formation rate using a smooth-resampled age base, as prescribed by Asari et al. (2007) <http://adsabs.harvard.edu/abs/2007MNRAS.381..263A>.

Parameters **logtc_ini** : float

Logarithm (base 10) of initial age. Defaults to `logtb.min()`.

logtc_fin : float

Logarithm (base 10) of final age. Defaults to `logtb.max()`.

logtc_step : float

Logarithm (base 10) of age step. Defaults to `0.05`.

logtc_FWHM : float

Width of the age smoothing kernel. Defaults to `0.5`.

Returns **SFR** : array

Star formation rate.

logtc : array

Logarithm (base 10) of smooth-resampled age base.

See Also:

`pystarlight.util.StarlightUtils.calcSFR`

area_qMask
Area, in pixels, of the data regions. This is the area to use when dealing with `qMask`.

area_qHollowPixels
Aarea, in pixels, of the masked regions. This is the area to use when dealing with `qHollowPixels` (page 72).

area_qConvexHull
Area, in pixels, of the convex hull of `qMask`. This is the area to use when dealing with `qConvexHull` (page 72).

qHollowPixels
Masked (bad) pixels inside the data mask.

- Units: bool
- Shape: `(N_y, N_x)`

qConvexHull
Convex hull of data mask.

- Units: bool
- Shape: `(N_y, N_x)`

popx__tZyx
Spatially resolved light fractions for each population.

- Units: $[\%]$
- Shape: `(N_age, N_met, N_y, N_x)`

popmu_cor__tZyx
Spatially resolved corrected mass fractions for each population.

- Units: $[\%]$
- Shape: `(N_age, N_met, N_y, N_x)`

popmu_ini__tZyx
Spatially resolved initial mass fractions for each population.

- Units: $[\%]$
- Shape: `(N_age, N_met, N_y, N_x)`

popAV_tot__tZyx
Spatially resolved extinction by dust for each population.

- Units: $[mag]$
- Shape: `(N_age, N_met, N_y, N_x)`

A_V__yx
Spatially resolved extinction by dust.

- Units: $[mag]$
- Shape: `(N_y, N_x)`

v_0__yx
Spatially resolved velocity displacement.

- Units: $[km/s]$
- Shape: `(N_y, N_x)`

v_d__yx
Spatially resolved velocity dispersion.

- Units: $[km/s]$
- Shape: `(N_y, N_x)`

integrated_Lobn__tZ
Luminosity of each population in normalization window of the integrated spectrum.

- Units: $[L_\odot]$
- Shape: `(N_age, N_met)`

integrated_Lobn
Luminosity in normalization window of the integrated spectrum.

•Units: $[L_\odot]$

•Type: float

Lobn__tZz
Luminosity of each population in normalization window, in voronoi zones.

•Units: $[L_\odot]$

•Shape: `(N_age, N_met, N_zone)`

Lobn__z
Luminosity in normalization window, in voronoi zones.

•Units: $[L_\odot]$

•Shape: `(N_zone)`

LobnSD__tZyx
Spatially resolved luminosity surface density of each population in normalization window.

•Units: $[L_\odot/pc^2]$

•Shape: `(N_age, N_met, N_y, N_x)`

LobnSD__yx
Luminosity surface density of each population in normalization window.

•Units: $[L_\odot/pc^2]$

•Shape: `(N_y, N_x)`

integrated_DeRed_Lobn__tZ

"Dereddened" luminosity of each population in normalization window of the integrated spectrum.

- Units: $[L_\odot]$
- Shape: `(N_age, N_met)`

integrated_DeRed_Lobn
"Dereddened" luminosity in normalization window of the integrated spectrum.

•Units: $[L_\odot]$

•Type: float

DeRed_Lobn__tZz
"Dereddened" luminosity of each population in normalization window, in voronoi zones.

•Units: $[L_\odot]$

•Shape: `(N_age, N_met, N_zone)`

DeRed_Lobn__z
"Dereddened" luminosity in normalization window, in voronoi zones.

•Units: $[L_\odot]$

•Shape: `(N_zone)`

DeRed_LobnSD__tZyx
Spatially resolved "dereddened" luminosity surface density of each population in normalization window.

•Units: $[L_\odot/pc^2]$

•Shape: `(N_age, N_met, N_y, N_x)`

DeRed_LobnSD__yx
"Dereddened" luminosity surface density of each population in normalization window.

•Units: $[L_\odot/pc^2]$

•Shape: `(N_y, N_x)`

integrated_Mcor__tZ
Current mass of each population of the integrated spectrum.

•Units: $[M_\odot]$

•Shape: `(N_age, N_met)`

integrated_Mcor
Current mass of the integrated spectrum.

•Units: $[M_\odot]$

•Type: float

Mcor__tZz
Current mass of each population, in voronoi zones.

•Units: $[M_\odot]$

•Shape: `(N_age, N_met, N_zone)`

Mcor__z
Current mass, in voronoi zones.

•Units: $[M_\odot]$

•Shape: `(N_age, N_met, N_zone)`

McorSD__tZyx
Spatially resolved current mass surface density of each population.

•Units: $[M_\odot/pc^2]$

•Shape: `(N_age, N_met, N_y, N_x)`

McorSD__yx
Spatially resolved current mass surface density.

•Units: $[M_\odot/pc^2]$

•Shape: `(N_y, N_x)`

integrated_Mini__tZ
Initial mass of each population of the integrated spectrum.

•Units: $[M_\odot]$

•Shape: `(N_age, N_met)`

integrated_Mini
Initial mass of the integrated spectrum.

•Units: $[M_\odot]$

•Type: float

Mini__tZz
Initial mass of each population, in voronoi zones.

•Units: $[M_\odot]$

•Shape: `(N_age, N_met, N_zone)`

Mini__z
Initial mass, in voronoi zones.

•Units: $[M_\odot]$

- Shape: `(N_age, N_met, N_zone)`

MiniSD__tZyx
Spatially resolved initial mass surface density of each population.

- Units: $[M_\odot/pc^2]$
- Shape: `(N_age, N_met, N_y, N_x)`

MiniSD__yx
Spatially resolved initial mass surface density.

- Units: $[M_\odot/pc^2]$
- Shape: `(N_y, N_x)`

integrated_M2L
Mass to light ratio of the integrated spectrum.

- Units: $[M_\odot/L_\odot]$
- Type: float

M2L__yx
Spatially resolved mass to light ratio.

- Units: $[M_\odot/L_\odot]$
- Shape: `(N_y, N_x)`

integrated_DeRed_M2L
"Dereddened" mass to light ratio of the integrated spectrum.

- Units: $[M_\odot/L_\odot]$
- Type: float

DeRed_M2L__yx
Spatially resolved "dereddened" mass to light ratio.

- Units: $[M_\odot/L_\odot]$
- Shape: `(N_y, N_x)`

integrated_at_flux
Flux-weighted average log. age of the integrated spectrum.

- Units: $[\log Gyr]$
- Type: float

at_flux__z
Flux-weighted average log. age, in voronoi zones.

- Units: $[\log Gyr]$
- Shape: `(N_zone)`

at_flux__yx
Spatially resolved, flux-weighted average log. age.

- Units: $[\log Gyr]$
- Shape: `(N_y, N_x)`

integrated_at_mass
Mass-weighted average log. age of the integrated spectrum.

- Units: $[\log Gyr]$
- Type: float

at_mass__z
Mass-weighted average log. age, in voronoi zones.

- Units: $[\log Gyr]$
- Shape: `(N_zone)`

at_mass__yx
Spatially resolved, mass-weighted average log. age.

- Units: $[\log Gyr]$
- Shape: `(N_y, N_x)`

integrated_aZ_flux
Flux-weighted average metallicity of the integrated spectrum.

- Units: dimensionless
- Type: float

aZ_flux__z
Flux-weighted average metallicity, in voronoi zones.

- Units: dimensionless
- Shape: `(N_zone)`

aZ_flux__yx
Spatially resolved, flux-weighted average metallicity.

- Units: dimensionless
- Shape: `(N_y, N_x)`

integrated_aZ_mass
Mass-weighted average metallicity of the integrated spectrum.

- Units: dimensionless
- Type: float

aZ_mass__z
Mass-weighted average metallicity, in voronoi zones.

- Units: dimensionless
- Shape: `(N_zone)`

aZ_mass__yx
Spatially resolved, mass-weighted average metallicity.

- Units: dimensionless
- Shape: `(N_y, N_x)`

adev__yx
Mean absolute relative deviation, in percent, only for the `Nl_eff` points actually used in the synthesis.

- Units: $[\%]$
- Shape: `(N_y, N_x)`

chi2__yx
χ^2/Nl_{eff} of the fit.

- Units: dimensionless
- Shape: `(N_y, N_x)`

f_obs__lyx
Spatially resolved observed flux (input spectra for the synthesis).

- Units: $[erg/s/cm^2/\mathring{A}]$
- Shape: `(Nl_obs, N_y, N_x)`

f_flag__lyx
Spatially resolved flagged spaxels.

FIXME: describe flags.

•Units: dimensionless

•Shape: `(Nl_obs, N_y, N_x)`

f_syn__lyx
Spatially resolved synthetic spectra.

•Units: $[erg/s/cm^2/\mathring{A}]$

•Shape: `(Nl_obs, N_y, N_x)`

f_wei__lyx
Spatially resolved weight of the spaxels in the input spectra. This is the weight actually used by the synthesis, after clipping, etc.

FIXME: describe flags and weights.

•Units: dimensionless

•Shape: `(Nl_obs, N_y, N_x)`

f_err__lyx
Spatially resolved error in observed spetra.

•Units: $[erg/s/cm^2/\mathring{A}]$

•Shape: `(Nl_obs, N_y, N_x)`

logAgeBaseBins
Age bin edges, in log. Computes the bin edges as the bissection of the bins, expanding the borders accordingly.

•Units: $[\log age/yr]$

•Shape: `(N_age + 1)`

ageBaseMx(*n=2*)
Return `ageBase` as a row matrix to simplify multiplications.

Parameters **n** : integer

Dimension of the matrix.

Returns **ageBaseNd** : n-D array

`ageBase` as a row matrix.

See Also:

`metBaseMx` (page 77), `zoneAreaMx` (page 77)

metBaseMx(*n=2*)
Return `metBase` as a row matrix to simplify multiplications.

Parameters **n** : integer

Dimension of the matrix.

Returns **metBaseNd** : n-D array

`metBase` as a row matrix.

See Also:

`ageBaseMx` (page 77), `zoneAreaMx` (page 77)

zoneAreaMx(*n*)
Return `zoneArea_pc2` as a row matrix to simplify multiplications.

Parameters **n** : integer

Dimension of the matrix.

Returns **zoneAreaNd** : n-D array

zone area as a row matrix.

See Also:

ageBaseMx (page 77), metBaseMx (page 77)

adevS

From Cid @ 26/05/2012: Here's my request for pycasso reader: That it defines a new figure of merit analogous to adev, but which uses the synthetic flux in the denominator instead of the observed one. This is the adevS__z thing defined below in awful python. Why? Well, despite all our care there are still some non-flagged pixels with very low fluxes, which screws up adev, and this alternative definition fixes it.

Original code:

```
>>> adevS__z = np.zeros((self.nZones))
>>> for i_z in np.arange(self.nZones):
>>>     _a = np.abs(self.f_obs[:,i_z] - self.f_syn[:,i_z]) / self.f_syn[:,i_z]
>>>     _b = _a[self.f_wei[:,i_z] > 0]
>>>     adevS__z[i_z] = 100.0 * _b.mean()
>>> return adevS__z
```

Returns **aDevS** : array of length (nZones)

INDEX

A

A_V (pycasso.h5datacube.h5Q3DataCube attribute), 55
A_V__yx (pycasso.q3datacube_intf.IQ3DataCube attribute), 72
adev (pycasso.h5datacube.h5Q3DataCube attribute), 55
adev__yx (pycasso.q3datacube_intf.IQ3DataCube attribute), 76
adevS (pycasso.q3datacube_intf.IQ3DataCube attribute), 78
ageBase (pycasso.h5datacube.h5Q3DataCube attribute), 51
ageBaseMx() (pycasso.q3datacube_intf.IQ3DataCube method), 77
area_qConvexHull (pycasso.q3datacube_intf.IQ3DataCube attribute), 72
area_qHollowPixels (pycasso.q3datacube_intf.IQ3DataCube attribute), 71
area_qMask (pycasso.q3datacube_intf.IQ3DataCube attribute), 71
arrayAsRowMatrix() (in module pycasso.util), 43
at_flux__yx (pycasso.q3datacube_intf.IQ3DataCube attribute), 75
at_flux__z (pycasso.q3datacube_intf.IQ3DataCube attribute), 75
at_mass__yx (pycasso.q3datacube_intf.IQ3DataCube attribute), 76
at_mass__z (pycasso.q3datacube_intf.IQ3DataCube attribute), 75
ave_chi2 (pycasso.h5datacube.h5Q3DataCube attribute), 57
ave_Mcor (pycasso.h5datacube.h5Q3DataCube attribute), 57
aZ_flux__yx (pycasso.q3datacube_intf.IQ3DataCube attribute), 76
aZ_flux__z (pycasso.q3datacube_intf.IQ3DataCube attribute), 76
aZ_mass__yx (pycasso.q3datacube_intf.IQ3DataCube attribute), 76
aZ_mass__z (pycasso.q3datacube_intf.IQ3DataCube attribute), 76
azimuthalProfileNd() (pycasso.q3datacube_intf.IQ3DataCube method), 66

B

ba (pycasso.h5datacube.h5Q3DataCube attribute), 51
baseId (pycasso.h5datacube.h5Q3DataCube attribute), 49
best_chi2 (pycasso.h5datacube.h5Q3DataCube attribute), 57
best_Mcor (pycasso.h5datacube.h5Q3DataCube attribute), 57

C

califaID (pycasso.h5datacube.h5Q3DataCube attribute), 50
cha_chi2 (pycasso.h5datacube.h5Q3DataCube attribute), 57
cha_Mcor (pycasso.h5datacube.h5Q3DataCube attribute), 57
chains_ave_LAx (pycasso.h5datacube.h5Q3DataCube attribute), 57
chains_ave_mu_cor (pycasso.h5datacube.h5Q3DataCube attribute), 57
chains_ave_par (pycasso.h5datacube.h5Q3DataCube attribute), 57
chains_best_LAx (pycasso.h5datacube.h5Q3DataCube attribute), 57
chains_best_mu_cor (pycasso.h5datacube.h5Q3DataCube attribute), 57
chains_best_par (pycasso.h5datacube.h5Q3DataCube attribute), 57
chains_LAx (pycasso.h5datacube.h5Q3DataCube attribute), 57
chains_mu_cor (pycasso.h5datacube.h5Q3DataCube attribute), 57
chains_par (pycasso.h5datacube.h5Q3DataCube attribute), 57
chi2 (pycasso.h5datacube.h5Q3DataCube attribute), 56
chi2__yx (pycasso.q3datacube_intf.IQ3DataCube attribute), 76
chi2_FIR (pycasso.h5datacube.h5Q3DataCube attribute), 56
chi2_Opt (pycasso.h5datacube.h5Q3DataCube attribute), 56

chi2_PHO (pycasso.h5datacube.h5Q3DataCube attribute), 56
chi2_QHR (pycasso.h5datacube.h5Q3DataCube attribute), 56
chi2_TOT (pycasso.h5datacube.h5Q3DataCube attribute), 56
convexHullMask() (in module pycasso.util), 44

D

DeRed_Lobn__tZz (pycasso.q3datacube_intf.IQ3DataCube attribute), 73
DeRed_Lobn__z (pycasso.q3datacube_intf.IQ3DataCube attribute), 73
DeRed_LobnSD__tZyx (pycasso.q3datacube_intf.IQ3DataCube attribute), 73
DeRed_LobnSD__yx (pycasso.q3datacube_intf.IQ3DataCube attribute), 73
DeRed_M2L__yx (pycasso.q3datacube_intf.IQ3DataCube attribute), 75
distance_Mpc (pycasso.h5datacube.h5Q3DataCube attribute), 50
dl (pycasso.h5datacube.h5Q3DataCube attribute), 50

F

f_err (pycasso.h5datacube.h5Q3DataCube attribute), 53
f_err__lyx (pycasso.q3datacube_intf.IQ3DataCube attribute), 77
f_flag (pycasso.h5datacube.h5Q3DataCube attribute), 53
f_flag__lyx (pycasso.q3datacube_intf.IQ3DataCube attribute), 76
f_obs (pycasso.h5datacube.h5Q3DataCube attribute), 53
f_obs__lyx (pycasso.q3datacube_intf.IQ3DataCube attribute), 76
f_syn (pycasso.h5datacube.h5Q3DataCube attribute), 56
f_syn__lyx (pycasso.q3datacube_intf.IQ3DataCube attribute), 77
f_wei (pycasso.h5datacube.h5Q3DataCube attribute), 56
f_wei__lyx (pycasso.q3datacube_intf.IQ3DataCube attribute), 77
fbase_norm (pycasso.h5datacube.h5Q3DataCube attribute), 54
fill_value (pycasso.h5datacube.h5Q3DataCube attribute), 51
fillImage() (pycasso.q3datacube_intf.IQ3DataCube method), 67
findHLR_pix_CID() (califa.Q3DataCube.Q3DataCube method), 28
flux_unit (pycasso.h5datacube.h5Q3DataCube attribute), 50
fobs_norm (pycasso.h5datacube.h5Q3DataCube attribute), 53

G

galaxyName (pycasso.h5datacube.h5Q3DataCube attribute), 50
gen_rebin() (in module pycasso.util), 43
getAngle() (in module pycasso.util), 46
getApertureMask() (in module pycasso.util), 44
getAverageFilledImage() (in module pycasso.util), 44
getDezonificationWeight() (pycasso.q3datacube_intf.IQ3DataCube method), 59
getDistance() (in module pycasso.util), 45
getEllipseParams() (in module pycasso.util), 44
getEllipseParams() (pycasso.q3datacube_intf.IQ3DataCube method), 68
getGenHalfRadius() (in module pycasso.util), 44
getHalfRadius() (pycasso.q3datacube_intf.IQ3DataCube method), 69
getHLR_pix() (pycasso.q3datacube_intf.IQ3DataCube method), 70
getImageAngle() (in module pycasso.util), 46
getImageDistance() (in module pycasso.util), 45
getPixelAngle() (pycasso.q3datacube_intf.IQ3DataCube method), 69
getPixelDistance() (pycasso.q3datacube_intf.IQ3DataCube method), 68
getSFR() (pycasso.q3datacube_intf.IQ3DataCube method), 71
getYXToRadialBinsTensor() (califa.Q3DataCube.Q3DataCube method), 27
getYXToRadialBinsTensorExact() (pycasso.q3datacube_intf.IQ3DataCube method), 61
getZoneToRadialBinsTensor() (califa.Q3DataCube.Q3DataCube method), 27
getZoneToYXTensor() (califa.Q3DataCube.Q3DataCube method), 26
growthInAge() (pycasso.q3datacube_intf.IQ3DataCube method), 70

H

h5Q3DataCube (class in pycasso.h5datacube), 49
header (pycasso.h5datacube.h5Q3DataCube attribute), 51
HLR_pc (pycasso.h5datacube.h5Q3DataCube attribute), 50
HLR_pix (pycasso.h5datacube.h5Q3DataCube attribute), 50

I

index_Best_SSP (pycasso.h5datacube.h5Q3DataCube attribute), 55
integrated_at_flux (pycasso.q3datacube_intf.IQ3DataCube attribute), 75
integrated_at_mass (pycasso.q3datacube_intf.IQ3DataCube attribute), 75
integrated_aZ_flux (pycasso.q3datacube_intf.IQ3DataCube attribute), 76
integrated_aZ_mass (pycasso.q3datacube_intf.IQ3DataCube attribute), 76
integrated_chains_ave_LAx (pycasso.h5datacube.h5Q3DataCube attribute), 58
integrated_chains_ave_mu_cor (pycasso.h5datacube.h5Q3DataCube attribute), 59
integrated_chains_ave_par (pycasso.h5datacube.h5Q3DataCube attribute), 58
integrated_chains_best_LAx (pycasso.h5datacube.h5Q3DataCube attribute), 58
integrated_chains_best_mu_cor (pycasso.h5datacube.h5Q3DataCube attribute), 59
integrated_chains_best_par (pycasso.h5datacube.h5Q3DataCube attribute), 58
integrated_chains_LAx (pycasso.h5datacube.h5Q3DataCube attribute), 59
integrated_chains_mu_cor (pycasso.h5datacube.h5Q3DataCube attribute), 59
integrated_chains_par (pycasso.h5datacube.h5Q3DataCube attribute), 58
integrated_DeRed_Lobn (pycasso.q3datacube_intf.IQ3DataCube attribute), 73
integrated_DeRed_Lobn__tZ (pycasso.q3datacube_intf.IQ3DataCube attribute), 73
integrated_DeRed_M2L (pycasso.q3datacube_intf.IQ3DataCube attribute), 75
integrated_f_err (pycasso.h5datacube.h5Q3DataCube attribute), 54
integrated_f_flag (pycasso.h5datacube.h5Q3DataCube attribute), 54
integrated_f_obs (pycasso.h5datacube.h5Q3DataCube attribute), 54
integrated_f_syn (pycasso.h5datacube.h5Q3DataCube attribute), 58
integrated_f_wei (pycasso.h5datacube.h5Q3DataCube attribute), 58
integrated_keywords (pycasso.h5datacube.h5Q3DataCube attribute), 51
integrated_Lobn (pycasso.q3datacube_intf.IQ3DataCube attribute), 72
integrated_Lobn__tZ (pycasso.q3datacube_intf.IQ3DataCube attribute), 72
integrated_M2L (pycasso.q3datacube_intf.IQ3DataCube attribute), 75
integrated_Mcor (pycasso.q3datacube_intf.IQ3DataCube attribute), 74
integrated_Mcor__tZ (pycasso.q3datacube_intf.IQ3DataCube attribute), 74
integrated_Mini (pycasso.q3datacube_intf.IQ3DataCube attribute), 74
integrated_Mini__tZ (pycasso.q3datacube_intf.IQ3DataCube attribute), 74
integrated_popAV_tot (pycasso.h5datacube.h5Q3DataCube attribute), 58
integrated_popexAV_flag (pycasso.h5datacube.h5Q3DataCube attribute), 58
integrated_popmu_cor (pycasso.h5datacube.h5Q3DataCube attribute), 58
integrated_popmu_ini (pycasso.h5datacube.h5Q3DataCube attribute), 58
integrated_popx (pycasso.h5datacube.h5Q3DataCube attribute), 57
integrated_SSP_adev (pycasso.h5datacube.h5Q3DataCube attribute), 58
integrated_SSP_AV (pycasso.h5datacube.h5Q3DataCube attribute), 58
integrated_SSP_chi2r (pycasso.h5datacube.h5Q3DataCube attribute), 58
integrated_SSP_x (pycasso.h5datacube.h5Q3DataCube attribute), 58
IQ3DataCube (class in pycasso.q3datacube_intf), 59

K

keywords (pycasso.h5datacube.h5Q3DataCube attribute), 50

L

l_fin (pycasso.h5datacube.h5Q3DataCube attribute), 50
l_ini (pycasso.h5datacube.h5Q3DataCube attribute), 50
l_obs (pycasso.h5datacube.h5Q3DataCube attribute), 59
listBases() (in module pycasso.h5datacube), 49
listQbickRuns() (in module pycasso.h5datacube), 49
listRuns() (in module pycasso.h5datacube), 49
loadGalaxy() (pycasso.h5datacube.h5Q3DataCube method), 51
loadGalaxy() (pycasso.q3datacube_intf.IQ3DataCube method), 59
Lobn__tZz (pycasso.q3datacube_intf.IQ3DataCube attribute), 73
Lobn__z (pycasso.q3datacube_intf.IQ3DataCube attribute), 73
LobnSD__tZyx (pycasso.q3datacube_intf.IQ3DataCube attribute), 73
LobnSD__yx (pycasso.q3datacube_intf.IQ3DataCube attribute), 73
Lobs_norm (pycasso.h5datacube.h5Q3DataCube attribute), 55
logAgeBaseBins (pycasso.q3datacube_intf.IQ3DataCube attribute), 77

M

M2L__yx (pycasso.q3datacube_intf.IQ3DataCube attribute), 75
masterListData (pycasso.h5datacube.h5Q3DataCube attribute), 50
Mcor__tZz (pycasso.q3datacube_intf.IQ3DataCube attribute), 74
Mcor__z (pycasso.q3datacube_intf.IQ3DataCube attribute), 74
Mcor_tot (pycasso.h5datacube.h5Q3DataCube attribute), 55
McorSD__tZyx (pycasso.q3datacube_intf.IQ3DataCube attribute), 74
McorSD__yx (pycasso.q3datacube_intf.IQ3DataCube attribute), 74
metBase (pycasso.h5datacube.h5Q3DataCube attribute), 51
metBaseMx() (pycasso.q3datacube_intf.IQ3DataCube method), 77
Mini__tZz (pycasso.q3datacube_intf.IQ3DataCube attribute), 74
Mini__z (pycasso.q3datacube_intf.IQ3DataCube attribute), 74
Mini_tot (pycasso.h5datacube.h5Q3DataCube attribute), 55
MiniSD__tZyx (pycasso.q3datacube_intf.IQ3DataCube attribute), 75
MiniSD__yx (pycasso.q3datacube_intf.IQ3DataCube attribute), 75
Mstars (pycasso.h5datacube.h5Q3DataCube attribute), 54

N

N_age (pycasso.h5datacube.h5Q3DataCube attribute), 50
N_base (pycasso.h5datacube.h5Q3DataCube attribute), 50
N_met (pycasso.h5datacube.h5Q3DataCube attribute), 50
N_x (pycasso.h5datacube.h5Q3DataCube attribute), 50
N_y (pycasso.h5datacube.h5Q3DataCube attribute), 50
N_zone (pycasso.h5datacube.h5Q3DataCube attribute), 50
Nglobal_steps (pycasso.h5datacube.h5Q3DataCube attribute), 56
Nl_eff (pycasso.h5datacube.h5Q3DataCube attribute), 55
Nl_obs (pycasso.h5datacube.h5Q3DataCube attribute), 50
NOl_eff (pycasso.h5datacube.h5Q3DataCube attribute), 55
Ntot_clipped (pycasso.h5datacube.h5Q3DataCube attribute), 56

P

pa (pycasso.h5datacube.h5Q3DataCube attribute), 51
parsecPerPixel (pycasso.h5datacube.h5Q3DataCube attribute), 50
pixelAngle__yx (pycasso.h5datacube.h5Q3DataCube attribute), 51
pixelDistance__yx (pycasso.h5datacube.h5Q3DataCube attribute), 51
PIXSIZE (pycasso.h5datacube.h5Q3DataCube attribute), 50
popAV_tot (pycasso.h5datacube.h5Q3DataCube attribute), 54
popAV_tot__tZyx (pycasso.q3datacube_intf.IQ3DataCube attribute), 72
popexAV_flag (pycasso.h5datacube.h5Q3DataCube attribute), 54
popmu_cor (pycasso.h5datacube.h5Q3DataCube attribute), 54
popmu_cor__tZyx (pycasso.q3datacube_intf.IQ3DataCube attribute), 72
popmu_ini (pycasso.h5datacube.h5Q3DataCube attribute), 54
popmu_ini__tZyx (pycasso.q3datacube_intf.IQ3DataCube attribute), 72
popx (pycasso.h5datacube.h5Q3DataCube attribute), 54
popx__tZyx (pycasso.q3datacube_intf.IQ3DataCube attribute), 72
pycasso.util (module), 43
pycassoVersion (pycasso.h5datacube.h5Q3DataCube attribute), 49

Q

Q3DataCube (class in califa.Q3DataCube), 26
q_norm (pycasso.h5datacube.h5Q3DataCube attribute), 50
qbickRunId (pycasso.h5datacube.h5Q3DataCube attribute), 49
qConvexHull (pycasso.q3datacube_intf.IQ3DataCube attribute), 72
qFilledMask (pycasso.h5datacube.h5Q3DataCube attribute), 53
qFlagRatio (pycasso.h5datacube.h5Q3DataCube attribute), 52
qHollowPixels (pycasso.q3datacube_intf.IQ3DataCube attribute), 72
qMask (pycasso.h5datacube.h5Q3DataCube attribute), 53
qNoise (pycasso.h5datacube.h5Q3DataCube attribute), 52
qNoiseUnmasked (pycasso.h5datacube.h5Q3DataCube attribute), 52
qPipeNoise (pycasso.h5datacube.h5Q3DataCube attribute), 52
qPipeNoiseOrig (pycasso.h5datacube.h5Q3DataCube attribute), 53
qPipeZonesNoiseOrig (pycasso.h5datacube.h5Q3DataCube attribute), 53
qSegmentationSn (pycasso.h5datacube.h5Q3DataCube attribute), 53
qSignal (pycasso.h5datacube.h5Q3DataCube attribute), 52
qSignalUnmasked (pycasso.h5datacube.h5Q3DataCube attribute), 52
qSn (pycasso.h5datacube.h5Q3DataCube attribute), 52
qSnMask (pycasso.h5datacube.h5Q3DataCube attribute), 53
qSpatialMask (pycasso.h5datacube.h5Q3DataCube attribute), 53
qVersion (pycasso.h5datacube.h5Q3DataCube attribute), 49
qZones (pycasso.h5datacube.h5Q3DataCube attribute), 52
qZonesNoise (pycasso.h5datacube.h5Q3DataCube attribute), 52
qZonesNoiseOrig (pycasso.h5datacube.h5Q3DataCube attribute), 52
qZonesSn (pycasso.h5datacube.h5Q3DataCube attribute), 52
qZonesSnOrig (pycasso.h5datacube.h5Q3DataCube attribute), 52

R

radialProfile() (in module pycasso.util), 47
radialProfile() (pycasso.q3datacube_intf.IQ3DataCube method), 63
radialProfileExact() (pycasso.q3datacube_intf.IQ3DataCube method), 62
radialProfileNd() (pycasso.q3datacube_intf.IQ3DataCube method), 64

S

setGeometry() (pycasso.q3datacube_intf.IQ3DataCube method), 60
setSmoothDezonification() (pycasso.q3datacube_intf.IQ3DataCube method), 59
SSP_adev (pycasso.h5datacube.h5Q3DataCube attribute), 54
SSP_AV (pycasso.h5datacube.h5Q3DataCube attribute), 55
SSP_chi2r (pycasso.h5datacube.h5Q3DataCube attribute), 54
SSP_x (pycasso.h5datacube.h5Q3DataCube attribute), 55

V

v_0 (pycasso.h5datacube.h5Q3DataCube attribute), 55
v_0__yx (pycasso.q3datacube_intf.IQ3DataCube attribute), 72
v_d (pycasso.h5datacube.h5Q3DataCube attribute), 55
v_d__yx (pycasso.q3datacube_intf.IQ3DataCube attribute), 72

X

x0 (pycasso.h5datacube.h5Q3DataCube attribute), 50

Y

y0 (pycasso.h5datacube.h5Q3DataCube attribute), 50

Z

zoneArea_pc2 (pycasso.h5datacube.h5Q3DataCube attribute), 51
zoneArea_pix (pycasso.h5datacube.h5Q3DataCube attribute), 51
zoneAreaMx() (pycasso.q3datacube_intf.IQ3DataCube method), 77
zonePos (pycasso.h5datacube.h5Q3DataCube attribute), 51
zoneToRad() (pycasso.q3datacube_intf.IQ3DataCube method), 65
zoneToYX() (pycasso.q3datacube_intf.IQ3DataCube method), 59