CubeSPA

Harrison Souchereau

Jan 19, 2024

CONTENTS:

1 Loading Data **3**
- 1.1 Getting Started . . . 3

2 2D Analysis **9**
- 2.1 Generating Moment Maps . . . 9
- 2.2 Making Overlays . . . 10

3 Cube Analysis **17**
- 3.1 Comparing Two Datacubes . . . 17
- 3.2 Spectral Analysis . . . 17
- 3.3 Position-Velocity diagrams . . . 19

4 Coming soon **21**
- 4.1 Generating / Fitting Velocity Models . . . 21

5 Indices and tables **23**
- 5.1 Module Documentation . . . 23

Python Module Index **29**

Index **31**

CubeSPA is a fully-featured set of utilities to process, analyze, and display useful information of datacubes, particularly of astronomical data. It fully supports utilities such as moment map creation, spectral analysis, position-velocity diagrams, and many others.

To use **CubeSPA**, install it using

`pip install cubespa` (not yet, but soon)

To begin using CubeSPA, create a `cubespa.CubeSPA` object with the filename for your cube in the following way:

```
>>> filename = "path/to/cube.fits"
>>> c = cubespa.CubeSPA(filename)
```

You can load in moment maps (assuming the convention from maskmoment) with the following. If your maskmoment output is `path/to/maskmoment.mom0.fits.gz`, for the moment 0 map (.mom1, .mom2 for the others), these are loaded as follows below. With moment maps loaded, you can also create a bounding box around "valid" data by calling the `limits = "auto"` feature.

```
>>> filename = "path/to/cube.fits"
>>> mommaps = "path/to/maskmoment"
>>> c = cubespa.CubeSPA(cube_fn, mom_maps=mommaps, limits="auto")
```

See getting_started.rst for more information.

CHAPTER

ONE

LOADING DATA

1.1 Getting Started

The following documentation outlines how to load in data, from an initial cube, to moment maps, to additional data in both single frame and RGB image form.

1.1.1 Initial CubeSPA object

To begin using CubeSPA, create a `cubespa.CubeSPA` object with the filename for your cube in the following way:

```
>>> import cubespa
>>> filename = "path/to/cube.fits"
>>> c = cubespa.CubeSPA(filename)
```

You can load in moment maps (assuming the convention from maskmoment) with the following. If your maskmoment output is `path/to/maskmoment.mom0.fits.gz`, for the moment 0 map (.mom1, .mom2 for the others), these are loaded as follows below. With moment maps loaded, you can also create a bounding box around "valid" data by calling the `limits = "auto"` feature.

```
>>> filename = "path/to/cube.fits"
>>> mommaps = "path/to/maskmoment"
>>> c = cubespa.CubeSPA(filename, mom_maps=mommaps, limits="auto")
```

1.1.2 Additional data

Additional data that doesn't require a full cube object can be loaded as a `cubespa.DataSet()` object. Note that all of the data in the cubeSPA object loaded above are also cubespa.DataSet() objects, which stores the wcs and header information for easier access.

For example, if you had an H-alpha map of your galaxy, you might load it as follows:

```
>>> halpha = cubespa.load_data(halpha_fn, label="HALPHA")
```

However, this map might not be aligned with your cube. CubeSPA uses the reproject package to properly align maps together. Once you have your `DataSet` object, you can align it with

```
>>> halpha_interp = cubespa.align_image(c.mom_maps.mom0, halpha)
```

where we are using the moment 0 map to align the images. CubeSPA *should* be able to automatically align images with a cube directly, where it will do some wcs `dropaxis` trickery to try and match things together.

Lastly, it is good practice to add these datasets to the parent CubeSPA object's `additional_maps` attribute:

>> c.additional_maps.extend([halpha_interp, (...)])

1.1.3 RGB images

`matplotlib` is notoriously tricky for RGB images of astronomical data. CubeSPA has some built-in features to improve the experience with displaying RGB data, particularly HST images.

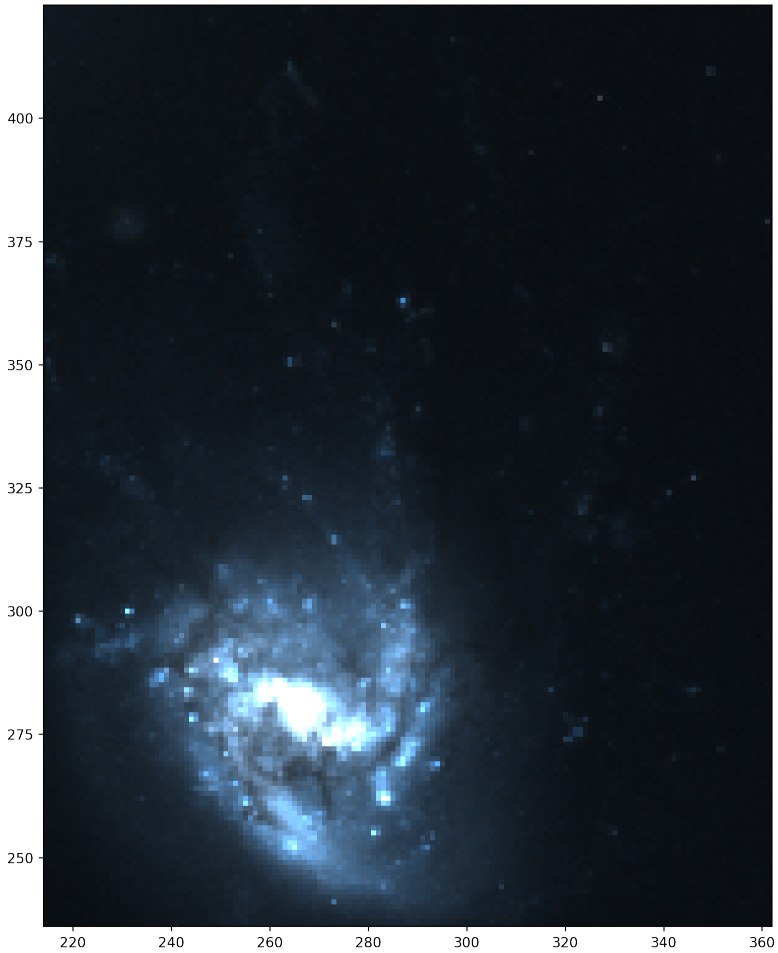
For example, if we load in an hst image and then create an aligned image, we can see what it looks like by default with the following code:

```
>>> hst = cubespa.load_data(hst_fn, rgb_index=None, label="HST")
>>> hst_interp = cubespa.align_image(c.mom_maps.mom0, hst)
```

```
>>> test = hst_interp.data.transpose(1,2,0) # Transpose the data into the proper rgb␣
↪pixel format for matplotlib
```

```
>>> cubespa.plotting.plot_rgb(test, lims=np.array(c.limits), outname="./rgb_nonorm.png")
```

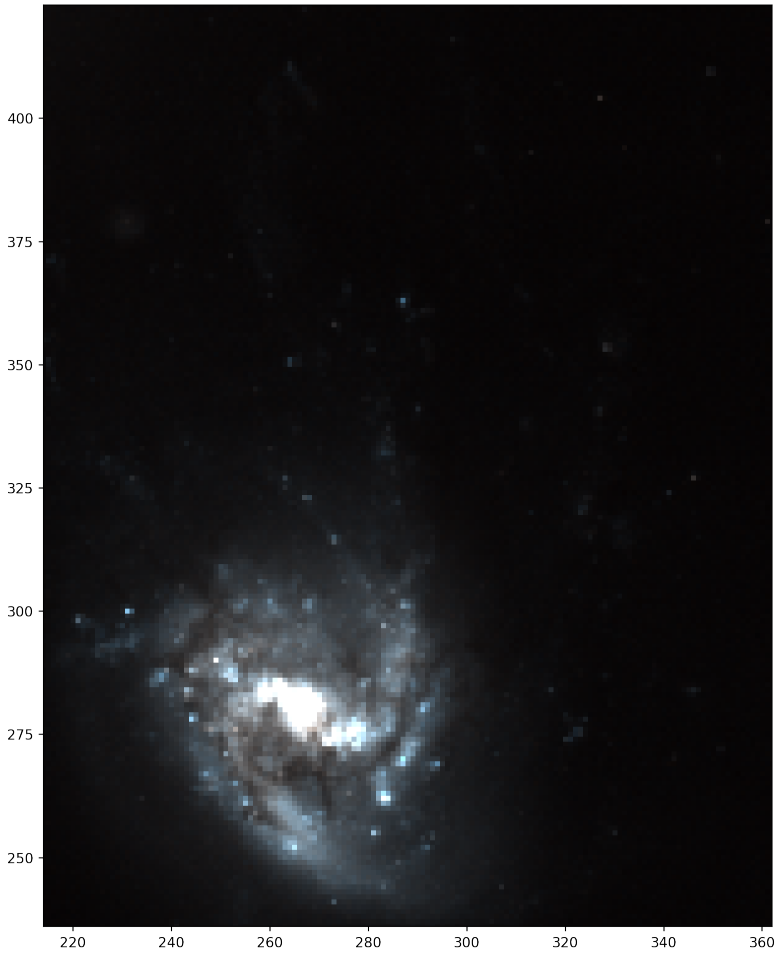
This returns the following.



The RGB image can be histogram-normalized by doing the following:

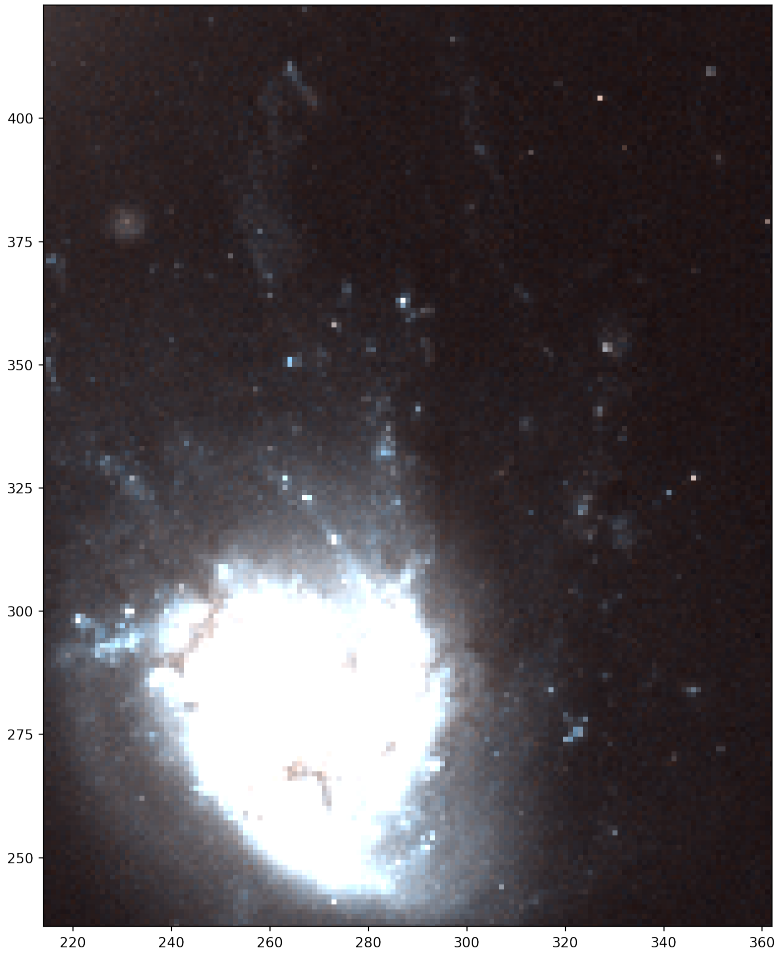
```
>>> norm = cubespa.normalized_rgb_image(hst_interp.data, sigma=(2, 5))
>>> cubespa.plotting.plot_rgb(norm, lims=np.array(c.limits))
```

where `sigma` defines the lower and upper bounds to stretch each RGB frame to. This returns a map that look like:



If the user wants to look at faint features, simply decrease the upper stretch. This will increase the visibility of faint features at the cost of saturating the central disk.

```
>>> norm = cubespa.normalized_rgb_image(hst_interp.data, sigma=(2, 1))
>>> cubespa.plotting.plot_rgb(norm, lims=np.array(c.limits))
```



1.1.4 Cutouts and Regions

To create specific cutouts (for analysis of certain regions), the user supplies the parent CubeSPA object, the central location of the cutout, and the size of the cutout (either as an int, for a square, or as a tuple to establish a rectangular region). The output is a new CubeSPA object where the cube, moment maps, and any additional maps are trimmed to the location and size of the cutout.

For example, if I was looking at 3 different regions (blob, fallback, and outskirts) for some datacube of a galaxy, I would create it with the following.

```
>>> blob = cubespa.gen_cutout(c, (345, 290), 15, show_bbox=True)
>>> fallback = cubespa.gen_cutout(c, (305, 310), (20, 15), show_bbox=True)
>>> outskirts = cubespa.gen_cutout(c, (270, 403), (15, 35), show_bbox=True)
```

The additional parameter `show_bbox` will generate a plot to show you where the cutout falls on the parent image. This is helpful for more closely aligning the cutouts.

CHAPTER

TWO

2D ANALYSIS

2.1 Generating Moment Maps

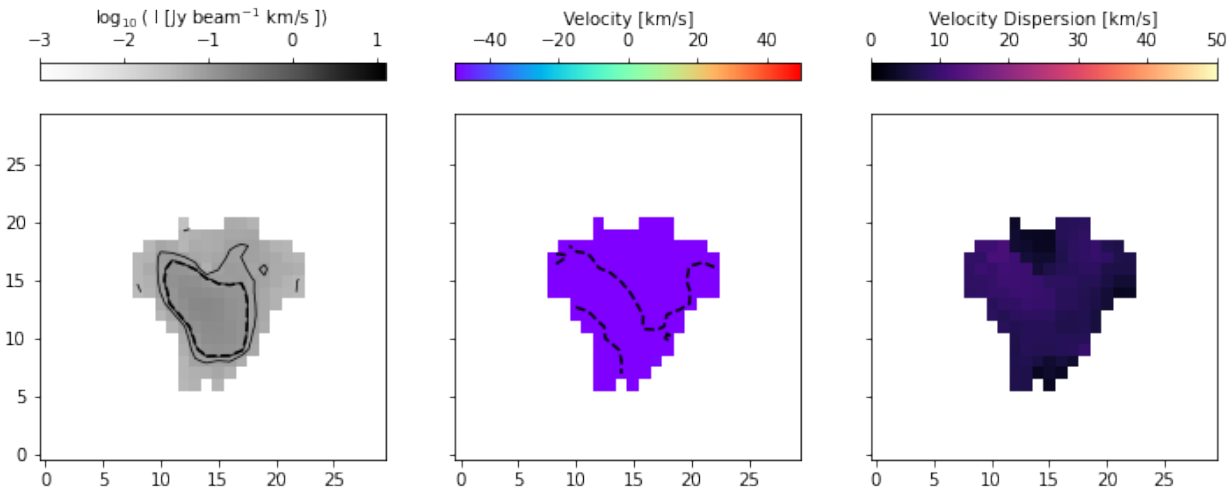
Generating moment maps is trivially easy using CubeSPA.

To make a display of the moment maps, simply do the following:

```
>>> import cubespa
>>> c = cubespa.CubeSPA(cube_filename, mom_maps=mom_maps_filename)
>>> c.plot_moment_maps()
```

If you have made some cutouts of your object, you can do the same, but it is recommended that you set `use_limits` to `False` as the limits are related directly to the parent cube object.

```
>>> blob.plot_moment_maps(use_limits=False)
```



2.1.1 Reference/API

cubespa.plotting.mommap_plots.**moment_map_plot**(*cubespa_obj*, *filename=None*, *use_limits=True*, ***kwargs*)

Generate moment map plots.

Parameters

- **cubespa_obj** (cubespa.CubeSPA) – The input CubeSPA object, with valid moment maps loaded.
- **filename** (*str, optional*) – Output filename, in which the plot is just shown instead of saved. Defaults to None.
- **use_limits** (*bool, optional*) – Whether or not to use limits from the CubeSPA object.

  It is a good idea to set to False for cutout objects, as their limits will be relative to the initial CubeSPA object, and their desired limits will be the entire array. Defaults to True.

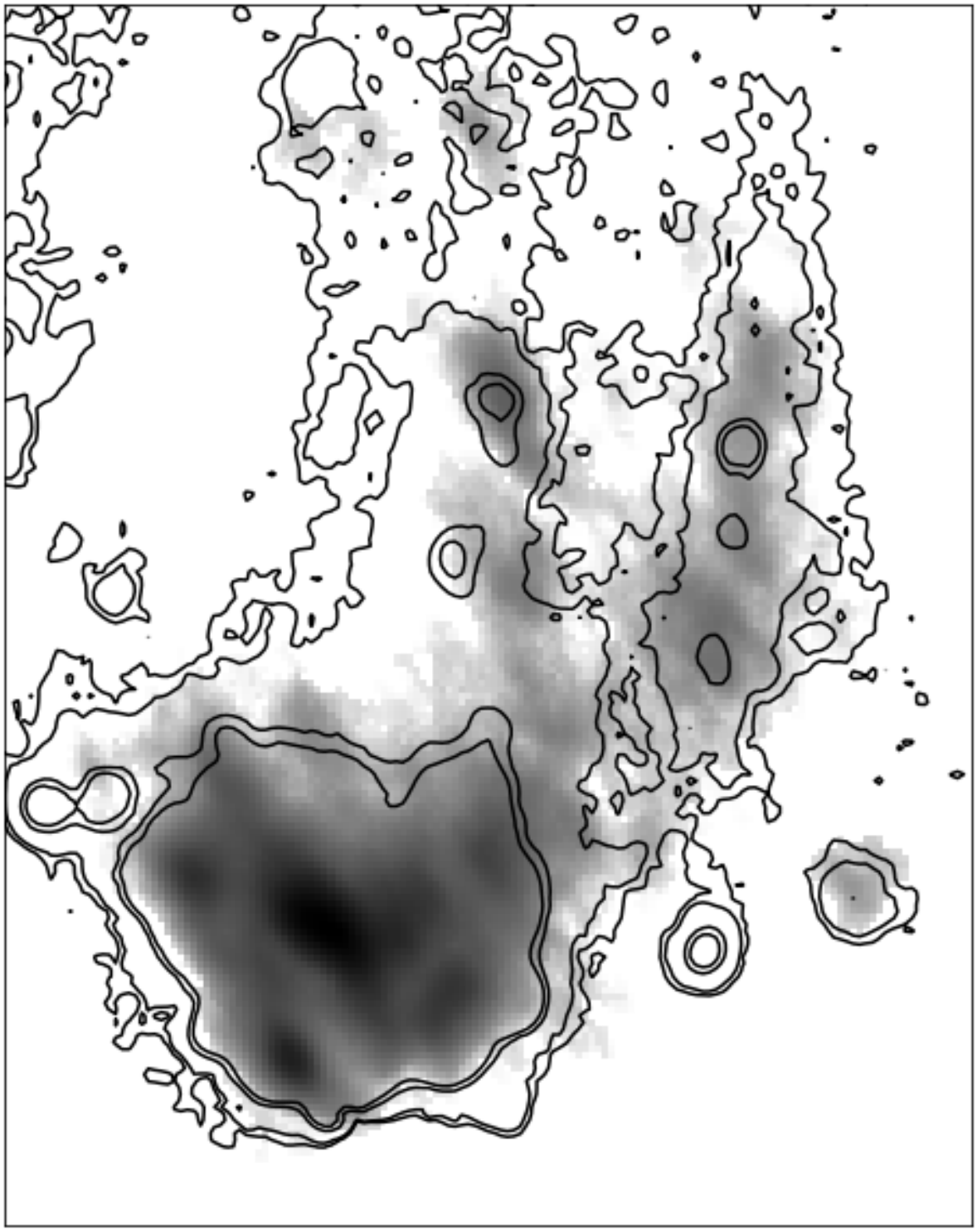
2.2 Making Overlays

Overlays are useful features to display isophotal contours of one image onto another.

Once you have aligned datasets (see getting_started), you can create an overlay by using:

```
>>> cubespa.plotting.overlay_plot(c.mom_maps.mom0, c.additional_maps[0],
...                               lims=c.limits, levels=[10, 20, 100, 150])
```

where in this instance, we are showing the moment 0 CO distribution with H-alpha contours overlaid in black. The output plot looks like this:



For an RGB image, use the following instead:

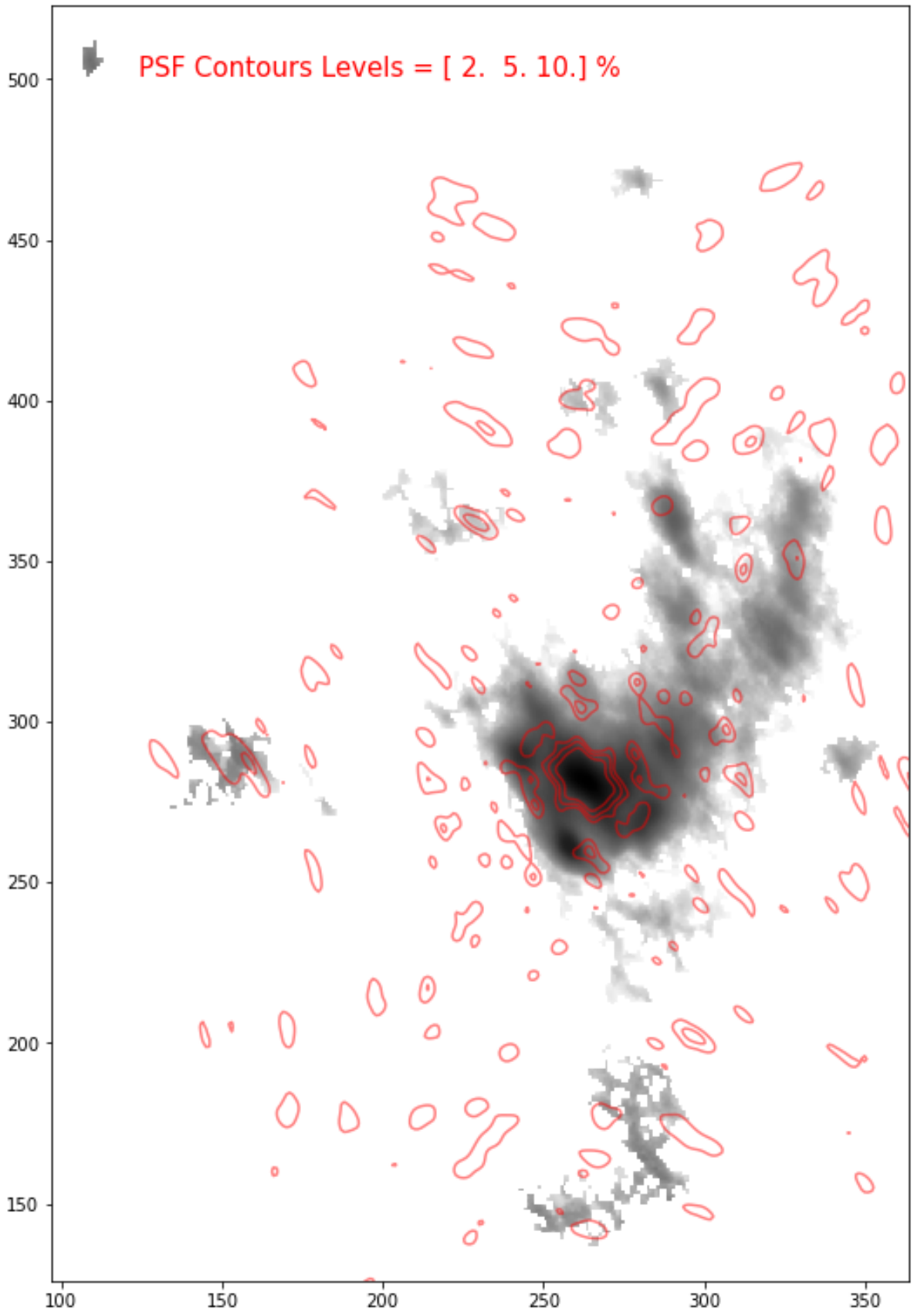
```
>>> norm = cubespa.normalized_rgb_image(hst_interp.data, sigma=(2, 1))
>>> cubespa.plotting.rgb_overlay(norm, c.mom_maps.mom0,
...                              levels=[0.02, 0.05, 0.1, 1], lims=blob.limits, colors=
↪"white")
```

which plots CO contours on top of the HST RGB image, which for this region reveals a small compact stellar feature at the head of the “blob”.

2.2.1 PSF Overlays for Sidelobe Checking

If your cube/cleaning has a PSF that you have imported into cubespa with the `psf=psf_fn` feature, you can overplot it using the following code:

```
>>> c = cubespa.CubeSPA(cube=cube_fn, psf=psf_fn, limits="auto")
>>> cubespa.plotting.plot_psf_overlay(c, y0=102, x0=25)
```


PSF Contours Levels = [2. 5. 10.] %
500
450
400
350
300
250
200
150
100
150
200
250
300
350

Note that in this case, the psf has been shifted down by 102 pixels, and to the right by 25 pixels. This overlay is especially helpful when hunting for sidelobe features in a dataset, which can be revealed by looking at contours in the PSF.

CHAPTER

THREE

CUBE ANALYSIS

3.1 Comparing Two Datacubes

Comparing two different datacubes is an important step for any analysis, whether it is cubes from a different cleaning algorithm (the PHANGs pipeline vs a manual CASA cleaning, for example), or different momemt mask parameters, or cubes cleaned with different parameters, seeing how these cubes compare directly with one another is extremely important.

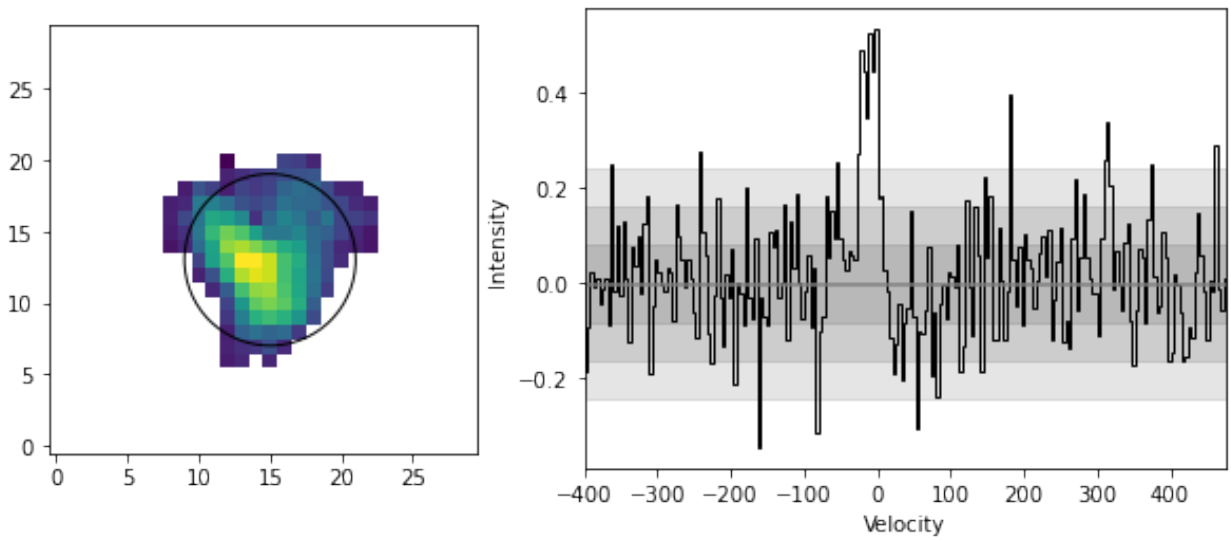
To get started comparing two cubes, one uses the `CubeComparison` object.

3.2 Spectral Analysis

To generate spectra, the user specifies the location of the aperture (elliptical by default), and the shape. Continuing with our `blob` example, we generate spectra like so:

```
>>> blob.create_spectra((15, 13), (6, 6), plot=True)
```

This creates the following plot, showing the location of the spectral features in either velocity or channel space. CubeSPA uses sigma clipping to determine an RMS of the spectra, and then plots the 1,2 and 3-sigma levels as a shaded grey region.



3.2.1 SNR Analysis

To manually check the SNR of a given region (say, between channels 20 and 40 of the array `spec`), you can with the following method:

```
>>> chan_min, chan_max = 20, 40
>>> snr = cubespa.calc_snr(spec, chan_min, chan_max)
```

The signal to noise ratio between the two channels is given by:

$$SNR = \frac{\Sigma F}{RMS_{spec} * \sqrt{n_{chan}}}$$

cubespa.plotting.spectra_plots.**multispec_plot**(*cubespa_obj*, *aper_list*, *spec_list*, ***kwargs*)

cubespa.plotting.spectra_plots.**plot_spectra**(*data*, *aper*)

cubespa.plotting.spectra_plots.**spectra_comparison**(*cubecomp*, *a1*, *a2*, *s1*, *s2*, *chan_ranges=None*, ***kwargs*)

cubespa.plotting.spectra_plots.**spectra_plot**(*cubespa_obj*, *aper*, *spectrum*)

Create a plot showing both the image with overlaid spectra, as well as the spectrum with RMS levels shown.

Parameters

- **cubespa_obj** (*cubespa.CubeSPA*) – CubeSPA object.
- **aper** (*photutils.aperture*) – Input aperture, generated using cubespa.spectra
- **spectrum** (*_type_*) – _description_

cubespa.spectra.**align_apertures**(*aper_list*, *wcs1*, *wcs2*)

Align and resize a set of apertures from wcs1 to wcs2

Parameters

- **aper_list** (*list*) – List of apertures in the following format: [(p1, p2), (s1, s2)] where the first tuple is the (x,y) position and the second tuple is the (x,y) height.
- **wcs1** (*astropy.wcs.WCS*) – WCS that aper_list apertures are already aligned to
- **wcs2** (*astropy.wcs.WCS*) – WCS to align apertures to

Returns

description

Return type

type

cubespa.spectra.**analyze_spectra**(*spec*, *sigma=2*, *cmin=None*, *cmax=None*)

cubespa.spectra.**calc_snr**(*spec*, *chan_min*, *chan_max*)

cubespa.spectra.**create_aperture**(*cubespa_obj*, *position*, *shape*, *aper_type='elliptical'*, *plot=False*)

Generate photutils aperture of desired type, position, and shape.

Returns

Photutils aperture

Return type
photutils.aperture

cubespa.spectra.**get_spectra**(*cube*, *aper*)

Get the spectra through a datacube at the position and size of a given aperture.

Parameters

- **cube** (*ndarray*) – _description_
- **aper** (*photutils aperture*) – Elliptical or circular aperture/annulus.

Returns
description

Return type
type

cubespa.spectra.**multi_spec**(*cubespa_obj*, *spec_info*)

Generate a list of apertures and spectra for better diagnostics.

Parameters

- **cubespa_obj** (cubespa.CubeSPA) – Input CubeSPA object to pull spectra from.
- **spec_info** (*_type_*) – A list of spectra position and sizes, for example, to get two spectra, you would
- **have** –

  spec_info = [[(p1, p2), (s1, s2)],
  [(p3, p4), (s3, s4)]]

Returns
A list of apertures and a list of spectra.

Return type
tuple(array)

cubespa.spectra.**spectra_comparison**(*cubecomp*, *aper_list*, *outname=None*, *plot_ticks=True*, *chan_ranges=None*)

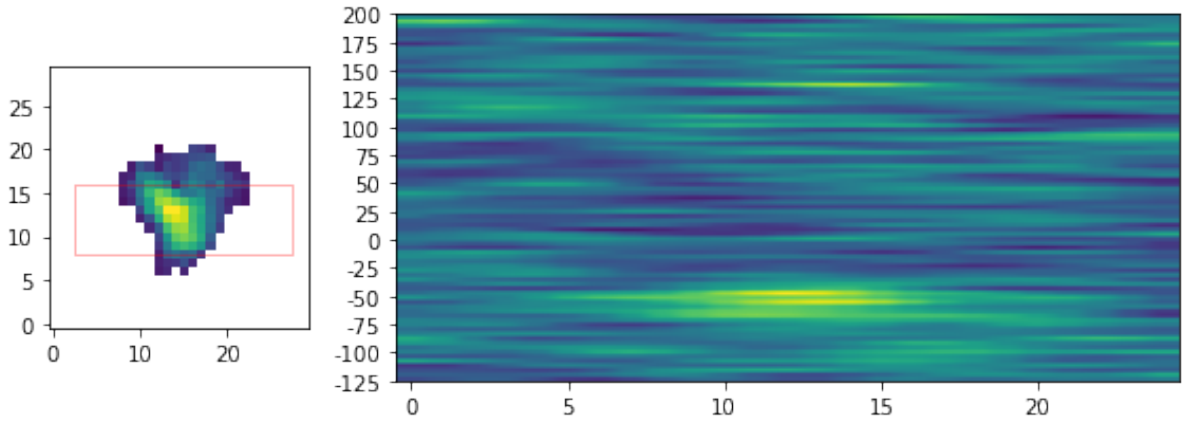
3.3 Position-Velocity diagrams

Position-velocity diagrams (PVDs) plot the strength of velocity features along some specified path, most often along the major and minor axes. A disk with only circular motions will show the rotation curve along the major axis, and should be flat along the minor axis. Therefore, PVDs are useful tools to find non-circular velocity features without explicit modeling of components.

To generate a PVD, do the following:

```
>>> pvd = cubespa.gen_pvd(blob, (15, 12), 25, 0, width=8)
>>> cubespa.plotting.pvd_plot(blob, pvd, vmin=-125)
```

Wich will spit out both the map of the region with the overlaid PVD path (with proper width displayed), as well as the PVD itself. In the example, the region of the PVD cutting through the "blob" shows emission in blue-shifted velocity channels, consistent with what is seen in the moment 1 mean velocity map.


25
20
15
10
5
0
0
10
20
200
175
150
125
100
75
50
25
0
-25
-50
-75
-100
-125
0
5
10
15
20

CHAPTER

FOUR

COMING SOON

4.1 Generating / Fitting Velocity Models

CHAPTER

FIVE

INDICES AND TABLES

5.1 Module Documentation

5.1.1 Utilities

class cubespa.cubespa.**CubeComparison**(*cube1:* CubeSPA, *cube2:* CubeSPA)

Bases: object

create_spectra(*position*, *size*, *return_products=False*, *plot=False*)

readout()

class cubespa.cubespa.**CubeSPA**(*cube*, *data_index=0*, *psf=None*, *mom_maps=None*, *additional_maps=[]*, *center=None*, *position_angle=None*, *eps=None*, *vsys=0*, *limits=None*, *plot_dir=None*, ***kwargs*)

Bases: object

Base input class for a CubeSPA object

create_spectra(*position*, *size*, *return_products=False*, *plot=False*)

get_beam_area()

load_dir()

plot_moment_maps(*use_limits=True*, ***kwargs*)

velocities_from_wcs(*vsys=0*)

class cubespa.data.**DataSet**(*data=None*, *wcs=None*, *header=None*, *label=None*)

Bases: object

class cubespa.data.**MomentMaps**(*mom0=None*, *mom1=None*, *mom2=None*, *data_index=0*)

Bases: object

class cubespa.data.**RGBImage**(*data=None*, *wcs=None*, *header=None*, *label=None*)

Bases: *DataSet*

cubespa.data.**gen_cutout**(*cubespa_obj*, *cent*, *size*, *show_bbox=False*)

cubespa.data.**handle_data**(*data*, *handler*, *data_index=0*)

Handle incoming data for the CubeSPA object. If not a string, CubeSPA will return the data enclosed in a dataset without additional info. :param data: Incoming data. If str, CubeSPA will automatically load it in. :type data: str or cubespa.DataSet :param handler: Method to handle data, either set to load_data or load_moment_maps :type handler: method :param data_index: Index to find data in. Defaults to 0. :type data_index: int, optional

Returns
The output DataSet object.

Return type
utils.DataSet

cubespa.data.**load_data**(*filename*, *data_index=0*, *rgb_index=None*, *label=None*)

cubespa.data.**load_moment_maps**(*topdir*, *data_index=0*)

cubespa.spectra.**align_apertures**(*aper_list*, *wcs1*, *wcs2*)
Align and resize a set of apertures from wcs1 to wcs2

Parameters

- **aper_list** (*list*) – List of apertures in the following format: [(p1, p2), (s1, s2)] where the first tuple is the (x,y) position and the second tuple is the (x,y) height.
- **wcs1** (*astropy.wcs.WCS*) – WCS that aper_list apertures are already aligned to
- **wcs2** (*astropy.wcs.WCS*) – WCS to align apertures to

Returns
description

Return type
type

cubespa.spectra.**analyze_spectra**(*spec*, *sigma=2*, *cmin=None*, *cmax=None*)

cubespa.spectra.**calc_snr**(*spec*, *chan_min*, *chan_max*)

cubespa.spectra.**create_aperture**(*cubespa_obj*, *position*, *shape*, *aper_type='elliptical'*, *plot=False*)
Generate photutils aperture of desired type, position, and shape.

Returns
Photutils aperture

Return type
photutils.aperture

cubespa.spectra.**get_spectra**(*cube*, *aper*)
Get the spectra through a datacube at the position and size of a given aperture.

Parameters

- **cube** (*ndarray*) – _description_
- **aper** (*photutils aperture*) – Elliptical or circular aperture/annulus.

Returns
description

Return type
type

cubespa.spectra.**multi_spec**(*cubespa_obj*, *spec_info*)
Generate a list of apertures and spectra for better diagnostics.

Parameters

- **cubespa_obj** (cubespa.CubeSPA) – Input CubeSPA object to pull spectra from.
- **spec_info** (*_type_*) – A list of spectra position and sizes, for example, to get two spectra, you would

- **have** –

  spec_info = [[(p1, p2), (s1, s2)],
  [(p3, p4), (s3, s4)]]

Returns
A list of apertures and a list of spectra.

Return type
tuple(array)

cubespa.spectra.**spectra_comparison**(*cubecomp*, *aper_list*, *outname=None*, *plot_ticks=True*, *chan_ranges=None*)

cubespa.utils.**H2_Mass**(*SCO*, *D_L=100.0*, *z_gal=0.01*, *freq=220.0*, *a_CO=3.2*, *R_21=0.8*)

Calculate the H2 mass from a CO(2-1) emission map

Parameters

- **SCO** (*float*) – The integrated CO(2-1) flux (in Jy km/s)
- **D_L** (*float, optional*) – The luminosity distance (in MPC). Defaults to 100.
- **z_gal** (*float, optional*) – The redshift of the galaxy. Defaults to 0.01.
- **freq** (*float, optional*) – The observing frequency (in GHz). Defaults to 220.
- **a_CO** (*float, optional*) – The CO conversion factor (in solar masses per square pc). Defaults to 3.2.
- **R_21** (*float, optional*) – The CO(2-1)/(1-0) ratio. Defaults to 0.8 (from Leroy et al, 2009).

cubespa.utils.**RMS**(*a*, *sigclip=False*)

cubespa.utils.**beam_area**(*bmaj*, *bmin*)

Get the area of the beam based on a standard elliptical Gaussian beam with major and
minor axes bmaj and bmin.

Parameters

- **bmaj** (*float*) – Major axis of beam
- **bmin** (*float*) – Minor axis of beam

Returns
The beam area.

Return type
float

cubespa.utils.**bounds_from_moment_map**(*data*, *padding=0*)

cubespa.utils.**centre_coords**(*input_wcs*, *ra*, *dec*)

cubespa.utils.**check_and_make_dir**(*directory*)

cubespa.utils.**check_kwarg**(*key*, *default*, *kwargs: dict*)

cubespa.utils.**create_channel_ranges**(*n*)

Create an array of None values of length n (to initialize channel ranges for spectral analysis)

cubespa.utils.**estimate_rms**(*cube*, *channel_min*, *channel_max*)

cubespa.utils.**im_bounds**(*stats*, *sigma=1*)

cubespa.utils.**imstat**(*a*)

Simple tool to return statistics for a given array.

Parameters
a (*ndarray*) – Input array to get statistics.

Returns
description

Return type
type

cubespa.utils.**line_corners**(*x0*, *y0*, *w*, *L*, *theta*)

cubespa.utils.**line_endpoints**(*x0*, *y0*, *L*, *theta*)

cubespa.utils.**match_wcs_axes**(*wcs1*, *wcs2*)

Match the axes in WCS axes (mostly for image alignment with cubes and images).
This method assumes that the ra/dec axes are always at indices 0 and 1.

Parameters

- **wcs1** (*astropy.wcs*) – WCS A
- **wcs2** (*astropy.wcs*) – WCS B

Returns
Both WCS objects.

Return type
type

cubespa.utils.**normalize**(*a*, *clip_low=None*, *clip_high=None*, *stretch=None*)

cubespa.utils.**normalized_rgb_image**(*image*, *sigma=1*, *stretch=None*)

Generate a properly formatted RGB image from a 3xmxn input.

Parameters

- **image** (*ndarray*) – 3 x m x n input image.
- **sigma** (*int, float, or array optional*) – Sigma levels to adjust stretches and clips. Defaults to 1. int: sigma for all 3 RGB images tuple: All 3 images clipped to (sigma[0], sigma[1]) Array of len(3): R,G,B images clipped to sigma[0], sigma[1], sigma[2], respectively.
- **stretch** (*str, optional*) – Type of stretch to apply. Defaults to None.

Returns
The properly transposed, stretched and clipped RGB image.

Return type
ndarray

cubespa.utils.**recommended_figsize**(*a*, *width=8*)

5.1.2 Plotting

cubespa.plotting.channel_map_plots.**channel_maps**(*cubespa_obj*, *n_chan=25*, *limits=None*, ***kwargs*)

cubespa.plotting.mommap_plots.**moment_map_plot**(*cubespa_obj*, *filename=None*, *use_limits=True*, ***kwargs*)

Generate moment map plots.

Parameters

- **cubespa_obj** (cubespa.CubeSPA) – The input CubeSPA object, with valid moment maps loaded.
- **filename** (*str, optional*) – Output filename, in which the plot is just shown instead of saved. Defaults to None.
- **use_limits** (*bool, optional*) – Whether or not to use limits from the CubeSPA object.

  It is a good idea to set to False for cutout objects, as their limits will be relative to the initial CubeSPA object, and their desired limits will be the entire array. Defaults to True.

cubespa.plotting.overlay_plots.**overlay_plot**(*img_obj*, *overlay_obj*, *lims=None*, ***kwargs*)

cubespa.plotting.overlay_plots.**plot_psf_overlay**(*cubespa_obj*, *psf_conv=None*, *x0=0*, *y0=0*, ***kwargs*)

cubespa.plotting.overlay_plots.**rgb_overlay**(*rgb_img*, *overlay_obj*, *lims=None*, *levels=None*, *colors=None*, *filename=None*, ***kwargs*)

cubespa.plotting.spectra_plots.**multispec_plot**(*cubespa_obj*, *aper_list*, *spec_list*, ***kwargs*)

cubespa.plotting.spectra_plots.**plot_spectra**(*data*, *aper*)

cubespa.plotting.spectra_plots.**spectra_comparison**(*cubecomp*, *a1*, *a2*, *s1*, *s2*, *chan_ranges=None*, ***kwargs*)

cubespa.plotting.spectra_plots.**spectra_plot**(*cubespa_obj*, *aper*, *spectrum*)

Create a plot showing both the image with overlaid spectra, as well as the spectrum with RMS levels shown.

Parameters

- **cubespa_obj** (cubespa.CubeSPA) – CubeSPA object.
- **aper** (*photutils.aperture*) – Input aperture, generated using cubespa.spectra
- **spectrum** (*_type_*) – _description_

cubespa.plotting.util_plots.**limit_plot**(*cubespa_obj*)

cubespa.plotting.util_plots.**plot_bbox**(*cubespa_obj*, *lims*)

cubespa.plotting.util_plots.**plot_rgb**(*rgb*, *lims=None*, *outname=None*)

Plot an RGB image using matplotlib

Parameters

- **rgb** (*nxmx3 array*) – RGB image formatted for matplotlib
- **lims** (*arr, optional*) – x and y limits for plotting. Defaults to None.

- **outname** (*str, optional*) – Output filename. If not, show plot instead of save figure. Defaults to None.

- genindex
- modindex
- search

PYTHON MODULE INDEX

C

cubespa.cubespa, 23
cubespa.data, 23
cubespa.plotting.channel_map_plots, 27
cubespa.plotting.mommap_plots, 10
cubespa.plotting.overlay_plots, 27
cubespa.plotting.spectra_plots, 18
cubespa.plotting.util_plots, 27
cubespa.spectra, 18
cubespa.utils, 25

INDEX

A

align_apertures() (*in module cubespa.spectra*), 18, 24
analyze_spectra() (*in module cubespa.spectra*), 18, 24

B

beam_area() (*in module cubespa.utils*), 25
bounds_from_moment_map() (*in module cubespa.utils*), 25

C

calc_snr() (*in module cubespa.spectra*), 18, 24
centre_coords() (*in module cubespa.utils*), 25
channel_maps() (*in module cubespa.plotting.channel_map_plots*), 27
check_and_make_dir() (*in module cubespa.utils*), 25
check_kwarg() (*in module cubespa.utils*), 25
create_aperture() (*in module cubespa.spectra*), 18, 24
create_channel_ranges() (*in module cubespa.utils*), 25
create_spectra() (*cubespa.cubespa.CubeComparison method*), 23
create_spectra() (*cubespa.cubespa.CubeSPA method*), 23
CubeComparison (*class in cubespa.cubespa*), 23
CubeSPA (*class in cubespa.cubespa*), 23
cubespa.cubespa
    module, 23
cubespa.data
    module, 23
cubespa.plotting.channel_map_plots
    module, 27
cubespa.plotting.mommap_plots
    module, 10, 27
cubespa.plotting.overlay_plots
    module, 27
cubespa.plotting.spectra_plots
    module, 18, 27
cubespa.plotting.util_plots
    module, 27
cubespa.spectra
    module, 18, 24
cubespa.utils
    module, 25

D

DataSet (*class in cubespa.data*), 23

E

estimate_rms() (*in module cubespa.utils*), 25

G

gen_cutout() (*in module cubespa.data*), 23
get_beam_area() (*cubespa.cubespa.CubeSPA method*), 23
get_spectra() (*in module cubespa.spectra*), 19, 24

H

H2_Mass() (*in module cubespa.utils*), 25
handle_data() (*in module cubespa.data*), 23

I

im_bounds() (*in module cubespa.utils*), 26
imstat() (*in module cubespa.utils*), 26

L

limit_plot() (*in module cubespa.plotting.util_plots*), 27
line_corners() (*in module cubespa.utils*), 26
line_endpoints() (*in module cubespa.utils*), 26
load_data() (*in module cubespa.data*), 24
load_dir() (*cubespa.cubespa.CubeSPA method*), 23
load_moment_maps() (*in module cubespa.data*), 24

M

match_wcs_axes() (*in module cubespa.utils*), 26
module
    cubespa.cubespa, 23
    cubespa.data, 23
    cubespa.plotting.channel_map_plots, 27

cubespa.plotting.mommap_plots, 10, 27
cubespa.plotting.overlay_plots, 27
cubespa.plotting.spectra_plots, 18, 27
cubespa.plotting.util_plots, 27
cubespa.spectra, 18, 24
cubespa.utils, 25
moment_map_plot() (*in module cubespa.plotting.mommap_plots*), 10, 27
MomentMaps (*class in cubespa.data*), 23
multi_spec() (*in module cubespa.spectra*), 19, 24
multispec_plot() (*in module cubespa.plotting.spectra_plots*), 18, 27

N

normalize() (*in module cubespa.utils*), 26
normalized_rgb_image() (*in module cubespa.utils*), 26

O

overlay_plot() (*in module cubespa.plotting.overlay_plots*), 27

P

plot_bbox() (*in module cubespa.plotting.util_plots*), 27
plot_moment_maps() (*cubespa.cubespa.CubeSPA method*), 23
plot_psf_overlay() (*in module cubespa.plotting.overlay_plots*), 27
plot_rgb() (*in module cubespa.plotting.util_plots*), 27
plot_spectra() (*in module cubespa.plotting.spectra_plots*), 18, 27

R

readout() (*cubespa.cubespa.CubeComparison method*), 23
recommended_figsize() (*in module cubespa.utils*), 26
rgb_overlay() (*in module cubespa.plotting.overlay_plots*), 27
RGBImage (*class in cubespa.data*), 23
RMS() (*in module cubespa.utils*), 25

S

spectra_comparison() (*in module cubespa.plotting.spectra_plots*), 18, 27
spectra_comparison() (*in module cubespa.spectra*), 19, 25
spectra_plot() (*in module cubespa.plotting.spectra_plots*), 18, 27

V

velocities_from_wcs() (*cubespa.cubespa.CubeSPA method*), 23