CubeSPA

Harrison Souchereau

Dec 30, 2023

CONTENTS:

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(cube_fn)
```

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(cube_fn)
```

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")
```

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.

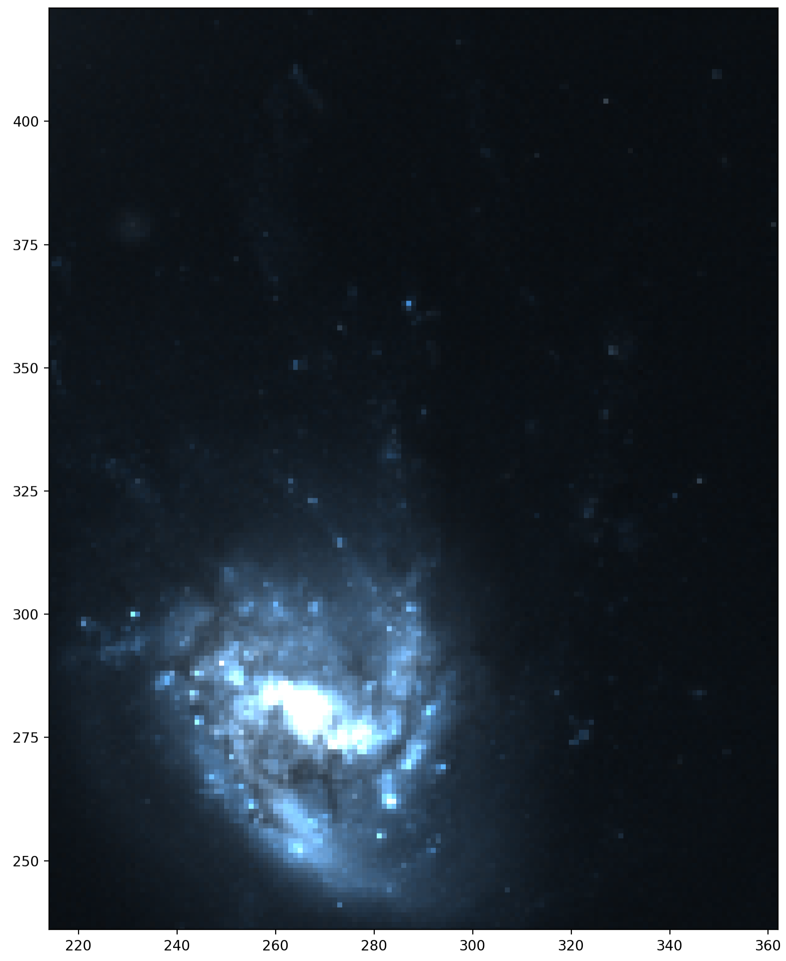
For example, if we load in an hst image and then create an aligned image, we can see what it looks like by default with the following code:

```
>>> 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")
```

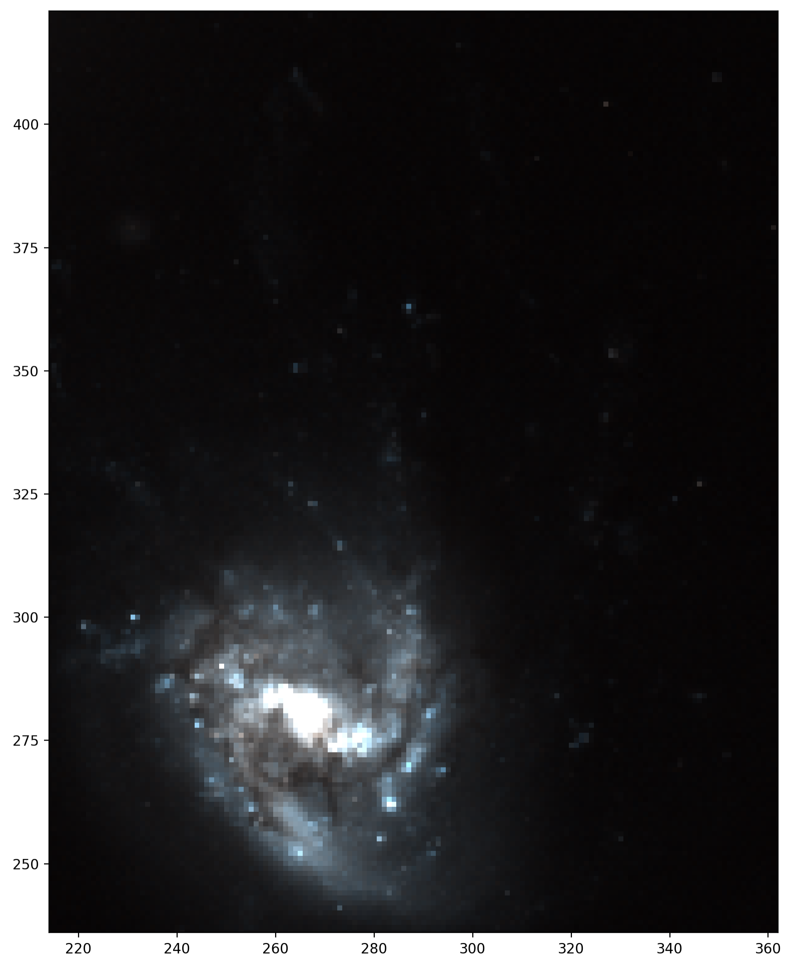
This returns the following.



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

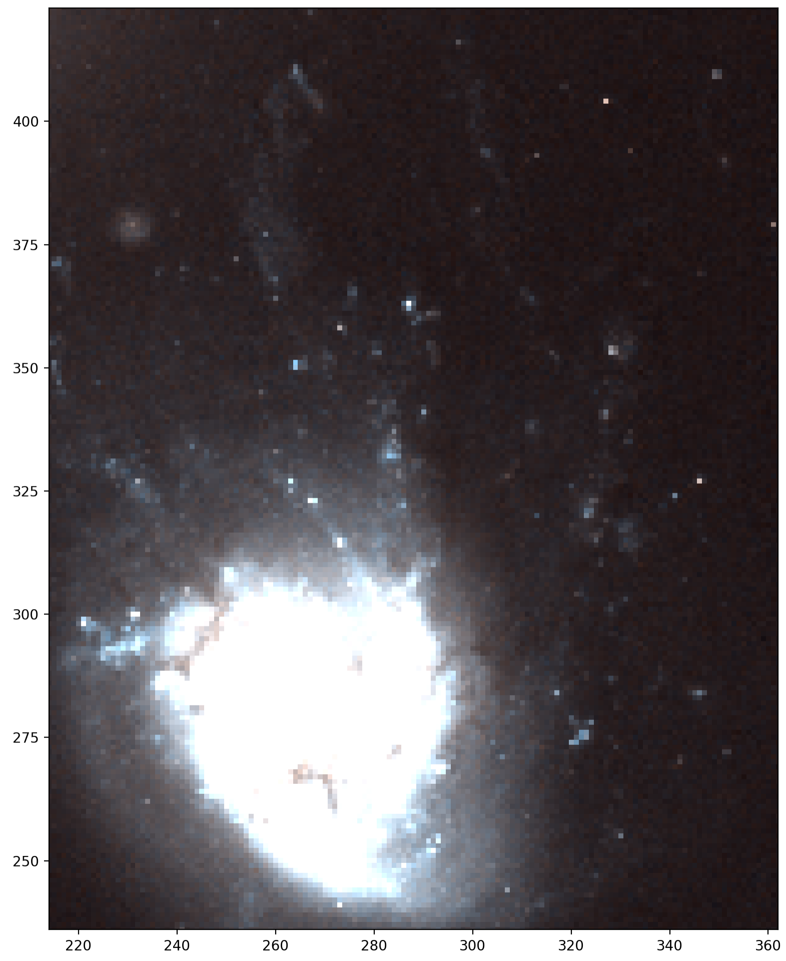
```
>>> norm = cubespa.normalized_rgb_image(hst_interp.data, sigma=(2, 5))
>>> cubespa.plotting.plot_rgb(norm, lims=np.array(c.limits))
```

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.

1.1.5 Reference/API

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

Bases: `object`

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

load_dir()

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

velocities_from_wcs()

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

CHAPTER

TWO

2D ANALYSIS

2.1 Generating Moment Maps

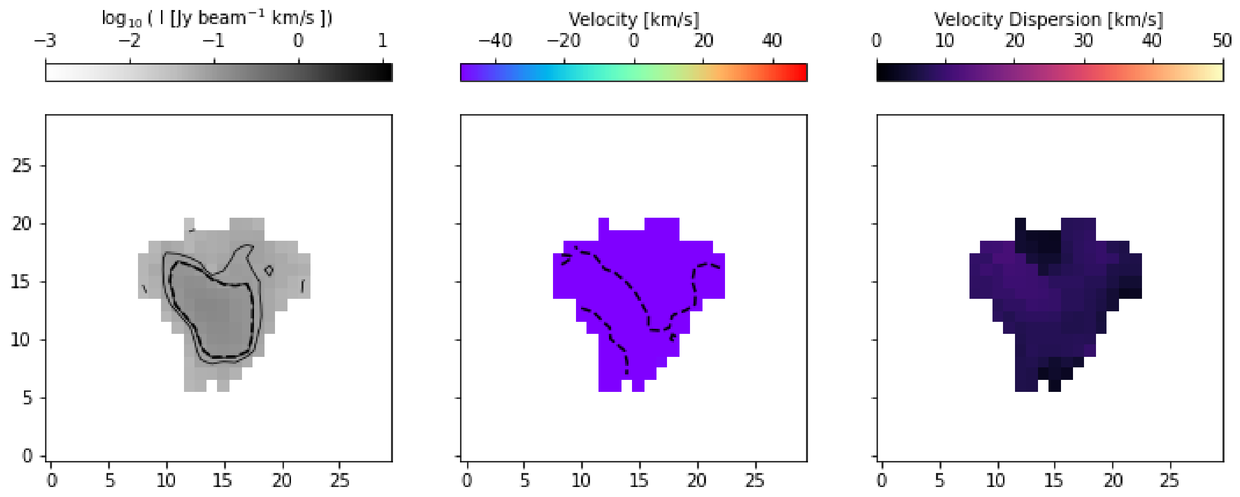
Generating moment maps is trivially easy using CubeSPA.

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*`)`

summary

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.

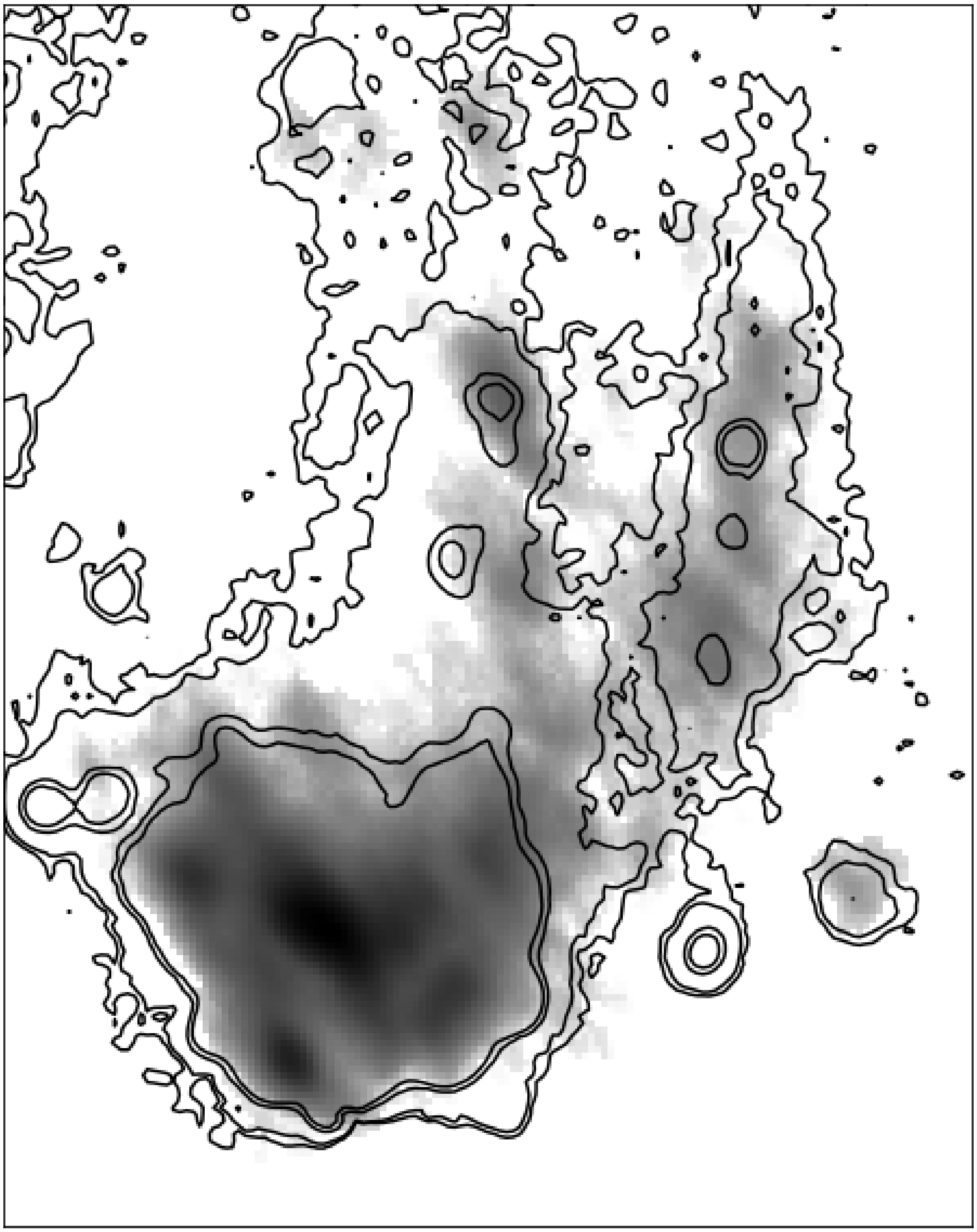
2.2 Making Overlays

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”.

CHAPTER

THREE

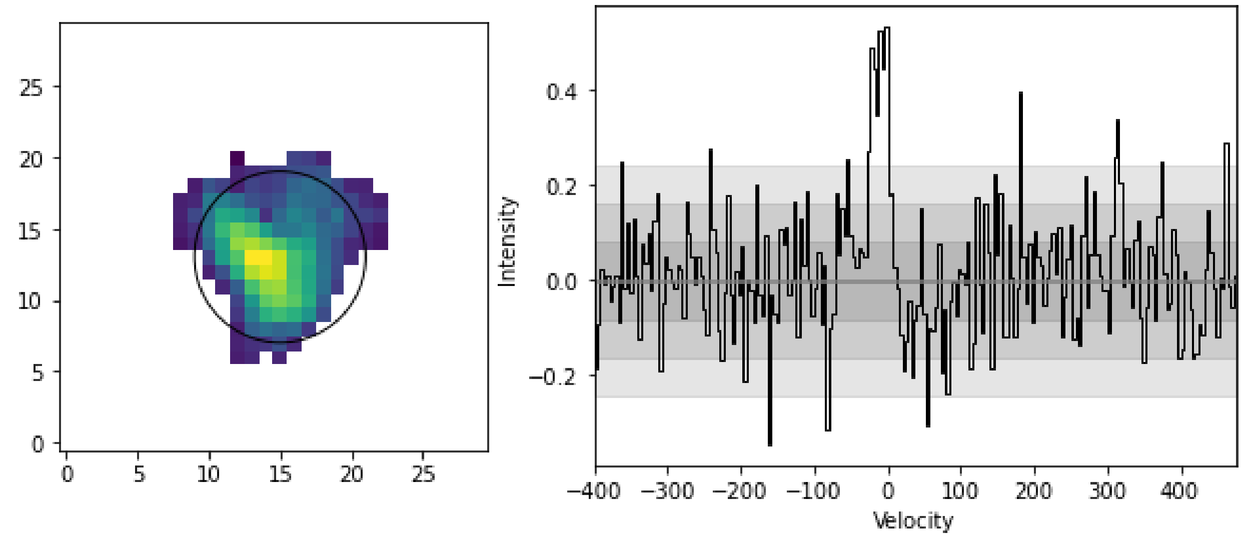
CUBE ANALYSIS

3.1 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.



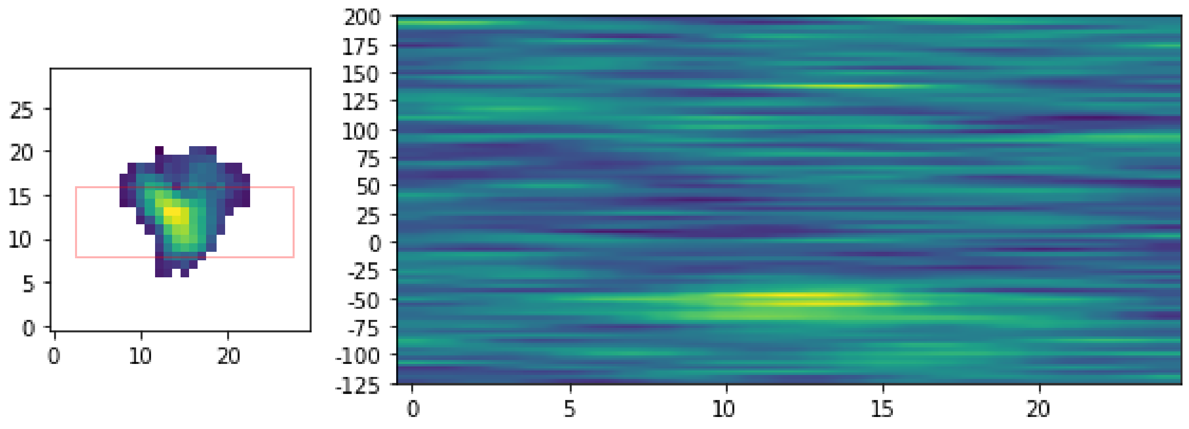
3.2 Position-Velocity diagrams

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.



CHAPTER

FOUR

COMING SOON

4.1 Generating / Fitting Velocity Models

CHAPTER FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX