# pvextractor Documentation

*Release 0.0.dev289*

**Adam Ginsburg and Thomas Robitaille**

**Sep 12, 2017**

# Contents

The concept of the `pvextractor` package is simple - given a path defined in sky coordinates, and a spectral cube, extract a slice of the cube along that path, and along the spectral axis, producing a position-velocity or position-frequency slice.

The path can be defined programmatically in pixel or world coordinates, but it can also be drawn interactively using a simple GUI. We also provide a DS9 analysis plug-in that allows the path to be drawn in DS9, and the resulting slice shown in a new frame in DS9. Finally, the slicing capability is available in Glue.

# Part I

# Extracting slices programmatically

CHAPTER 1

# Defining a path

## Pixel coordinates

To define a path in pixel coordinates, import the `Path` class:

```
>>> from pvextractor import Path
```

Then initialize it using a list of `(x,y)` tuples. The simplest path that can be defined is a line connecting two points:

```
>>> path1 = Path([(0., 0.), (10., 10.)])
```

Multi-segment paths can also be similarly defined:

```
>>> path2 = Path([(0., 0.), (4., 6.), (10., 10.)])
```

By default, slices are extracted using interpolation along the line, but it is also possible to define a path with a finite width, and to instead measure the average flux or surface brightness in finite polygons (rather than strictly along the line). To give a path a non-zero width, simply use the `width=` argument, which is also in pixels by default:

```
>>> path3 = Path([(0., 0.), (10., 10.)], width=0.5)
```

## World coordinates

To define a path in world coordinates, pass a coordinate array to the `Path` object. In addition, the width (if passed) should an Astropy `Quantity` object:

```
>>> from astropy import units as u
>>> from astropy.coordinates import Galactic
>>> g = Galactic([3.4, 3.6] * u.deg, [0.5, 0.56] * u.deg)
>>> path4 = Path(g, width=1 * u.arcsec)
```

In additon to the `Path` class, we provide a convenience `PathFromCenter` class that can be used for cases where the center and position angle of the path are known (rather than the end points of the path). This class is used as follows:

```
>>> from pvextractor import PathFromCenter
>>> from astropy import units as u
>>> from astropy.coordinates import Galactic
>>> g = Galactic(3 * u.deg, 5 * u.deg)
>>> path5 = PathFromCenter(center=g,
...                        length=1 * u.arcmin,
...                        angle=30 * u.deg,
...                        width=1 * u.arcsec)
```

The position angle is defined counter-clockwise from North, and the direction of the path is such that for a position angle of zero, the path is defined from South to North.

CHAPTER 2

# Extracting a slice

Once the path has been defined, you can make use of the `extract_pv_slice()` function to extract the PV slice. The data to slice can be passed to this function as:

- A 3-d Numpy array
- A `SpectralCube` instance
- An HDU object containing a spectral cube
- The name of a FITS file containing a spectral cube

For example:

```
>>> from pvextractor import extract_pv_slice
>>> slice1 = extract_pv_slice(array, path1)

>>> from spectral_cube import SpectralCube
>>> cube = SpectralCube.read('my_cube.fits')
>>> slice2 = extract_pv_slice(cube, path3)

>>> slice3 = extract_pv_slice('my_cube.fits', path3)
```

**Note:** If a path is passed in in world coordinates, and the data are passed as a plain Numpy array, the WCS information should be passed as a `WCS` object to the `wcs=` argument.

# CHAPTER 3

## Saving the slice

The returned slice is an Astropy `PrimaryHDU` instance, which you can write to disk using:

```
>>> slice1.writeto('my_slice.fits')
```

# Part II

# Using the built-in graphical user interface

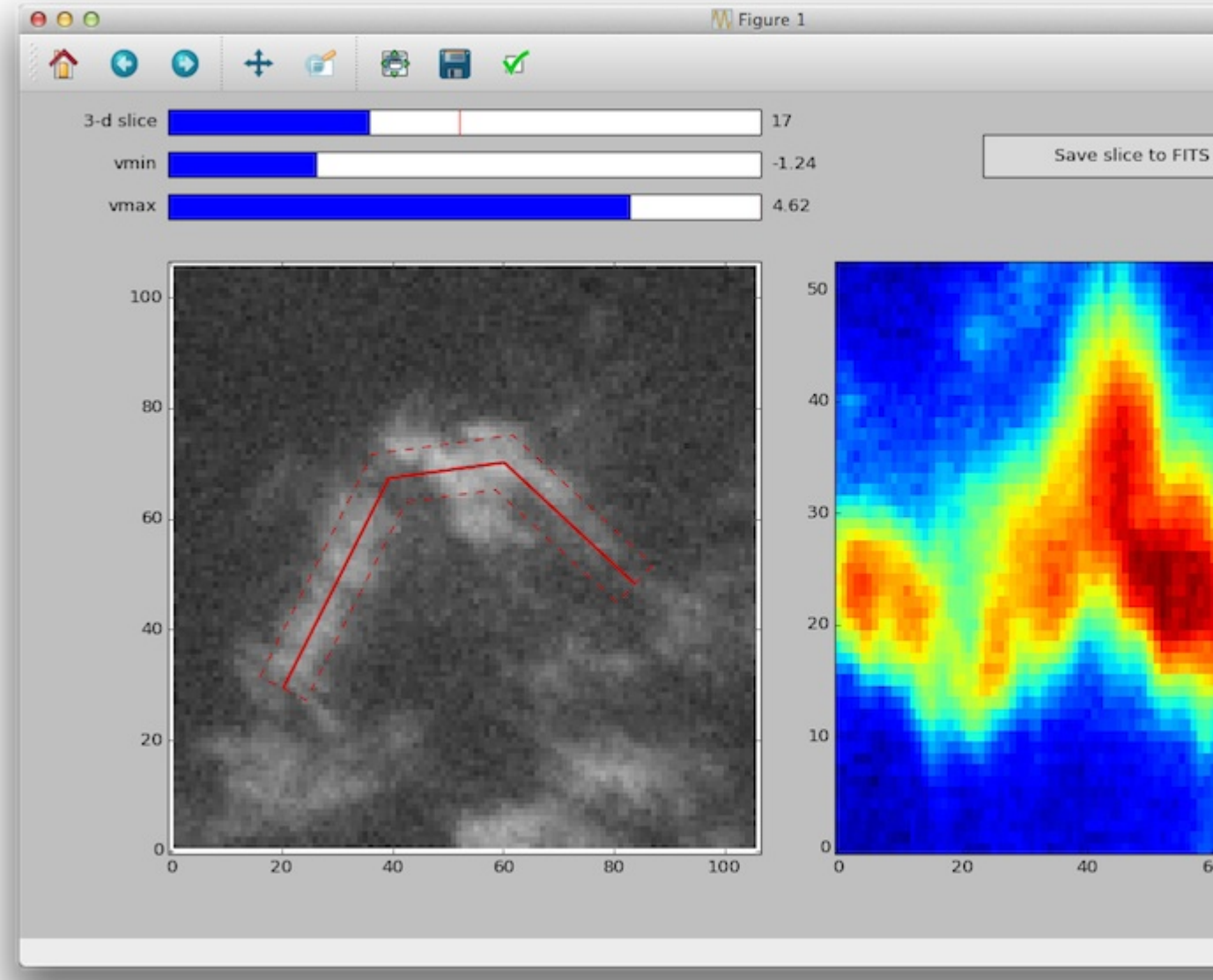
The extractor GUI provdes the most direct interface available to the pixel-matched version of the position-velocity extractor. It is simple to initialize:

```
from pvextractor.gui import PVSlicer
pv = PVSlicer('cube.fits')
pv.show()
```

A window will pop up showing the data on the left hand side. Click to draw the vertices of the path, then press "enter" to expand the width of the slice, then move your mouse away from the path to define how wide it should be. Once you are happy with the width, click, and the slice will be computed and shown on the right.

Once a path has been defined, you can optionally press "y" to see the polygons along which the data has been collapsed to compute the slice.

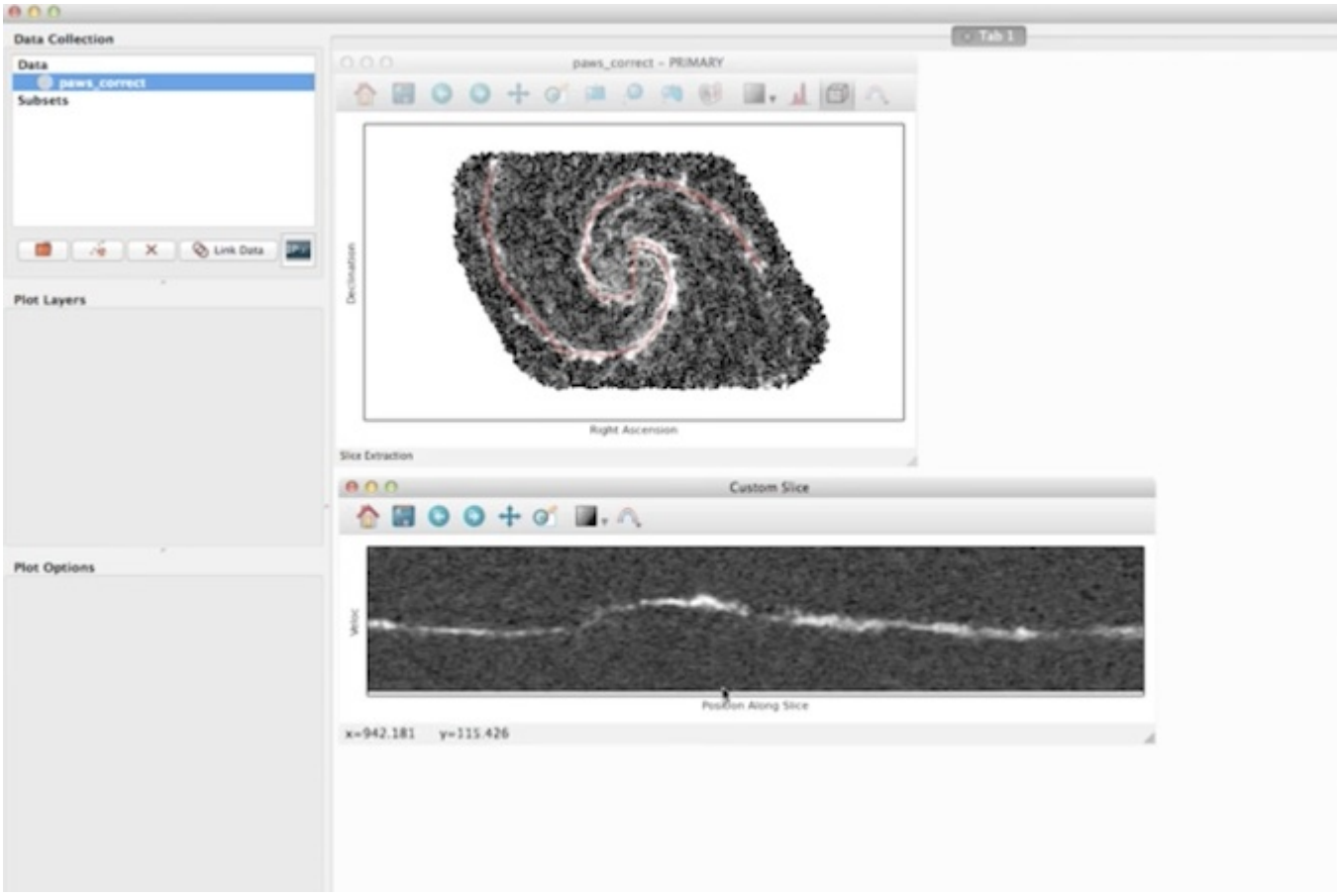
The following shows an example of a slice derived from a 13CO cube of L1448:



# Part III

# Slicing in Glue

Glue has a position-velocity diagram extraction tool that allows arbitrary paths to be specified:




A beautiful demo shows this in action (movie).

# Part IV

# Slicing in DS9

**Note:** This feature is experimental and does not work with all versions of DS9 at this point.

There is a python command-line script that will be installed into your path along with `pvextractor`. You can invoke it from the command line, but the preferred approach is to load the tool into DS9. First, determine the path to `ds9_pvextract.ans`; it is in the `scripts` subdirectory of the source code. Then start up DS9 with the analysis tool loaded

```
ds9 -analysis load /path/to/pvextractor/scripts/ds9_pvextract.ans  &
```

Then load any cube in DS9. You can draw a line, a vector, or a "segment"; only the first one drawn will have any effect. To extract the PV diagram, press the 'x' key or click "PV Extractor" in the analysis menu. Be patient - especially for big cubes, it may take a little while, and there is no progress bar.

# Part V

# API Documentation

CHAPTER 4

# pvextractor Package

This is an Astropy affiliated package.

## Functions

| | |
|---|---|
| extract_pv_slice(cube, path[, wcs, spacing, ...]) | Given a position-position-velocity cube with dimensions (nv, ny, nx), and a path, extract a position-velocity slice. |
| paths_from_regfile(regfile) | Given a ds9 region file, extract pv diagrams for each: |
| slice_wcs(wcs, spatial_scale) | Slice a WCS header for a spectral cube to a Position-Velocity WCS, with |
| test([package, test_path, args, plugins, ...]) | Run the tests using py.test. |

### extract_pv_slice

pvextractor.**extract_pv_slice**(*cube*, *path*, *wcs=None*, *spacing=1.0*, *order=3*, *respect_nan=True*)

Given a position-position-velocity cube with dimensions (nv, ny, nx), and a path, extract a position-velocity slice.

**Alternative implementations:**

gipsy::sliceview karma::kpvslice casaviewer::slice

**Parameters**

**cube** : ndarray or SpectralCube or str or HDU

The cube to extract a slice from. If this is a plain ndarray instance, the WCS information can optionally be specified with the wcs parameter. If a string, it should be the name of a file containing a spectral cube.

**path** : Path or list of 2-tuples

The path along which to define the position-velocity slice. The path can contain coordinates defined in pixel or world coordinates.

**wcs** : WCS, optional

The WCS information to use for the cube. This should only be specified if the cube parameter is a plain ndarray instance.

**spacing** : float

The position resolution in the final position-velocity slice. This can be given in pixel coordinates or as a Quantity instance with angle units.

**order** : int, optional

Spline interpolation order when using paths with zero width. Does not have any effect for paths with a non-zero width.

**respect_nan** : bool, optional

If set to False, NaN values are changed to zero before computing the slices. If set to True, in the case of line paths a second computation is performed to ignore the NaN value while interpolating, and set the output values of NaNs to NaN.

**Returns**

**slice** : PrimaryHDU

The position-velocity slice, as a FITS HDU object

## paths_from_regfile

pvextractor.**paths_from_regfile**(*regfile*)

**Given a ds9 region file, extract pv diagrams for each:**

group of points [NOT IMPLEMENTED] panda [NOT IMPLEMENTED] vector [NOT IMPLEMENTED] segment [NOT IMPLEMENTED] group of lines [NOT IMPLEMENTED]

## slice_wcs

pvextractor.**slice_wcs**(*wcs*, *spatial_scale*)

Slice a WCS header for a spectral cube to a Position-Velocity WCS, with ctype "OFFSET" for the spatial offset direction

**Parameters**

**wcs** : WCS

The WCS of the spectral cube. This should already be sanitized and have the spectral axis along the third dimension.

**spatial_scale: :class:‘~astropy.units.Quantity‘**

The spatial scale of the position axis

**Returns**

wcs_slice WCS

The resulting WCS slice

## test

pvextractor.**test**(*package=None*, *test_path=None*, *args=None*, *plugins=None*, *verbose=False*, *pastebin=None*, *remote_data=False*, *pep8=False*, *pdb=False*, *coverage=False*, *open_files=False*, ***kwargs*)

Run the tests using py.test. A proper set of arguments is constructed and passed to pytest.main.

**Parameters**

**package** : str, optional

The name of a specific package to test, e.g. 'io.fits' or 'utils'. If nothing is specified all default tests are run.

**test_path** : str, optional

Specify location to test by path. May be a single file or directory. Must be specified absolutely or relative to the calling directory.

**args** : str, optional

Additional arguments to be passed to pytest.main in the `args` keyword argument.

**plugins** : list, optional

Plugins to be passed to pytest.main in the `plugins` keyword argument.

**verbose** : bool, optional

Convenience option to turn on verbose output from py.test. Passing True is the same as specifying `'-v'` in `args`.

**pastebin** : {'failed','all',None}, optional

Convenience option for turning on py.test pastebin output. Set to `'failed'` to upload info for failed tests, or `'all'` to upload info for all tests.

**remote_data** : bool, optional

Controls whether to run tests marked with @remote_data. These tests use online data and are not run by default. Set to True to run these tests.

**pep8** : bool, optional

Turn on PEP8 checking via the pytest-pep8 plugin and disable normal tests. Same as specifying `'--pep8 -k pep8'` in `args`.

**pdb** : bool, optional

Turn on PDB post-mortem analysis for failing tests. Same as specifying `'--pdb'` in `args`.

**coverage** : bool, optional

Generate a test coverage report. The result will be placed in the directory htmlcov.

**open_files** : bool, optional

Fail when any tests leave files open. Off by default, because this adds extra run time to the test suite. Requires the psutil package.

**parallel** : int, optional

When provided, run the tests in parallel on the specified number of CPUs. If parallel is negative, it will use the all the cores on the machine. Requires the pytest-xdist plugin installed. Only available when using Astropy 0.3 or later.

**kwargs**

Any additional keywords passed into this function will be passed on to the astropy test runner. This allows use of test-related functionality implemented in later versions of astropy without explicitly updating the package template.

# Classes

| | |
|---|---|
| Path(xy_or_coords[, width]) | A curved path that may have a non-zero width and is used to extract slices from cubes. |
| PathFromCenter(center[, length, angle, ...]) | A simple path defined by a center, length, and position angle. |

## Path

**class** pvextractor.**Path**(*xy_or_coords*, *width=None*)

Bases: object

A curved path that may have a non-zero width and is used to extract slices from cubes.

**Parameters**

**xy_or_coords** : list or Astropy coordinates

The points defining the path. This can be passed as a list of (x, y) tuples, which is interpreted as being pixel positions, or it can be an Astropy coordinate object containing an array of 2 or more coordinates.

**width** : None or float or Quantity

The width of the path. If coords is passed as a list of pixel positions, the width should be given (if passed) as a floating-point value in pixels. If coords is a coordinate object, the width should be passed as a Quantity instance with units of angle. If None, interpolation is used at the position of the path.

### Methods Summary

| | |
|---|---|
| add_point(xy_or_coord) | Add a point to the path |
| get_xy([wcs]) | Return the pixel coordinates of the path. |
| sample_points(spacing[, wcs]) | |
| sample_points_edges(spacing[, wcs]) | |
| sample_polygons(spacing[, wcs]) | |
| to_patches(spacing[, wcs]) | |

### Methods Documentation

**add_point**(*xy_or_coord*)

Add a point to the path

**Parameters**

**xy_or_coord** : tuple or Astropy coordinate

A tuple (x, y) containing the coordinates of the point to add (if the path is defined in pixel space), or an Astropy coordinate object (if it is defined in world coordinates).

**get_xy**(*wcs=None*)
Return the pixel coordinates of the path.

If the path is defined in world coordinates, the appropriate WCS transformation should be passed.

**Parameters**
**wcs** : WCS

The WCS transformation to assume in order to transform the path to pixel coordinates.

**sample_points**(*spacing*, *wcs=None*)

**sample_points_edges**(*spacing*, *wcs=None*)

**sample_polygons**(*spacing*, *wcs=None*)

**to_patches**(*spacing*, *wcs=None*, ***kwargs*)

## PathFromCenter

**class** pvextractor.**PathFromCenter**(*center*, *length=None*, *angle=None*, *sample=2*, *width=None*)
Bases: pvextractor.Path

A simple path defined by a center, length, and position angle.

**Parameters**
**center** : SkyCoord

The center of the path

**length** : Quantity

The length of the path in angular units

**angle** : Quantity

The position angle of the path, counter-clockwise

**sample** : int

How many points to sample along the path. By default, this is 2 (the two end points. For small fields of view, this will be a good approximation to the path, but for larger fields of view, where spherical distortions become important, this should be increased to provide a smooth path.

**width** : None or float or Quantity

The width of the path. If coords is passed as a list of pixel positions, the width should be given (if passed) as a floating-point value in pixels. If coords is a coordinate object, the width should be passed as a Quantity instance with units of angle. If None, interpolation is used at the position of the path.

### Notes

The orientation of the final path will be such that for a position angle of zero, the path goes from South to North. For a position angle of 90 degrees, the path will go from West to East.

# Python Module Index

## p

# Index