Draft version January 31, 2022
Typeset using LaTeX default style in AASTeX631

Radio-astro-tools

Eric Koch, Adam Ginsburg, Tom Robitaille, Erik Rosolowsky

ABSTRACT

We present the `radio-astro-tools` code suite, which consists of several Python packages that enable analysis of radio data, especially interferometric spectral cubes, in the context of the Astropy software ecosystem. While these tools were designed with radio data in mind, they are built to be general, and have applications on data sets at other wavelengths. The core package, `spectral-cube`, handles reading and writing and analysis of cube data, and it enables straightforward parallelization via `dask` and `joblib` backends. Support packages include `casa-formats-io` and `radio-beam`, which handle reading of CASA tables & images and reading and manipulation of point spread functions, respectively. The `pvextractor` package facilitates creation of position-velocity diagrams. The `uvcombine` package implements the 'feather' algorithm for combining single-dish and interferometric data. The development of `radio-astro-tools` included several contributions to other repositories, including `matplotlib`, `astropy`, and `astropy-regions` to support interaction between CASA and other parts of the astronomy software ecosystem.

1. INTRODUCTION

Astronomers often self-identify with a specific wavelength, but multiwavelength astronomy has become commonplace, especially in the era of queue observing in which specialization in observing in a given wavelength regime is not needed to obtain data. Space telescopes and radio interferometers have been operating in the mode of delivering nearly-final data products to astronomers for a long time. The delivery of data from different wavelengths with different instruments but common assumptions about locations on the sky and energy of the photons drove the development of standardized world coordinate systems (Greisen et al. 2006; Calabretta & Greisen 2002).

1.1. *Relationship to CASA*

CASA (McMullin et al. 2007) is the core package for radio astronomy data reduction and analysis, having supplanted AIPS in the early 2000's for VLA data and always being the main tool for ALMA. In addition to the interferometric data processing features uniquely available in CASA, it contains many tools for cube and image analysis.

2. THE PACKAGES

radio-astro-tools is comprised of several python packages: the main spectral-cube package and several light-weight packages that support specific operations or visualization for radio data. In this section, we describe the purpose and capabilities of each package. We provide a guide for translating between operations in CASA and radio-astro-tools: https://github.com/radio-astro-tools/tutorials/tree/master/casa_to_spectralcube_guide.

2.1. *`spectral-cube`*

`spectral-cube` is the primary radio-astro-tools package that enables fast and flexible I/O operations with most common analysis methods backed by the astropy ecosystem. Additionally, a primary goal of `spectral-cube` is seamless handling of larger-than-memory spectral data cubes.

Users primarily interact with the 'SpectralCube' class, which handles reading different data formats and types of spectral cubes. For example, `spectral-cube` handles data cubes with a varying spectral resolution (i.e., the beam varies between each spectral channel) as is expected for wide-bandwidth coverage and spectral scan data. Basic handling for 4D cubes with a polarization axis is available in the current version with additional capabilities planned.

Finally, we highlight that spectral-cube has a thorough testing suite to check all operations in the package. Users can access further information and documentation at https://spectral-cube.readthedocs.io.

We present an overview for how to integrate spectral-cube into new packages (§3), present tutorials with worked examples (§4), and describe further uses of spectral-cube with the accompanying packages described below (§5).

`spectral-cube` provides easy handling for common operations and analyses of spectral data cubes. This includes:

- Basic arithmetic between cubes with matching WCS coordinates.
- Signal and noise masking. The masking framework is flexible and includes new mask creation, mask combination, and passing/combining with pre-made masks. Operations such as sigma-masking can also be used.
- Spatial and spectral region extraction using pixel indices, WCS coordinates, and DS9/CRTF defined regions.
- Moment map calculation (zeroth, first, second, Nth moments) and other common projection methods including the maximum, minimum, and spectral value at max/min brightness (e.g., velocity at peak temperature).
- Spectra and projection visualization.
- Spatial convolution to a set Gaussian beam size (using `radio-beam`; §2.2), median smoothing and smoothing to an arbitrary kernel type using `astropy.convolution`.
- Spectral smoothing with `astropy.convolution` kernels and resampling to a new spectral axis.
- Spatial and spectral reprojection to a given FITS header using the `reproject` package.
- Integration with visualization packages including glue[1] (Robitaille et al. 2017), yt[2] (Turk et al. 2011) and DS9[3] (Smithsonian Astrophysical Observatory 2000), with future plans to integrate with CARTA[4] (Comrie et al. 2021).

These operations are implemented to avoid reading in the entire data cube and compute in chunks when handling large data cubes. For operations where this is not easily done, users are warned of operations that could exceed memory limits and given the option to disable this behavior.

2.2. *radio-beam*

Interferometric data is generally imaged using a variant of the CLEAN algorithm, which produces final images consisting of a model convolved with a synthesized point spread function, generally called the 'clean beam' or 'synthesized beam'. `radio-beam`[5] is a toolkit for working with standard Gaussian clean beams. It can read and write both single beams and per-channel beams for cubes, and includes operations for convolution, deconvolution and unit conversions (e.g., Jy/beam to K). It also includes a general algorithm for determining the smallest common beam for convolution of a data cube to a common resolution by solving for the minimum enclosing ellipse given a set of ellipses. Appendix A provides further details on the algorithm.

`radio-beam` also contains a helper function for adding the beam shape on to matplotlib figures (see Figure 1).

2.3. *pvextractor*

Position-velocity diagrams are two-dimensional slices through three-dimensional data cubes, where the third dimension is spectral and can generally be interpreted, for a given observed emission or absorption line, as doppler-shifted velocity of gas. The `pvextractor` package[6] allows users to produce slices along arbitrary paths. It also allows averaging perpendicular to the selected paths. Slice paths can be specified in pixel or world coordinates and can be produced either programmatically or through the `PVSlicer` GUI. `pvextractor` is incorporated into the `glueviz`[7] package.

2.4. *uvcombine*

Interferometric observations do not fully sample the Fourier plane of the sky; they intrinsically leave a gap at the center of the domain, which referred to as 'short spacing' or the 'DC component'. To fill in the missing short spacings, different observations that are taken with a single filled-aperture telescope ('single-dish') can be combined with interferometric images. CASA provides a tool for the Fourier-space combination of interferometric and single-dish

[1] https://glueviz.org/
[2] https://yt-project.org/
[3] https://ds9.si.edu
[4] https://cartavis.org/
[5] https://github.com/radio-astro-tools/radio-beam, https://radio-beam.readthedocs.io
[6] https://github.com/radio-astro-tools/pvextractor/, https://pvextractor.readthedocs.io
[7] https://glueviz.org/

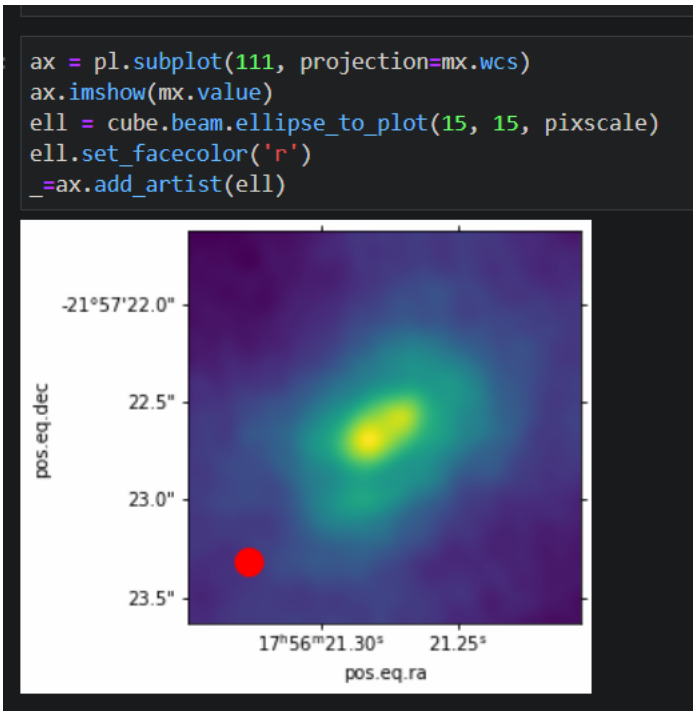

Figure 1. Example of plotting a beam with radio-beam

images in the `feather` task. The `uvcombine`[8] package provides a python-based implementation of the same algorithm. It includes a few additional configurable options and is more flexible about the input data type and the units of the input data.

`uvcombine` also includes tools to estimate the flux scaling factors between the single-dish and interferometric data, as described in Stanimirovic (2002) and Koch et al. (2018, see Appendix A). These tools compare the relative flux at scales that both the data sets are sensitive to, which is necessary to test whether the flux calibrations applied are consistent or whether a correction factor should be applied (more often to the single dish data).

As of February 2022, while `uvcombine` has been confirmed to run successfully and has been tested against CASA feather for one case, it is not yet recommended to use without further tests for correctness.

2.5. *casa-formats-io*

CASA natively writes data into directories that contain tabular data. The CASA format has historically only been readable by `casacore`[9] (Casacore Team 2019), which is a library written in C with bindings available in several other languages. `casacore` does many things besides file reading, though, and can be challenging to install[10].

`casa-formats-io`[11] is a table reader for CASA data formats written in python and c (though all user-facing functions are in python). It is cross-platform, operating on unix, windows, and mac operating systems. It uses `dask` to lazily load the data following CASA's chunking scheme. CASA tables are presented to the user as astropy tables or as dask arrays depending on the function used.

2.6. *statcont*

The `statcont` package was developed to enable continuum estimation from complex "line-forest" data sets (Sánchez-Monge et al. 2018). The original package used only `astropy.io.fits` and required loading the entire data set into memory. We have added a capability to use `statcont` with `spectral-cube`, which enables running `statcont` on cubes

[8] https://uvcombine.readthedocs.io/en/latest/

[9] https://casacore.github.io/casacore/

[10] https://newton.cx/~peter/howto/access-casa-in-python-without-casapy/

[11] https://casa-formats-io.readthedocs.io, https://github.com/radio-astro-tools/casa-formats-io

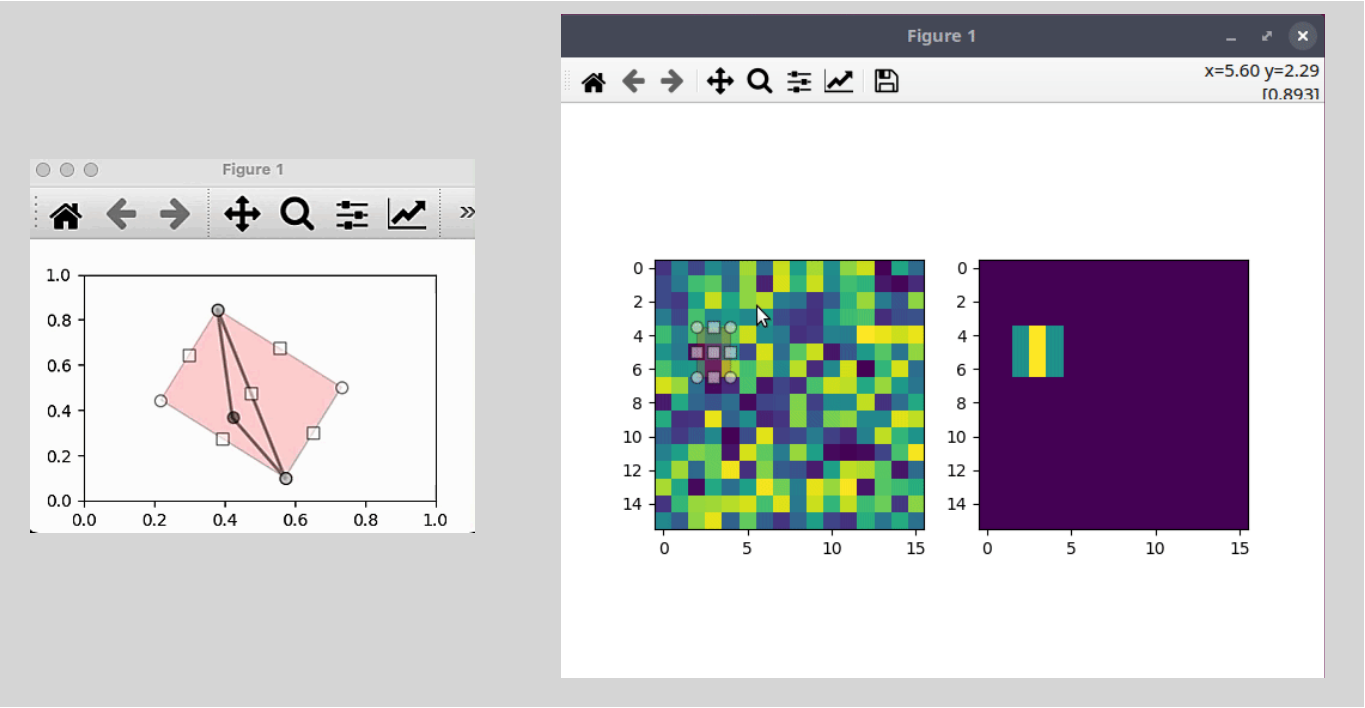

Figure 2. Static images of interactive region drawing with `matplotlib` and `regions`. Left: Square region on a blank canvas (Interactive Version). Right: Square region projected onto an image (Interactive Version). Examples from Derek Homeier and David Stansby.

that are larger than the computer's memory. As part of this process, significant performance enhancements were made to `astropy.stats.sigma_clip`[12].

2.7. *astropy regions*

The `astropy/regions` package[13] is not part of the `radio-astro-tools` grouping, but it includes several tools for interaction with CASA. The CRTF file format (CASA Region Text Format) is used internally by CASA to specify regions of interest, analogous to the popular `ds9` regions, but with some support for selection in more than two dimensions. `regions` implements a reader and writer for the CRTF format, enabling conversion between formats.

One of the motivations to use regions of interest in CASA is to specify "clean boxes" during an interactive image deconvolution run. To support creation of such clean boxes in graphical interfaces, regions can be displayed and modified within `matplotlib` plot windows[14]. Several additional features for manipulating elliptical, rectangular, and polgyon shapes were added to matplotlib as part of the development work to support this feature.

3. USING RADIO-ASTRO-TOOLS FOR DEVELOPMENT OF OTHER PACKAGES

The radio-astro-tools packages are ideally suited to handle I/O operations in other python packages. This is a significant strength given the efficient larger-than-memory data handling in `spectral-cube` and parallelization of operations with the `dask` integration that users can build upon. We include an example use case in our set of tutorials demonstrating how `spectral-cube` and `dask` can parallelize fitting a Gaussian model to every spectrum in an ALMA data cube, loading only small chunks into memory at a time (§4). This example overcomes an often-faced issue for some spectral cube modeling tools, which converted standard cube file formats (FITS, CASA image) to non-standard forms (pickle or text file) to boost performance. `spectral-cube` removes the need for these extra conversion and I/O steps.

Included in the `spectral-cube` documentation[15] is documentation for recommended practices for developers.

[12] https://docs.astropy.org/en/stable/changelog.html#id46

[13] https://astropy-regions.readthedocs.io/en/stable/, https://github.com/astropy/regions

[14] TODO: link to docs

[15] https://spectral-cube.readthedocs.io/en/latest/developing_with_spectralcube.html

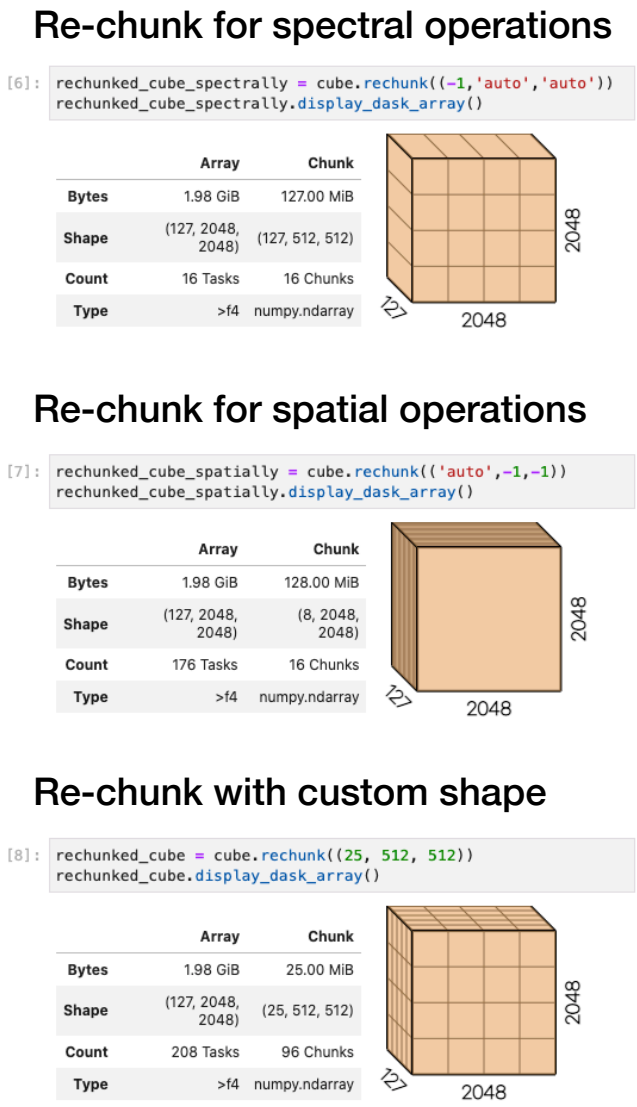

Figure 3. Examples of rechunking a data cube to optimize for spectral and spatial operations (top, middle), and passing a custom chunk shape (bottom) using the `dask` integration in `spectral-cube`.

We note that radio-astro-tools is already used in a number of calibration, imaging and analysis pipelines. Examples include ALMA (e.g., ALMA-IMF[16]; PHANGS[17]; Ginsburg et al. 2021; Leroy et al. 2021), GBT (GAS[18]; DEGAS[19] Friesen et al. 2017, , Kepley et al. in prep.), and ASKAP (e.g., GASKAP; Pingel et al. 2021) large programs, as well as the calibration pipeline for Aptertif (Apercal; Adebahr et al. 2022).

3.1. *glue*

The `glue` visualization and analysis toolkit uses many components of the `radio-astro-tools` package both for reading files and for analysis. The position-velocity extraction in glue directly uses `pvextractor`. We have also demonstrated that casa tables can be loaded into glue, enabling direct analysis of measurement sets.

4. TUTORIALS

[16] https://github.com/ALMA-IMF/reduction
[17] https://github.com/akleroy/phangs_imaging_scripts
[18] https://github.com/GBTAmmoniaSurvey/GAS
[19] https://github.com/GBTSpectroscopy/degas

We have written a series of tutorials with fully worked analyses for common operations of spectral-line data cubes. The tutorials are written as jupyter notebooks and can be accessed and ran interactively in a web browser using binder[20]. At least two tutorials of these tutorials are incorporate into the astropy learn project[21] **(two as of Feb. 2022)**. These tutorials are written at a suitable level for senior undergraduate or graduate students, or astronomers seeking to become more familiar with analyzing radio astronomy data.

The following completed tutorials include:

1. Reprojecting two spectral-cube to a common grid and resolution. astropy-learn https://github.com/astropy/astropy-tutorials/pull/504
2. Position-velocity diagram extraction and plotting. astropy-learn https://github.com/astropy/astropy-tutorials/pull/503. Figure 4 comes from this tutorial.
3. A user's guide to common operations in CASA and `spectral-cube`. `radio-astro-tools`. https://github.com/radio-astro-tools/tutorials/tree/master/casa_to_spectralcube_guide
4. Spatial and spectral fitting with `spectral-cube` and `astropy.modeling`. `radio-astro-tools`. https://github.com/radio-astro-tools/tutorials/tree/master/spectral_fitting.
5. Signal masking and moment map creation for spectral-line cubes. `radio-astro-tools`. https://github.com/radio-astro-tools/tutorials/tree/master/masking_and_moments.

Additional tutorials are in preparation, including

1. Position-velocity extraction using matplotlib interactive region drawing and manipulation
2. Reprojection to match two spectral cubes, using one to signal mask the other. `radio-astro-tools`https://github.com/radio-astro-tools/tutorials/pull/18.
3. Proof of concept for parallelizing fitting a spectral model to an entire cube using `spectral-cube`, `dask`, and `astropy.modeling`. `radio-astro-tools`. https://github.com/radio-astro-tools/tutorials/pull/12.

5. THE INTEGRATED ECOSYSTEM FOR RADIO ASTRONOMY

A key aim of the radio-astro-tools project is to integrate the radio astronomy analysis tools with the rest of the astronomical python ecosystem. This aim is achieved by supplying the tools for reading, manipulating, and writing astronomical images, cubes, spectra, and regions of interest from a wide variety of data sources.

5.1. *Reprojecting images with* `reproject`

Pixel-by-pixel comparison of data from different observations with varying pixel scales is often needed. The reprojection[22] tutorial shows how to smooth and re-grid two ALMA cubes onto a common spatial and spectral grid. All aspects of the smoothing and regridding, both spatial and spectral, can be handled within `spectral-cube` using `astropy.units` for unit handling, `astropy.convolution` for convolution, and `reproject`[23] for re-gridding onto new world coordinate systems.

5.2. *Regions-of-interest with* `regions` *and interactive* `matplotlib` *tools*

Selecting portions of the sky, or simply drawing on the sky, is an integral component of data analysis and publication figures. There are several region file formats and editors of varying popularity, particularly the SAOImage ds9 visualization tool (Smithsonian Astrophysical Observatory 2000), which defines a `.reg` format, and the CASA Viewer[24] and CARTA visualization tools (Comrie et al. 2021), which primarily use the CRTF region format[25]. The IVOA has also defined a more general region specification format, though we are not aware of any code implemented to use it.

[20] mybinder.org

[21] learn.astropy.org

[22] https://github.com/radio-astro-tools/tutorials/blob/master/SpectralCubeReprojectExample.ipynb

[23] https://reproject.readthedocs.io/

[24] https://casa.nrao.edu/casadocs/casa-5.4.1/image-cube-visualization/viewer-basics

[25] https://casadocs.readthedocs.io/en/stable/notebooks/image_analysis.html, https://casaguides.nrao.edu/index.php/CASA_Region_Format

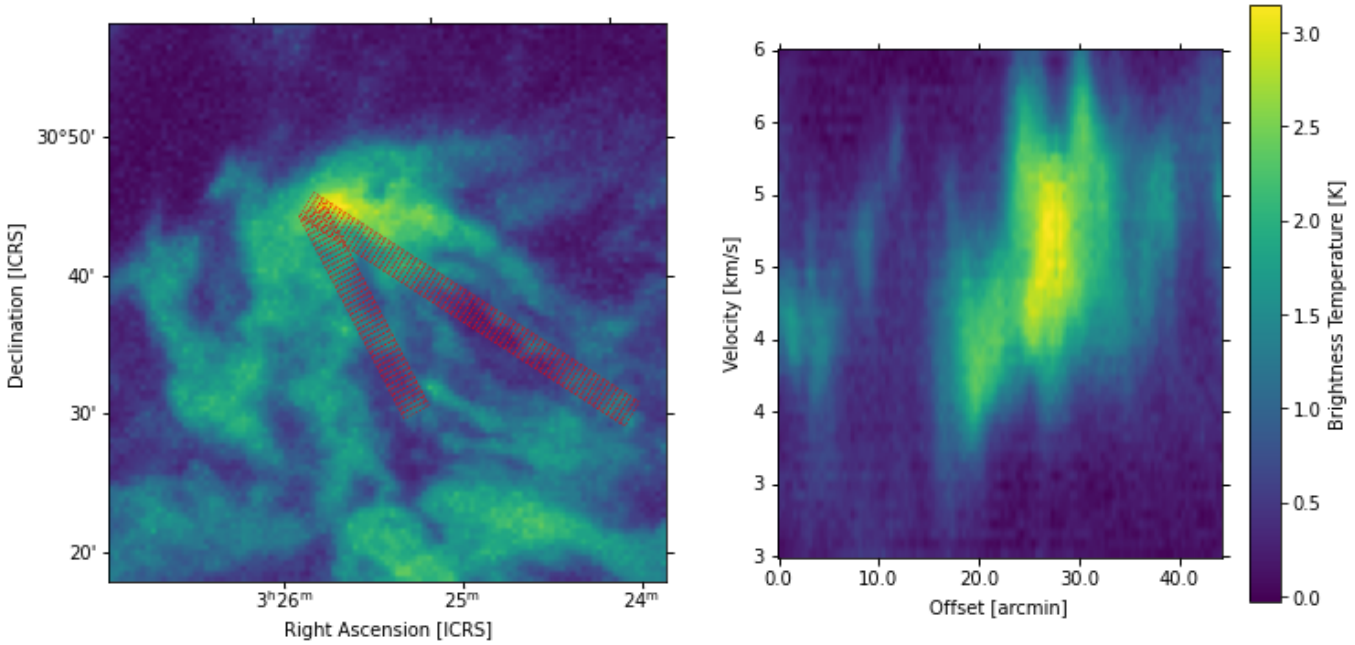

Figure 4. Example position-velocity diagram (right) and trace (left) produced in Tutorial https://github.com/astropy/astropy-tutorials/pull/504. The red rectangles shown in the left figure are extracted from the data cube and spatially averaged. The average spectrum is shown in intensity as a vertical slice in each column of the right panel.

The astropy `regions` package[26] implements a unified interface to CRTF and ds9 regions. It is able to read and write both formats.

Spectral-cube can use regions to mask out spatial subsets. The region-based masking enables, e.g., averaging spectra over a source of interest and creating position-velocity diagrams across irregular shapes. The latter is demonstrated in the disk position-velocity diagram tutorial[27].

Regions can be used in interactive cleaning to specify where model components should be added or disallowed. The CASA viewer has been used for interactive cleaning. At present, no alternatives exist, though there are some experimental implementations using matplotlib to select regions in a cube[28]. New additions to matplotlib allow creation and editing of region files interactively with the matplotlib viewer, which is agnostic of backend - i.e., it can be used on any matplotlib-compatible interactive viewer, including notebook viewers. The regions created in the interactive viewer can be saved in world coordinates when astropy's `wcscaxes` is used to display the regions.

5.3. *All CASA-produced images are spectral cubes*

CASA writes all of its outputs (its `.image`, `.residual`, etc.) files as 4-dimensional cubes. The first two dimensions represent sky coordinates, the third spectral coordinates, and the fourth the Stokes axis. For many images, i.e., continuum images or single-polarization Stokes I cubes, there are one or more 'degenerate' axes, i.e., axes with length 1. CASA explicitly acknowledges this by including the `dropdeg` keyword in the `exportfits` function, which will output a FITS file with no header information for the axes with length 1. However, for any data files not exported in this way, or any native CASA files, the underyling object is still 4-dimensional. That means that these objects can be opened directly with spectral cube, even if they are just 2-dimensional images.

6. SUMMARY

The `radio-astro-tools` project provides a broad range of tools for spectral cube analysis. It integrates with the broader astropy ecosystem, enabling analysis of multiwavelength data in a common framework.

[26] https://github.com/astropy/regions/

[27] https://github.com/radio-astro-tools/tutorials/pull/20

[28] https://github.com/urvashirau/Interactive-Imaging-with-CASA6

Development on the project is ongoing. Future development plans include additional integration into the still-under-development CASA Next Generation Infrastructure (CNGI)[29] framework. Additional tests for correctness will be added into the `uvcombine` repository. Tools for correction of the JvM effect (Jorsater & van Moorsel 1995) will be integrated into the general use tools. The `astropy.regions` interactive editing tools will be incorporated into the `glue` visualization and analysis package.

APPENDIX

A. APPROXIMATIONS IN SOLVING FOR THE COMMON BEAM IN `RADIO-BEAM`

A common operation for radio data is to convolve to a common resolution, either internal to the data (e.g., between spectral channels) or between different data sets. `radio-beam` includes two algorithms to solve for the minimum-enclosing common beam: (i) an exact solution for sets of two beams following the implementation in `casacore`, and an approximation for more than two beams based on the Khachiyan algorithm (Khachiyan & Todd 1993). Here we describe our implementation and the approximations made to enable fast computation for large (> 1000) sets of beams. Further information is given in the documentation[30].

Finding the common beam is equivalent to solving for the minimum enclosing ellipse given a set of ellipses. The formal Nd ellipsoid solution to this problem requires convex optimization as the solution lies at the edge of the valid parameter space ("minimally" enclosing Boyd et al. 2004, ; Section[31] 8.4). Our problem for the common beam is simplified in that it is only in 2D, all ellipses are centered at the origin, and we require only loose convergence to much less than pixel size since most data sample $\sim 3-7$ pixels per full-width-half-max (FWHM). We found that the Khachiyan algorithm (Khachiyan & Todd 1993; Todd & Yıldırım 2007) was well-suited given these simplifications. The Khachiyan algorithm samples points along the edge of each ellipse in the set. From these points, we compute the convex hull of the set to define the boundaries and use those boundary points to solve for the minimum enclosing ellipse.

`radio-beam` checks that all beams in the set can be deconvolved by the common beam solution. However, the algorithm can converge within the allowed tolerance to marginally smaller than the true enclosing ellipse. To avoid these cases where a beam cannot be marginally deconvolved, we increase the boundaries used by the Khachiyan algorithm by a relative fraction $1+\epsilon$ ($\epsilon = 0.001$ by default). ϵ is allowed to incrementally increase to a set maximum (default of 0.01) to ensure convergence to a common beam that can deconvolved from each beam in the set. While the addition of ϵ increases the common beam area, the default limits ensure that the increase should be far smaller than the typical pixel size and therefore negligible.

The time to compute the common beam with our implementation, including consistency checks, is far shorter than most cube operations. With a set of 1178 beams, the computation takes 0.7 s with the default parameters. We note that performance will decrease when reducing the allowed tolerance; the number of iterations linearly increases with the inverse of the threshold value.

A.1. *Beam Convolution and Deconvolution*

A poorly-resolved source will result in a Gaussian slightly larger than the beam. If observed at high signal-to-noise, the source size can still be inferred by deconvolving the beam shape from the observed source shape. This feature is implemented in `radio_beam` as `Beam.deconvolve`; for example, for an observed source with fitted size `MeasuredSourceSize`, the deconvolved source size would be `MeasuredSourceSize.deconvolve(observational_beam)`.

B. FEATHERING COMPARISON BETWEEN `UVCOMBINE` AND CASA

We demonstrate that `uvcombine` and CASA's feather task produce equivalent combined maps by generating an image with a known power law and simulating a single-dish and interferometer response. The top left panel in Figure 5 shows the original image with a power law index of -3. We produce simulate a single-dish observation by convolving the data to a beam size of $15''$ (using a Gaussian kernel). For the interferometer response, we assume an idealized case

[29] https://cngi-prototype.readthedocs.io

[30] https://radio-beam.readthedocs.io/en/latest/commonbeam.html

[31] http://web.stanford.edu/~boyd/cvxbook/

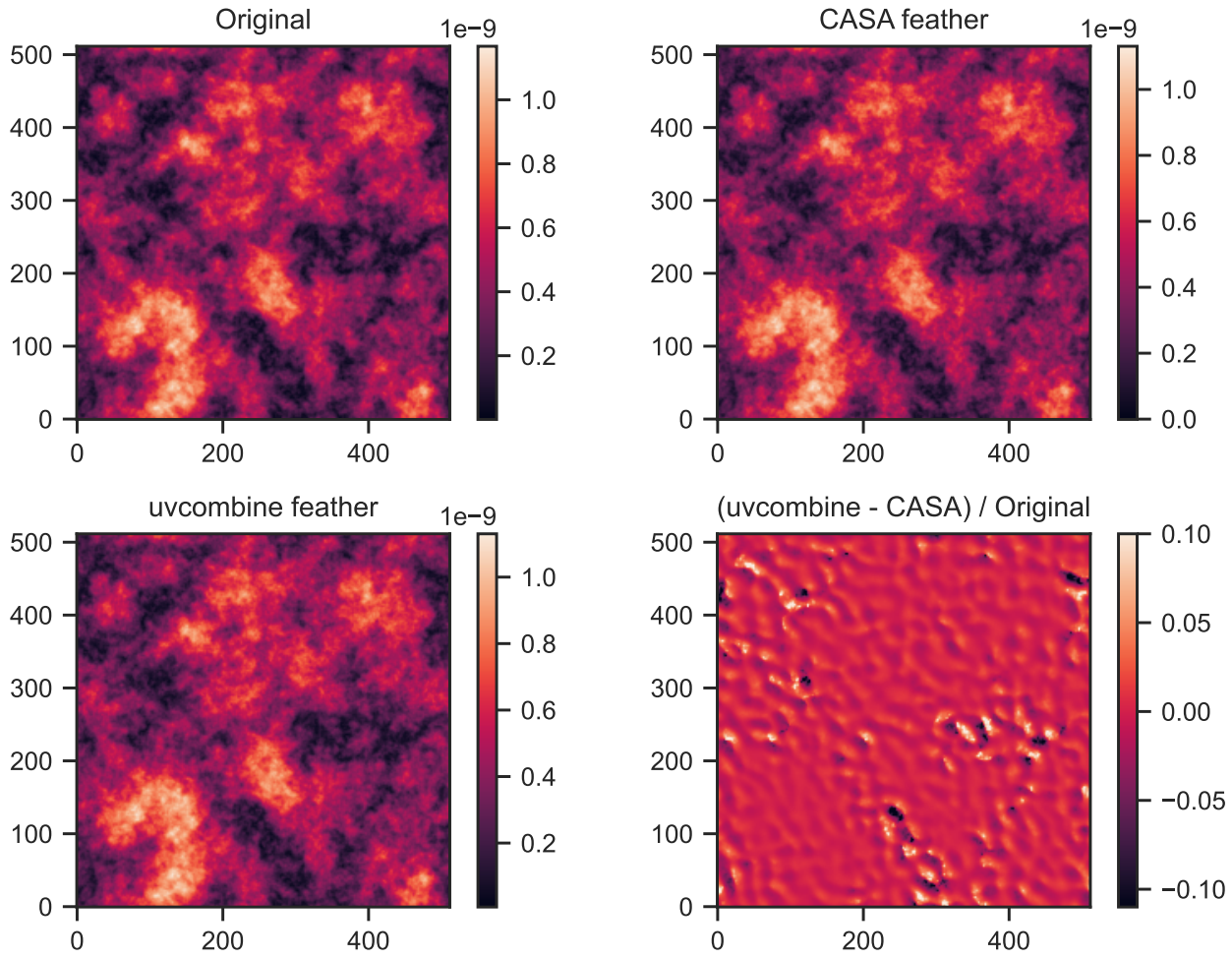

Figure 5. Demonstration of equivalent feathered maps using `uvcombine` and CASA on a generated power law data. Only small deviations remain between the feathered maps relative to the original image.

of a filled uv-plane between scales of 2–40″, or equivalently a low- and high-pass top-hat filter. We use the same inputs when feathering: a scale factor of 1.0 and no low pass filtering of the single-dish data on small scales. The feathered data, shown in the top right and bottom left in Figure 5, are visually identical. The bottom right panel shows the difference between the feathered data, normalized by the original image. The feather maps are consistent where the simulated emission is brightest, and small deviations persist only for the faintest features. We find that the power law indices of the feathered maps are consistent within $< 1\sigma$ uncertainty.

The continuous integration testing for `uvcombine` includes tests comparing to the output from CASA's feather task on simulated power law, similar to this example.

REFERENCES

Adebahr, B., Schulz, R., Dijkema, T. J., et al. 2022, Astronomy and Computing, 38, 100514, doi: 10.1016/j.ascom.2021.100514

Boyd, S., Boyd, S. P., & Vandenberghe, L. 2004, Convex optimization (Cambridge university press)

Calabretta, M. R., & Greisen, E. W. 2002, A&A, 395, 1077, doi: 10.1051/0004-6361:20021327

Casacore Team. 2019, casacore: Suite of C++ libraries for radio astronomy data processing. http://ascl.net/1912.002

Comrie, A., Wang, K.-S., Hsu, S.-C., et al. 2021, CARTA: The Cube Analysis and Rendering Tool for Astronomy, 2.0.0, Zenodo, doi: 10.5281/zenodo.3377984

Friesen, R. K., Pineda, J. E., co-PIs, et al. 2017, ApJ, 843, 63, doi: 10.3847/1538-4357/aa6d58

Ginsburg, A., Csengeri, T., Galván-Madrid, R., et al. 2021, arXiv e-prints, arXiv:2112.08183. https://arxiv.org/abs/2112.08183

Greisen, E. W., Calabretta, M. R., Valdes, F. G., & Allen, S. L. 2006, A&A, 446, 747, doi: 10.1051/0004-6361:20053818

Jorsater, S., & van Moorsel, G. A. 1995, AJ, 110, 2037, doi: 10.1086/117668

Khachiyan, L., & Todd, M. 1993, Mathematical Programming, 61, doi: 10.1007/BF01582144

Koch, E. W., Rosolowsky, E. W., Lockman, F. J., et al. 2018, MNRAS, 479, 2505, doi: 10.1093/mnras/sty1674

Leroy, A. K., Hughes, A., Liu, D., et al. 2021, ApJS, 255, 19, doi: 10.3847/1538-4365/abec80

McMullin, J. P., Waters, B., Schiebel, D., Young, W., & Golap, K. 2007, in Astronomical Society of the Pacific Conference Series, Vol. 376, Astronomical Data Analysis Software and Systems XVI, ed. R. A. Shaw, F. Hill, & D. J. Bell, 127

Pingel, N. M., Dempsey, J., McClure-Griffiths, N. M., et al. 2021, arXiv e-prints, arXiv:2111.05339. https://arxiv.org/abs/2111.05339

Robitaille, T., Beaumont, C., Qian, P., Borkin, M., & Goodman, A. 2017, glueviz v0.13.1: multidimensional data exploration, 0.13.1, Zenodo, doi: 10.5281/zenodo.1237692

Sánchez-Monge, Á., Schilke, P., Ginsburg, A., Cesaroni, R., & Schmiedeke, A. 2018, A&A, 609, A101, doi: 10.1051/0004-6361/201730425

Smithsonian Astrophysical Observatory. 2000, SAOImage DS9: A utility for displaying astronomical images in the X11 window environment. http://ascl.net/0003.002

Stanimirovic, S. 2002, in Astronomical Society of the Pacific Conference Series, Vol. 278, Single-Dish Radio Astronomy: Techniques and Applications, ed. S. Stanimirovic, D. Altschuler, P. Goldsmith, & C. Salter, 375–396. https://arxiv.org/abs/astro-ph/0205329

Todd, M. J., & Yıldırım, E. A. 2007, Discrete Applied Mathematics, 155, 1731

Turk, M. J., Smith, B. D., Oishi, J. S., et al. 2011, ApJS, 192, 9, doi: 10.1088/0067-0049/192/1/9