Macalester Journal of Physics and Astronomy

Volume 4
Issue 1 *Spring 2016*

Article 6

May 2016

Contributing to Astropy: A community Python library for astronomers

Asra Nizami
Macalester College, asra.nizami.92@gmail.com

Contributing to Astropy: A community Python library for astronomers

Abstract

This paper discusses the author's contributions to two packages affiliated with Astropy, a community Python library for astronomers. The packages the author contributed to were modeling, a sub-package within the core Astropy package, and WCSAxes, an Astropy affiliated package, outside the core package.

Keywords

Astropy python astronomy modeling wcsaxes

I. Introduction

Python has been one of the fastest growing programming languages in the astronomy community. Prior to 2011, there were a number of separate Python packages for astronomy-specific functionality, but these were mostly disjoint software development efforts with little or no coordination. This resulted in a duplication of efforts, lack of homogeneity, and no single source for most of the software tools needed by astronomers. Thus the Astropy project was started in 2011 in order to coordinate and facilitate the development of a set of common tools for astronomers and astrophysicists [1]. The Astropy project develops and maintains the core *astropy* package that contains much of the tools astronomers frequently use. In addition, the Astropy project includes work on more specialized Python packages, referred to as Astropy-affiliated packages, that are not included in the core *astropy* repository for various reasons. The *astropy* core package and Astropy-affiliated packages are open-sourced and hosted on GitHub, a web-based repository hosting service.

II . ASTROPY.MODELING

A. Background

astropy.modeling provides a framework for representing models and performing model evaluation and fitting. It currently supports different 1-D and 2-D models and allows users to fit data to these models with parameter constraints. The goal of this package is to provide users with a rich set of models and fitting algorithms such that most users will not need to define new models or fitting routines. When users try to fit their data to a certain model, they have to provide initial estimates for the parameters of the model. For instance, if a user is trying to model their dataset to a Gaussian, they have to provide the program with their estimates of the center of the peak, the depth or height and the full width at half maximum of the curve.

Data analysis tools for the upcoming James Webb Space Telescope (JWST) are to be written in Python and distributed as part of *astropy*. During a JWST analysis tools training session, scientists at the Space Telescope Science Institute were unable to fit example spectral data to certain models. The data they were using was from a sample Hubble Space Telescope (HST) spectra calibrator, in particular three absorption lines in model spectra. The models they attempted to use were the Gaussian distribution and the Lorentzian, also known as the Cauchy, distribution.

Figure 1. shows an example fit to the Lorentzian distribution. As is clearly visible in the figure, the program was unable to produce a good fit to the data. We investigated the cause of this failed fit.

Initially, we proposed several possible reasons for this failure. It could be that the data were too complicated to be modeled by this software, as the different absorption lines could be interfering with each other. It was also possible that the Gaussian and Lorentzian distributions simply were not good models to apply to these data. The last possibility was that the fitting algorithms were not behaving as we expected them to be. We decided to compare *astropy.modeling*'s results with those of another program, IRAF.

B. Testing with IRAF

IRAF, Image Reduction and Analysis Facility, is a collection of data analysis tasks for optical astronomy data. Though the first version of the software was released over 30 years ago, IRAF is still the primary optical astronomy data analysis tool used by researchers. Though it is immensely powerful, its interface is difficult to use. It is somewhat inflexible when it comes to dealing with data in new formats and cannot be integrated easily to work with other languages and programs.

An IRAF task 'splot' allows users to fit spectral data to Gaussian, Lorentzian distributions, and to Voigt profiles, where the latter is a convolution of the first two. We tested the HST spectra data against this task to eliminate the possibility that the data was too complicated. However, the data was in a form that was incompatible with IRAF. The sampling rate of the spectra varied over the range of wavelengths, but IRAF is only able to plot spectra assuming constant sampling rates. We had to interpolate the flux so that it was distributed over equally sized bins to allow splot to fit the models to it. After interpolation, we plotted the spectral data and attempted to fit the Gaussian, Lorentzian and Voigt profiles to it. Splot was successful in fitting these three distributions to the data. The best model fit was of the Voigt profile, which will be discussed later in the paper.

Successful fitting with IRAF allowed us to eliminate the possibility that the data was too complicated. We then used the parameters produced by IRAF's fitting routine and used them as inputs for *astropy.modeling*'s fitting functions. These results produced a visibly better fit compared to our initial attempt. Figure 2. shows such a fit of the Lorentzian distribution to the spectral data. Upon comparing the values produced by IRAF and the initial parameter estimates from *astropy.modeling*'s fitting, we discovered that these varied very slightly. By experimenting with parameters that varied very slightly, we discovered that *astropy.modeling*, though fully functional, is very sensitive to the initial parameter estimates. We suggest that users use *astropy.modeling* in the interactive mode of Python, iPython, such that they are able to look at their data more closely and create better estimates of the model parameters.

C. Voigt Profile

IRAF's splot command allows users to fit spectral line data to three models: the Gaussian distribution, Lorentzian distribution and the Voigt profile. The modeling sub-package in *astropy* only contains the Gaussian and Lorentzian models as compatible ones for spectral data. IRAF's splot command had produced the best fit with the Voigt profile which *astropy.modeling* did not initially have.

As part of this project, we coded a numerical approximation to the Voigt profile from [2]. A Voigt profile is a convolution of a Gaussian and a Lorentzian distribution and is computationally expensive. Since the numerical approximation fit the data nicely and worked well with the parameters produced by IRAF's splot, it appears to be well-behaved and we integrated into the main *astropy* codebase.

D. Extinction models

Extinction models are mathematical models that account for the extinction of light due to interference by interstellar and intergalactic dust. Extinction depends on the wavelength of light and several different models exist for its calculation. An Astropy-affiliated package *specutils* contains code to calculate these extinction models. However, this project is not currently well-maintained and it is the intention of the Astropy project coordinators to move these models to the *astropy.modeling* package.

We worked on porting this code over from *specutils* to *astropy.modeling*. However the original code in *specutils* uses the units package from *astropy* and needs to convert values between different units. Currently *astropy.modeling* is unable to work with models that contain units and only works for dimensionless data. There is an incomplete version of *astropy.modeling* that allows the user to create models with units but this code branch has not yet been merged with the main repository. The code we attempted to port over works well with this branch of *astropy.modeling* but it cannot be integrated and used by the community until the units branch is merged.

III. WCSAxes

A .Background

Astronomers frequently have to deal with the problem of projection in astronomical images. The image of any source on the sky is essentially the projection of something from a curved surface onto a flat two-dimensional image, which will always result in some distortions. The severity of the distortions depends on the location of the object in the sky, the size of the image and the coordinate system the image is being plotted in. Information about the default coordinate system for an astronomical image is contained in the FITS filer headers. This coordinate system is referred to as the World Coordinate System (WCS) and contains the mappings from the physical to pixel coordinate system.

The problem of creating astronomical plots with the correct WCS information on the axes isn't a new one; a lot of existing programs in different languages allow this. There are also different Python packages such as APLPY (Astronomical Plotting Library in Python), *pwcsgrid2* and *kapteyn*, that allow users to create astronomical plots. However, these programs are not very flexible and the goal was to create a common 'core' package that includes all the functionality astronomers need. WCSAxes is designed to be a flexible and extensible image plotting framework that other users and software developers can later build upon. The interface for WCSAxes was designed by the developers of APLPY and *pywcsgrid2* along with other members from the Astropy project. Currently WCSAxes is an Astropy-affiliated package and is not intended to be included in the core Astropy repository. However, it uses a lot of functionality from the *astropy* so users need to have it installed on their machines to use the package.

WCSAxes is a framework built on top of matplotlib, a popular plotting library in Python. The API is designed to be as close as possible to matplotlib so that users already familiar with matplotlib can use it easily. Plotting an image using WCSAxes to show the correct WCS information on the axes simply involves adding a few extra lines of code. Figure 3. shows an example image plotted using WCSAxes. Figure 4. demonstrates another feature of the software, that allows users to overlay gridlines on their images.

B. Image Testing Framework

Every sub-package in the core repository of Astropy as well as every Astropy-affiliated package is strongly recommended, if not required, to have a number of thorough tests to ensure that future changes in the code do not break existing functionality. Most Astropy-affiliated packages use the testing framework *pytest*, which makes the process easier. Any class or function that begins with the word 'test' are run automatically.

Since WCSAxes is an image plotting package, the tests need to be able to compare images to check for bugs. Matplotlib, the plotting package that WCSAxes is built upon, already contains a function to compare two images pixel-by-pixel. If the overall difference is greater than a set specified tolerance, the image tests fail. Figure 5. shows such an example of two images that can be compared using this function.

We wrote several tests that vary the different options available in WCSAxes. Each function tests one major or several minor parameters and creates a 'baseline image' that is stored in a directory ‘baseline_images’ in the source code. Every time the tests are run, it creates new images with the same code and compares these against the baseline images. If these images differ by more than the previously defined threshold, the image test fails and we know that something is "broken" in the source code.

Figure 6. shows a sample image in the ‘baseline_images’ directory. Since WCSAxes is only used to modify the axes, and any functions used to display the actual image data come from matplotlib, the reference images can only contain the axes. This has the added benefit of smaller overall size of the source code repository.

Astropy-affiliated packages make use of a continuous integration testing framework called Travis Continuous Integration, or Travis-CI. This means that every time someone makes changes to the source code and tries to submit them to the main repository on GitHub, Travis-CI runs all the tests online on a virtual machine. If the tests fail, these changes cannot be merged into the main repository.

However, attempts to merge the first set of image tests and baseline images failed due to failures on Travis-CI. Running these image tests locally on our own machines was easy because we were able to visually look at the baseline images and compare them to the newly generated images. Since Travis-CI runs the images on an online server, it discarded the newly generated images and we had no way to visually comparing the images created by our computer with the images created by Travis.

With a process of severe trial and error and by manipulating each of the various parameters changed in the image tests, we discovered that these image tests were failing due to differences in the font styles for the axis labels on our computers and the virtual machines used by Travis. This difference in font styles, though small, was enough to cause a failure in the tests. We worked around the problem by not changing the font styles of text at all. This isn't a severe issue since the font styles come from matplotlib and any subtle differences in font styles across different machines will most likely arise from differences in the matplotlib settings.

B.Changing axis units

Though WCSAxes was written with the primary goal of displaying angular spatial coordinates properly on the axes, astronomical data frequently contains other information such as velocity or frequency. This is particularly important for multi-dimensional data, for instance in a "data cube", where the images in physical space might be stacked in frequency or velocity space. WCSAxes allows users to slice along a multi-dimensional dataset and create a plot of this slice along any two axes. Figure 7. shows an example of such a slice.

The WCS information in images like these contain the units for the different dimensions in the dataset. For instance, in Figure 7., the default units are in m/s. However, users requested the feature to be able to change these units. Since the values for these units are larger, in thousands of meters, it would be a nice feature to allow users to change this to a unit with more manageable numbers like km/s.

WCSAxes already had the capability of handling angular coordinates and changing the units for these angular values but was unable to do so for other physical coordinates. Fortunately, *astropy*'s units module contained the exact functionality required to implement this feature. The units module is aware of almost all major physical units as well as which units can be converted to and from each other. The new feature written as part of this project extracts the units of each dimension from the WCS headers and allows the users to convert this unit to another compatible one of their choosing. Figure 8. shows such an example of an image with changed axis units.

C. Viewing world coordinates in interactive mode

Python has an interactive shell, iPython, that allows users to work more easily with Graphical User Interfaces or GUIs. WCSAxes and Matplotlib are compatible with iPython and some users that work with solar data requested a feature that would allow them to view world coordinates interactive mode.

Initially, a plot with WCSAxes in iPython could only display the pixel coordinates at the bottom of the screen. A feature was added as part of this project that would compute the pixel to world coordinate transformation of the mouse position using already built-in functions. It would then display the world coordinates instead of the pixel coordinates at the bottom of the interactive plot window. If a user overlays multiple coordinate systems onto the image, they are able to switch between the different world coordinate systems by simply pressing a key.

IV. Summary and Conclusion

With the work done as part of this project, we can conclude that the current functionality of *astropy.modeling* and WCSAxes is stable. Though there are still a lot more features that can be added to *astropy.modeling*, the current implementation works as it should. WCSAxes is mostly stable as well, though there is room of course for more features which users are free to request as they emerge.

The work on extinction models is incomplete because *astropy.modeling* still does not have support for units. The work for integrating units is currently underway by other Astropy developers, and once that is completed, extinction models will also be available in *astropy.modeling*.

I would like to thank my advisors Dr. Perry Greenfield and Dr. Thomas Robitaille for the opportunity to do this research and contribute to software that facilitates astronomy research.

References

1. The Astropy Collaboration. Astronomy & Astrophysics **558** (2013): 9

2. McLean, A. B., C. E. J. Mitchell, and D. M. Swanston. Journal of Electron Spectroscopy and Related Phenomena **69**, 2 (1994): 125–32

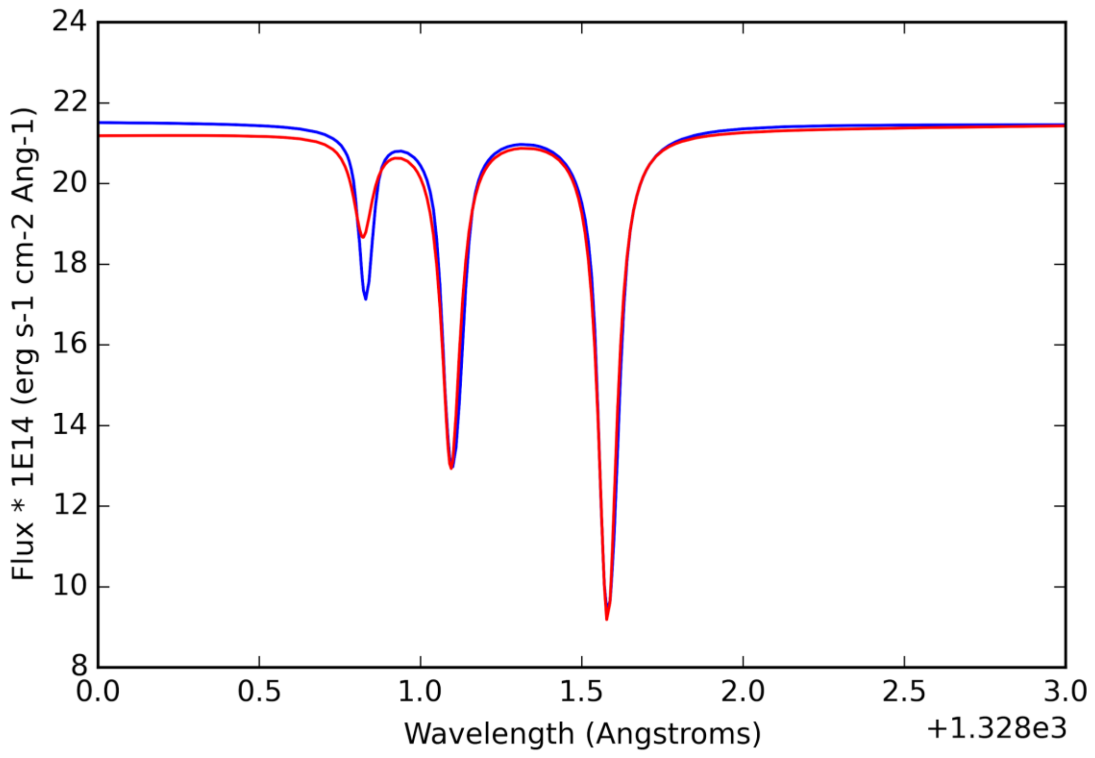

Figure 1. A fit of the Lorentzian distribution to three spectral lines from a standard HST spectra calibrator. The red curve is the model data and the blue curve is the sample data. The bad nature of the fit is clearly visible.

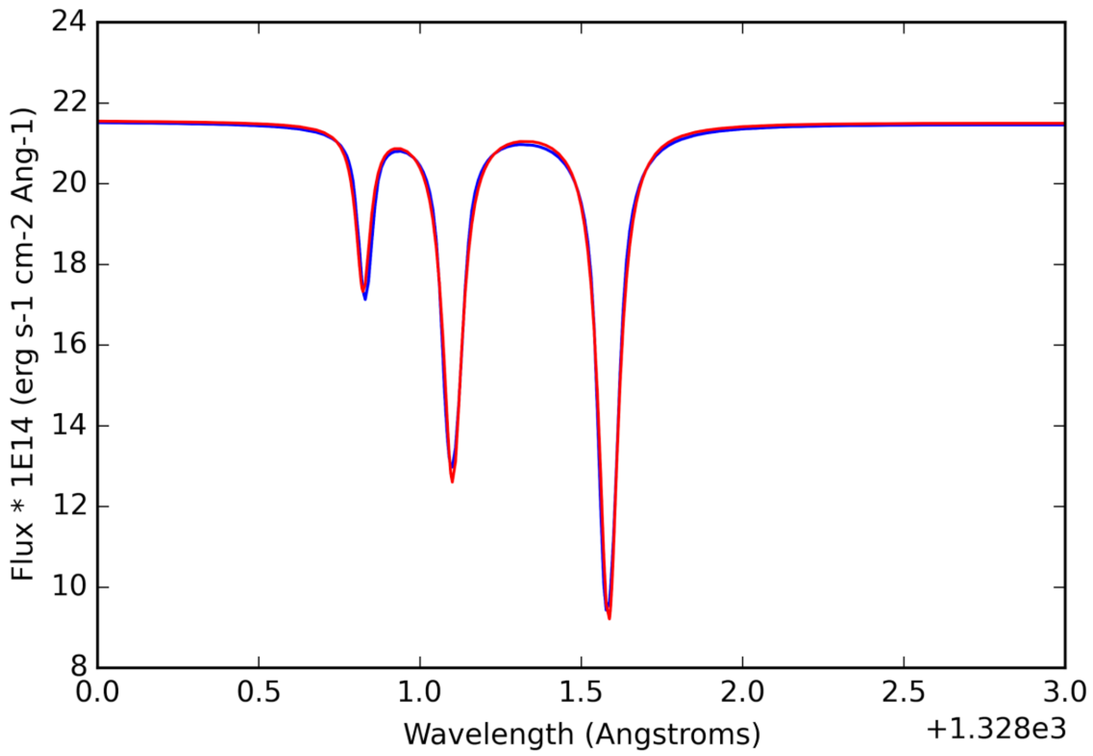

Figure 2. A fit of the Lorentzian distribution to three spectral lines from a standard HST spectra calibrator. The red curve is the model data and the blue curve is the sample data. The parameters for the model were taken from the results of IRAF's splot command. This is clearly a very good fit to the model.

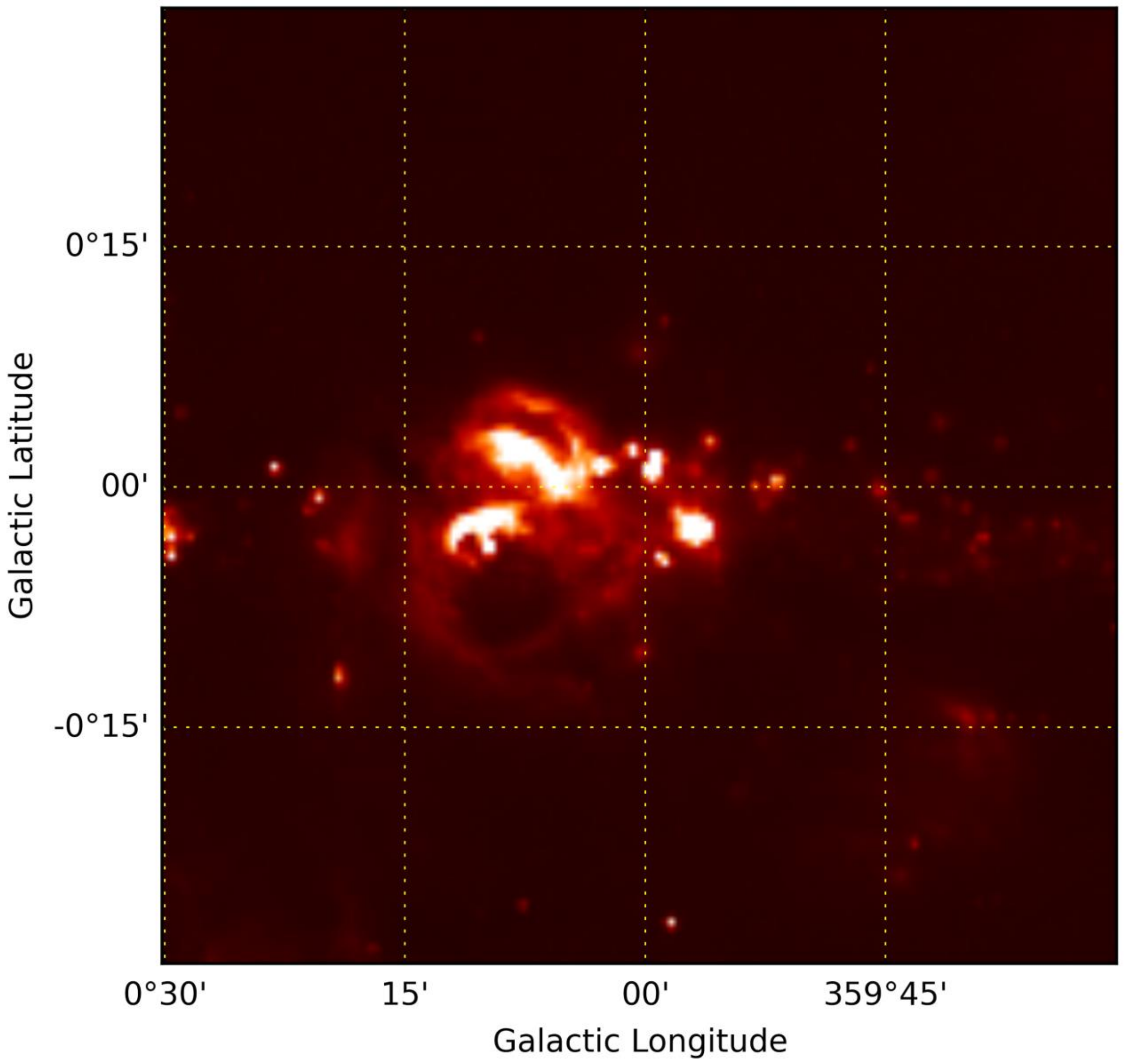

Figure 3. An image of the galactic center plotted using WCSAxes with overlaid grid contours. The default coordinate system of this image is the Galactic coordinate system.

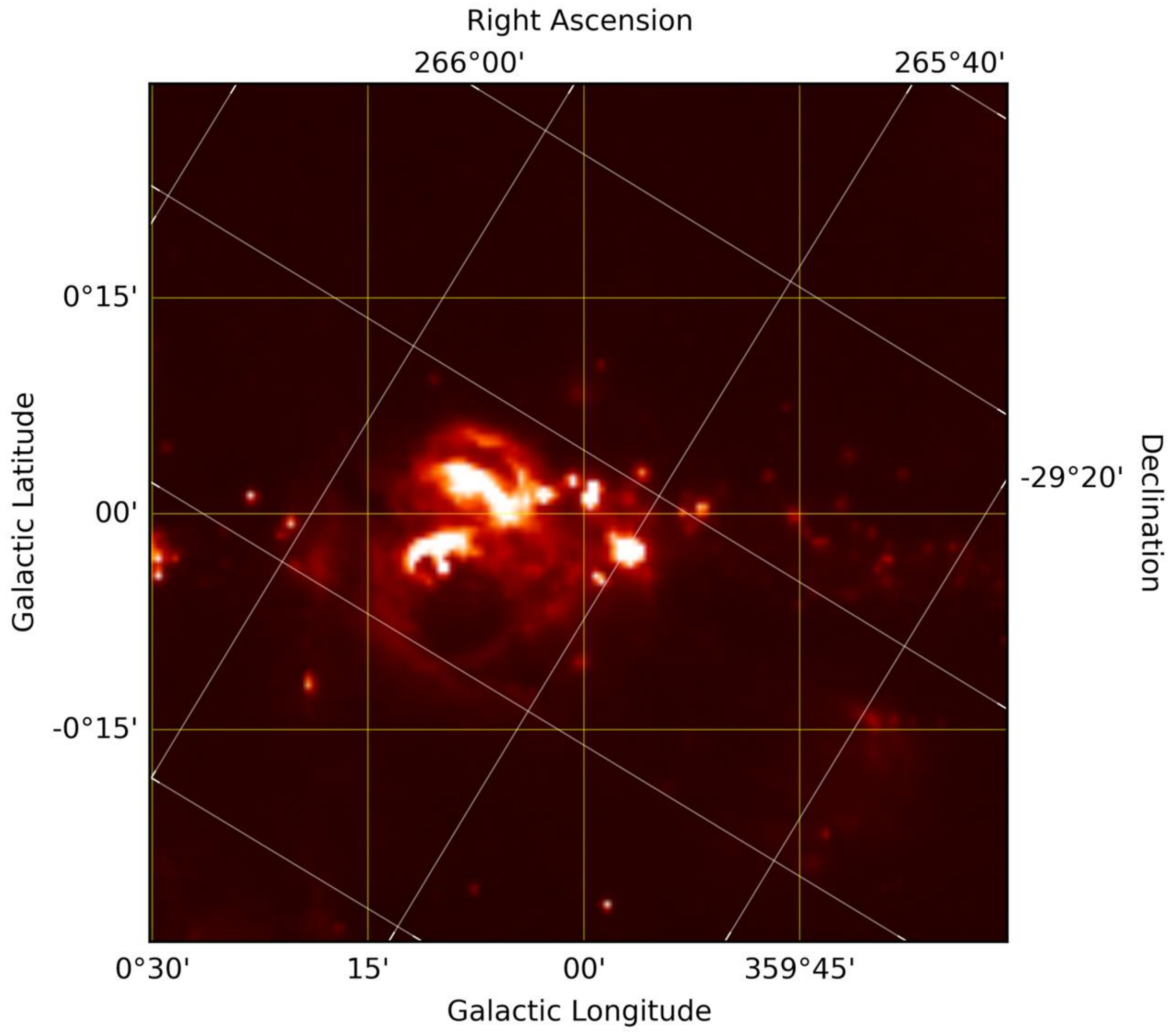

Figure 4. An image of the galactic center plotted using WCSAxes with overlaid grid contours. The default coordinate system of this image is the Galactic coordinate system with the Equatorial coordinate system overlaid. The yellow and white gridlines correspond to the Galactic and Equatorial coordinate systems respectively.

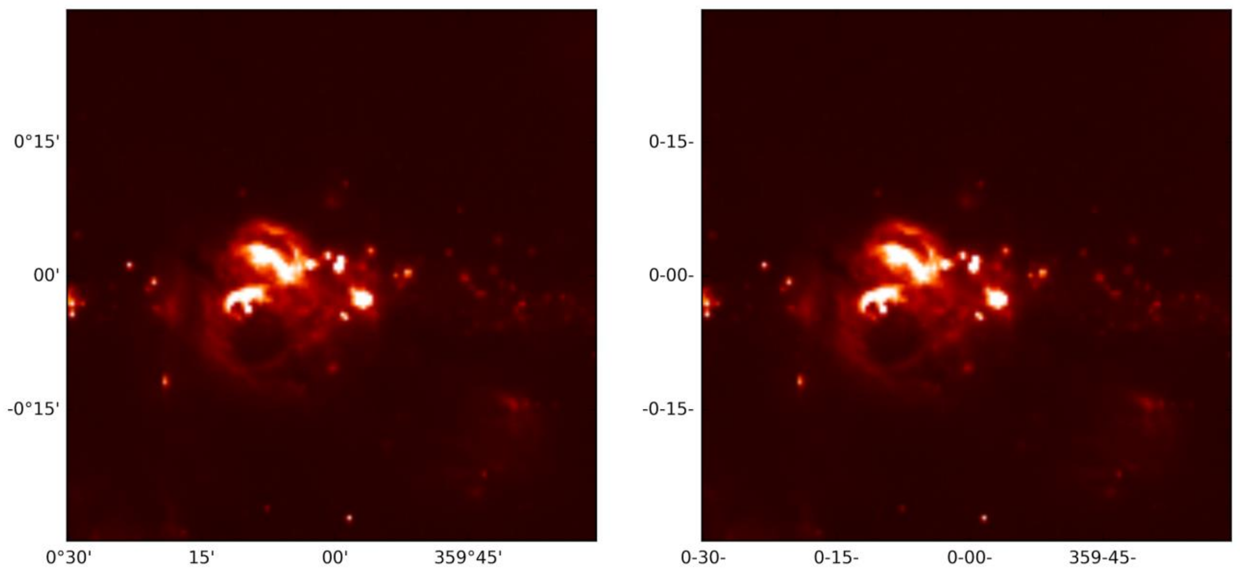

Figure 5. Two side by side plots of the same image. The characters separating the angle and arcminute values are different in both plots, and this small difference is enough to cause a failure in a pixel-by-pixel image comparison test.

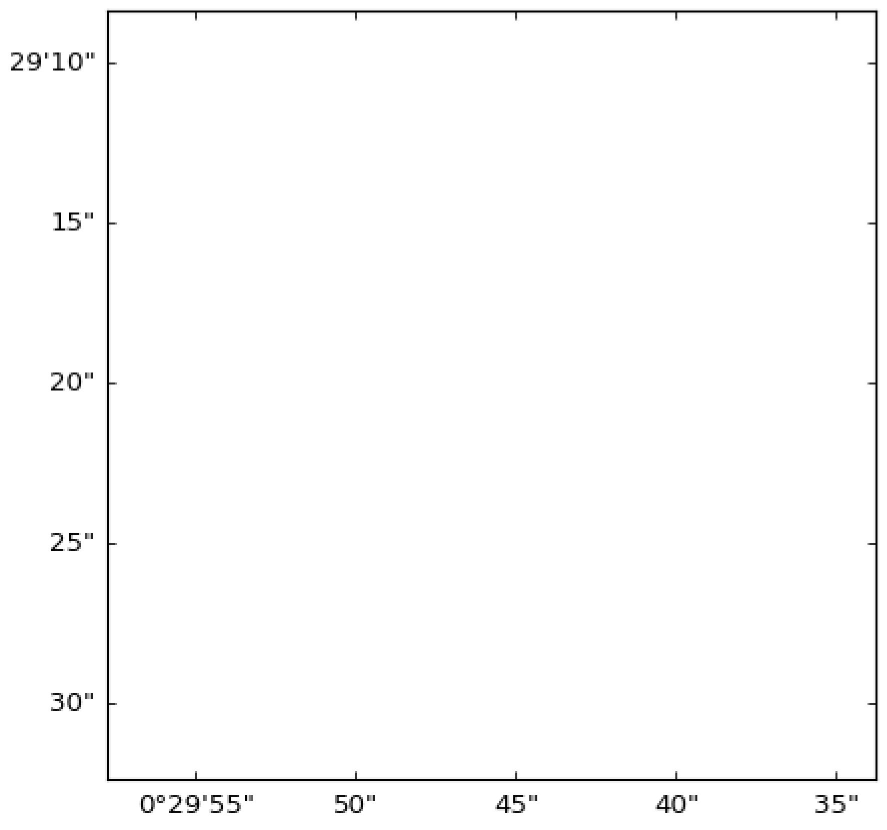

Figure 6. A sample image contained in the 'baseline_images' directory in WCSAxes. For the purposes of testing, this image contains no axis labels and does not show any image data.

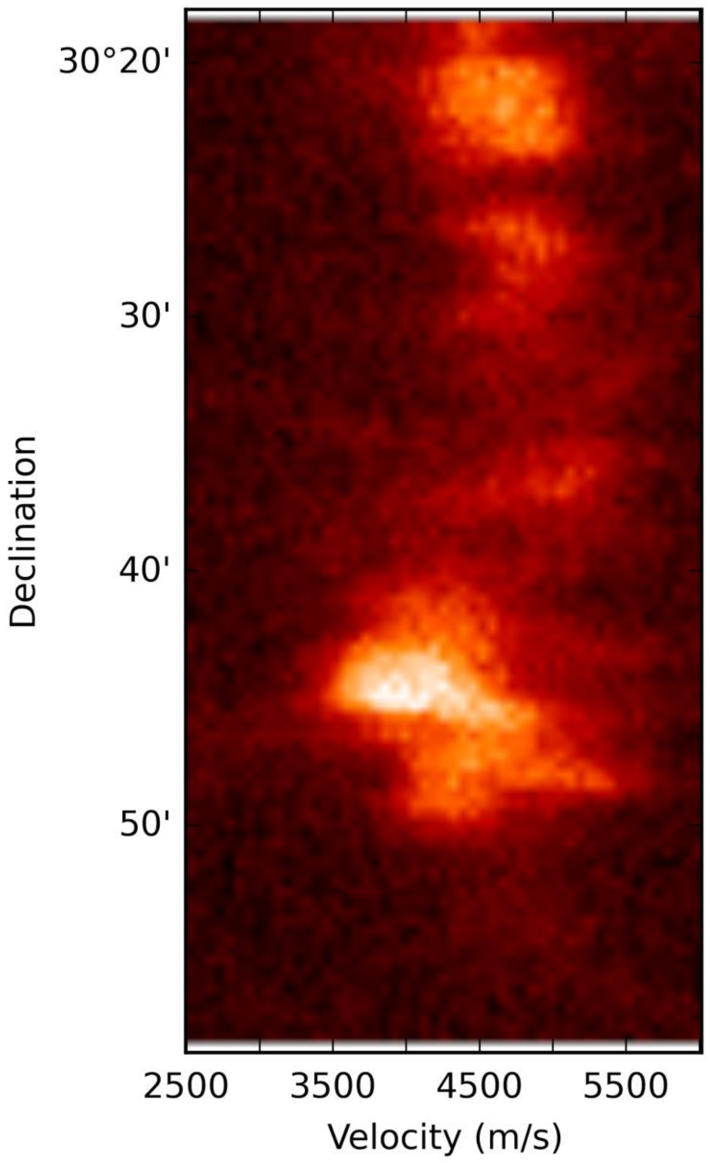

Figure 7. A plot of a slice of a three-dimensional dataset, where two are spatial dimensions corresponding to right ascension and declination and the third dimension is in velocity space. The default units of the velocity are m/s.

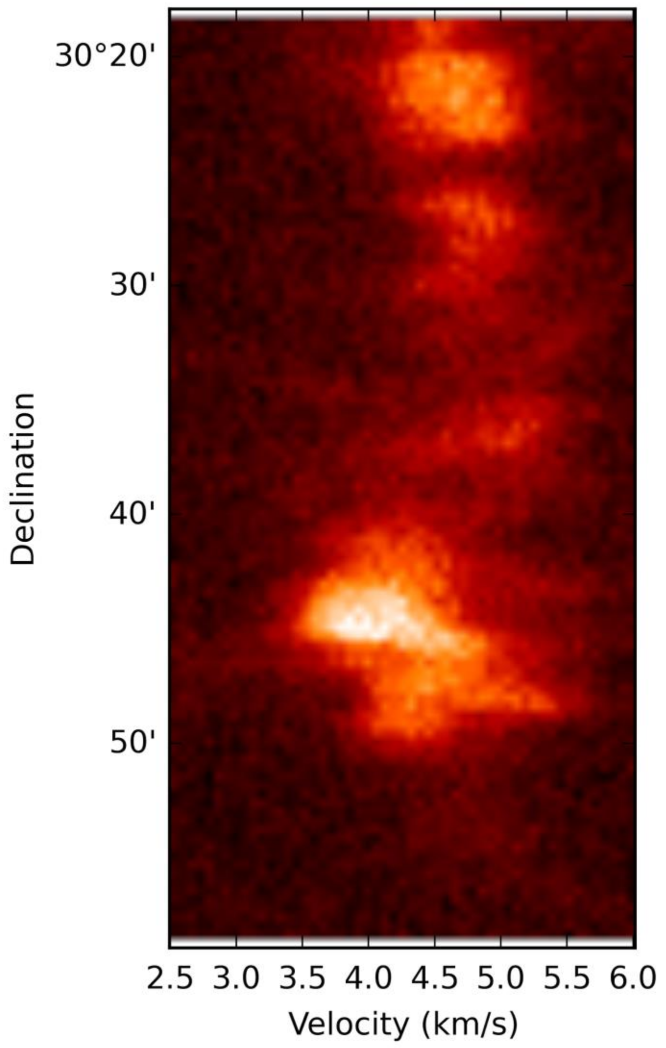

Figure 8. A plot of a slice of a three-dimensional dataset, where two are spatial dimensions corresponding to right ascension and declination and the third dimension is in velocity space. The units of the velocity axis have been changed to km/s.