Draft version November 3, 2017
Typeset using LaTeX **twocolumn** style in AASTeX61

PYSPECKIT

Adam Ginsburg

ABSTRACT

Pyspeckit is a tool and library for spectroscopic analysis.

1. INTRODUCTION

Spectroscopy is an important tool for astronomy.

2. BASIC STRUCTURE

The central object in `pyspeckit` is a `Spectrum` object, which has associated `data`, `error`, and `xarr`, the latter of which represents the spectroscopic axis. A `Spectrum` object has several attributes that are themselves sophisticated classes that can be called as functions: the `plotter`, the fitter `specfit`, and the continuum fitter `baseline`.

There are several important subclasses of `Spectrum`: `Spectra` is a collection of spectra intended to be stitched together (i.e., with non-overlapping spectral axes), `ObsBlock` is a collection of spectra with identical spectral axes intended to be operated on as a group, e.g., to take an average spectrum, and `Cube` is a 3D spatial-spatial-spectral cube.

2.1. *Plotter*

The plotter is a basic line plotter. See the GUI section (§3) for more details.

2.2. *Fitter*

The fitting tool in `pyspeckit` is the `Spectrum.specfit` object. This object is a class that is created for every `Spectrum` object. The fitter can be used with any of the models included in the model registry.

To fit a profile to a spectrum, several optimizers are available. Two implementations of the Levenberg-Marquardt optimization method are provided, `mpfit`[1] and `lmfit`[2]. Wrappers of `pymc`[3] and `emcee`[4] are also available, though these tools are better for parameter error analysis than for optimization.

The basic approach to fit a spectrum is to assign an error to each pixel in the `data` array by providing a value in the `sp.error` array. These values are assumed to be $1-\sigma$ Gaussian errors on the data.

One convenience provided by `pyspeckit` is also a potential 'gotcha': if no error is provided, the first time a spectrum is fit, the error will be automatically set by computing the standard deviation of the fit residuals. Fitting the same spectrum twice may therefore result in different parameter errors, but it should never change the fitted parameter values.

[1] Originally implemented by Craig Markwardt https://www.physics.wisc.edu/~craigm/idl/fitqa.html and ported to python by Mark Rivers and then Sergei Koposov. The version in pyspeckit has been updated somewhat from Koposov's version.

[2] https://lmfit.github.io/lmfit-py/, http://dx.doi.org/10.5281/zenodo.11813

[3] https://pymc-devs.github.io/pymc/

[4] http://dfm.io/emcee/current/, ?

3. GRAPHICAL DESIGN CHOICES

3.1. *GUI development*

Many astronomers are familiar with IRAF's `splot` tool, which is useful for fitting Gaussian profiles to spectral lines. It used keyboard interaction to specify the fitting region and guesses for fitting the line profile, but for most users, gave them access to those results *only* through the GUI.

`pyspeckit`'s fitting GUI was built to match `splot`'s functionality, but with addition means of interacting with the fitter. In `splot`, reproducing any given fit is challenging, since subtle changes in the cursor position can significantly change the fit result. In `pyspeckit`, it is possible to record the results of fits programatically and re-fit using those same results.

The GUI was built using `matplotlib`'s canvas interaction tools. These tools are limited to GUI capabilities that are compatible with all platforms (e.g., Qt, Tk, Gtk) and therefore exclude some of the more sophisticated fitting tools found in other software (e.g., `glue`).

3.2. *Plotting*

Plotting in `pyspeckit` is meant to provide the shortest path to publication-quality figures. The default plotting mode uses histogram-style line plots and labels axes with LaTeX-formatted versions of units.

When the plotter is active and a model is fit, the model parameters are displayed with LaTeXformatting. The errors on the parameters, if available, are also shown, and these errors are used to decide on how many significant figures to display.

4. `astropy` INTEGRATION

Development of `pyspeckit` began several years before `astropy` began. Several features were therefore implemented that were later replaced by similar `astropy` features, in particular the unit system. Unit conversions in `pyspeckit` are now (as of October 2015) done entirely using the `astropy` system.

5. MODELS

While many of `pyspeckit`'s internal functions are likely to replaced by `astropy` tools and affiliated packages, the models in `pyspeckit` are likely to remain useful indefinitely. The model library in `pyspeckit` includes some of the most useful general functions (e.g., Gaussian, Lorentzian, and Voigt profiles) and a wide range of specific model types.

In radio and millimeter astromomy, there are several molecular line groups that consist of several Gaussian

profiles separated by a fixed frequency offset. These hyperfine line groups are often unique probes of physical parameters because in many cases the "main" line can become optically thick, while the 'satellite' lines remain thin, meaning that the line optical depth can be measured with a single spectrum. `pyspeckit` provides the `hyperfine` model class to handle this class of molecular line transitions, and it includes several molecular species specific implementations (HCN, N_2H^+, NH_3, H_2CO).

6. CUBES

Spectral cubes are growing more important in radio astronomy since they are the natural data products produced by interferometers like ALMA and the JVLA. While many cube operations are handled well by `numpy`-based packages like `spectral-cube`[5], it is sometimes necessary or desirable to fit a profile to each spectrum in a cube. The `Cube.fiteach` method is a tool for automated line fitting that includes parallelization of the fit. This tool is not the best tested component of `pyspeckit`, but it has already seen significant use in papers and other tools (e.g., `multicube`[6]).

REFERENCES

[5] spectral-cube.readthedocs.io
[6] https://github.com/vlas-sokolov/multicube