Draft version December 29, 2017
Typeset using LaTeX **twocolumn** style in AASTeX61

PYSPECKIT

Adam Ginsburg

ABSTRACT

Pyspeckit is a tool and library for spectroscopic analysis. We describe the `pyspeckit` package, highlighting some of its unique capabilities and less unique limitations. Some of the most important assumptions regarding error estimation are described here, while details are left in the official online documentation. We discuss the motivation for the graphical design choices.

1. INTRODUCTION

Spectroscopy is an important tool for astronomy. Spectra are represented as the number of photons, or total energy in photons, arriving over a specified wavelength (or equivalently, frequency or energy) range.

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 (Levenberg 1944; Marquardt 1963) 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.

Once a fit is performed, the results of the fit are accessible through the `parinfo` object, which is a dictionary-like structure containing the parameter values, errors, and other metadata (e.g., information about whether the parameter is fixed, tied to another parameter, or limited). Other information about the fit, such as the χ^2 value, are available as attributes of the `specfit` object.

Optimal χ^2 —There is a tool to compute the 'optimal' χ^2, which is the χ^2 value computed only over the range where the model contains statistically significant signal. By default, the function selects all pixels where the model value is greater (in absolute value) than the corresponding error. In principle, this optimal χ^2 may be helpful for obtaining correctly scaled errors (see Section 2.3.2), though this claim has never been rigorously tested.

2.3. *Error Treatment*

The `Spectrum` objects used by pyspeckit have an attached `error` array, which is meant to hold the $1-\sigma$ independent Gaussian errors on each pixel. While this error representation may be a dramatic oversimplification of the true errors for almost all instruments (since it completely ignores correlation between adjacent pixels), it is also the most commonly used assumptions in most astronomical applications.

The errors are used to determine the best-fit parameters and their errors (see §2.2). They can be displayed as error bars on individual pixels or as shaded regions around those pixels using different display modes [FIGURE?].

2.3.1. *Automatic Error Estimation*

If a spectrum is created with no associated errors, `pyspeckit` can automatically estimate the errors from a fit residual. If no errors are specified and a fit is initiated, the errors will default to being uniformly 1.0, which is rarely ever correct, but will result in the appropriate weights and therefore the appropriate best-fit parameters with *incorrect* errors.

One common approach in spectroscopy in the limit that there are few pixels with signal and many pixels representing only noise is to estimate the errors by the standard deviation of the signal-free pixels. This approach generally assumes the noise is constant across the spectrum. In the case that a single signal feature is present in the spectrum, and it can be accurately modeled, the standard deviation of the residual spectrum from the model fit will accurately represent the uniform errors. `pyspeckit` will automatically replace the errors with the standard deviation of the residuals if a fit is performed with uninitialized errors; this means that performing a fit on the same data (without associated errors) twice will result in the same parameter values both times, but different errors the second time.

[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/`, Foreman-Mackey et al. (2013)

2.3.2. *Parameter error estimation*

Parameter errors are adopted from the `mpfit` or `lmfit` fit results. The Levenberg-Marquardt algorithm finds a local minimum in parameter space, and one of its returns is the parameter covariance matrix. This covariance matrix is not directly the covariance of the parameters, and must be rescaled to deliver an approximate error.

The standard rescaling is to multiply the covariance by the sum of the squared errors divided by the degree of freedom of the fit, usually referred to as χ^2/N. The degree of freedom is assumed to be equal to the number of free parameters, e.g., for a 1-dimensional Gaussian, there would be three: the amplitude, width, and center. This approach implicitly assumes that the model describes the data well and is an optimal fit. It also assumes that the model is linear with all of the parameters, at least in the region immediately surrounding the optimal fit. These parameters are frequently not satisfied; see Andrae et al. (2010) and Andrae (2010) for details.

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 be 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 spectral model 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]).

7. SUMMARY

`pyspeckit` is a versatile tool for spectroscopic analysis.

REFERENCES

Andrae, R. 2010, ArXiv e-prints

Andrae, R., Schulze-Hartung, T., & Melchior, P. 2010, ArXiv e-prints

Foreman-Mackey, D., Hogg, D. W., Lang, D., & Goodman, J. 2013, PASP, 125, 306

Levenberg, K. 1944, Quarterly of Applied Mathematics, 2, 164

Marquardt, D. W. 1963, Journal of the Society for Industrial and Applied Mathematics, 11, 431

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