Draft version June 21, 2018
Typeset using LaTeX **twocolumn** style in AASTeX62

pyspeckit: A spectroscopic analysis and plotting package

Adam Ginsburg, Vlas Sokolov, Miguel de Val-Borro, Allison Youngblood, Erik Rosolowsky, and Jaime E. Pineda

ABSTRACT

`pyspeckit` is a tool and library for spectroscopic analysis in Python. We describe the `pyspeckit` package and highlight some of its unique capabilities such as interactively fitting a model to data, akin to the widely-used `splot` function of `IRAF`. `pyspeckit` employs the Levenberg-Marquardt optimization method via `mpfit` and `lmfit`, and important assumptions regarding error estimation are described here. Wrappers for `pymc` and `emcee` are available, as well as a method to fit lines in spectral cubes. As part of the `astropy` affiliated packages, `pyspeckit` is open source and welcomes input and collaboration from the community.

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. Emission and absorption lines due to ions, atoms, and molecules bear important information via their intensity, width, and velocity centroid. These parameters are typically measured from model fits to the data, such as Gaussians, Lorentzians, and Voigt profiles. Historically, `IRAF` has provided the astronomy community with easy-to-use tools for line fitting, but `IRAF` development has mostly ceased in the last several years and is currently only supported in Python 2.7 by AstroConda[1], though the PyRAF command language (https://github.com/spacetelescope/pyraf) supports both Python 2 and Python 3 and is still maintained.

`pyspeckit` development began in 2009 with a script called 'showspec' in the `agpy` package hosted on Google Code. It was created and used by a graduate student to plot and sometimes fit profiles to spectra in python. At the time, IDL was still more popular than python at most institutes (Momcheva & Tollerud 2015), and there were no publicly available and advertised tools for spectral plotting, fitting, and general manipulation (`astropysics` (Tollerud 2012) was developed contemporaneously and solved many of the same problems as `pyspeckit`). The `astropy` project had its first commit in 2011, so even the basic infrastructure for such analysis was not yet established.

`pyspeckit`'s graphical user interface (GUI) features were inspired by IRAF's `splot` tool, while the fitting features were inspired in part by `xspec` (https://heasarc.gsfc.nasa.gov/xanadu/xspec/). Over subsequent years, `pyspeckit` has grown by incorporating more sophisticated models and improving its internal structure. The package was moved out of `agpy` and into its own repository in 2011, first spending a few years on Bitbucket in a mercurial repository, then it finally moved to GitHub, where it currenly resides, in 2012.

Because `pyspeckit`'s initial development preceded `astropy`, some features were included that later became redundant with `astropy`. Most notably, `pyspeckit` included a limited system for spectroscopic unit conversion. In 2015, this system was completely replaced with `astropy`'s unit system. Around the same time, the Doppler conversion tools (converting from frequency or wavelength to velocity) that existed in `pyspeckit` were pushed upstream into `astropy`, highlighting the mutually beneficial role of `astropy`'s affiliated packaged system (Price-Whelan et al. 2018). `pyspeckit` was finally accepted as an `astropy` affiliated package in 2017.

In this paper we briefly outline `pyspeckit` architecture and highlight its key capabilities. In Section 2, we outline the basic structure of the package. In Section 3, we describe the GUI system. In Sections 5 and 4, we outline `pyspeckit`'s cube handling capabilities and model library.

2. BASIC STRUCTURE

The central object in `pyspeckit` is a `Spectrum` object, which has associated `data` (e.g., flux), `error`, and `xarr` (e.g., wavelength, frequency, energy), the latter of which represents the spectroscopic axis. A `Spectrum` object has several attributes that are themselves sophisticated

[1] http://astroconda.readthedocs.io/en/latest/index.html

classes that can be called as functions: the `plotter`, the fitter `specfit`, and the continuum fitter `baseline`[2].

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. *Supported data formats*

`pyspeckit` supports a variety of open and proprietary data formats that have been traditionally used to store spectral data products in astronomy. We list the currently supported formats here:

- ASCII: Reading a one-dimensional spectrum from a text file with an optional error column can be done using the `astropy.io.ascii` module in any of its supported formats.

- FITS: The Flexible Image Transport System (FITS; Wells et al. 1981; Greisen et al. 2006; Pence et al. 2010) format is supported in `pyspeckit` with `astropy.io.fits`. FITS spectra are expected to have their spectral axis defined using the WCS keywords in the FITS header. FITS binary tables can also be used.

- SDFITS: Data files following the Single Dish FITS (SDFITS; Garwood 2000) convention for radio astronomy data as produced by the Green Bank Telescope are partially supported in `pyspeckit`.

- HDF5: The Hierarchical Data Format (HDF5) file format has been designed to store and organize large amounts of data and offers significant advantages over FITS. Although not widely used in observational astronomy, the pipeline of the Low-Frequency Array (LOFAR) radio telescope uses the HDF5 data format to efficiently store large data volumes (Alexov et al. 2012). If the `h5py` package is installed, `pyspeckit` will support read access to files containing spectra in the HDF5 format (although there is no specified standard for spectra in HDF5, so additional user effort may be required to create `pyspeckit` Spectrum objects from the extracted HDF5 data).

- Finally, `pyspeckit` is capable of reading files from some versions of the GILDAS Continuum and Line Analysis Single-dish Software format (CLASS; Gildas Team 2013). The CLASS reader is known to be compatible with data files from the Arizona Radio Observatory telescopes (12-m and 10-m Submillimeter Telescope) and the Atacama Pathfinder Experiment (APEX) radio telescope.

2.2. *Plotter*

The `plotter` is a basic plot tool that comprises `pyspeckit`'s main graphical user interface. It is described in more detail in the GUI section (§3).

2.3. *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, or a custom model can be created and registered.

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`[3] and `lmfit`[4]. Wrappers of `pymc`[5] and `emcee`[6] 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 —Specfit computes 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

[2] It is common in radio astronomy to have wide instrumental residual features in the data that need to be fitted and removed; this process is called 'baseline subtraction'. In other wavelength regimes, this would typically be referred to as continuum fitting or continuum subtraction. In practical algorithmic terms, fitting a true astrophysical continuum and a residual instrumental baseline are indistinguishable.

[3] 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.

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

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

[6] http://dfm.io/emcee/current/, Foreman-Mackey et al. (2013)

for obtaining correctly scaled errors (see Section 2.6.2), though this claim has never been rigorously tested.

2.4. *Data Selection*

An important feature of the spectral fitter is the ability to select the region of the spectrum to be fit. This selection process can either be done manually, using the `selectregion` method to set one or more ranges of data to include in the fit, or interactively using the GUI. By default, the selected regions are then highlighted.

2.5. *Continuum Fitting*

The fitting process in `pyspeckit` is capable of treating line and continuum independently or jointly. If a model includes continuum, e.g., for the case of a four-parameter Gaussian profile that includes an additive constant, it can be fitted through the standard `specfit` fitter.

However, it is common practice to fit the continuum independently prior to fitting lines. Such practice is necessary when fitting absorption lines and practically necessary for heterodyne radio observations where the continuum is usually poorly measured and corrupted by instrumental effects. Following radio convention, the `pyspeckit` continuum fitting tool is called `baseline`. This module supports polynomial, spline, and power-law fitting.

2.6. *Error Treatment*

The `Spectrum` objects used by `pyspeckit` have an attached `error` array, which is meant to hold the 1σ 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 ignores correlations between adjacent data), it is also the most commonly used assumption in astronomical applications.

The `error` array is used to determine the best-fit parameters and their uncertainties (see §2.3). They can be displayed as error bars on individual pixels or as shaded regions around those pixels using different display modes.

A typical example is given below, where we generate a spectrum and error array using `numpy` and `astropy` tools.

```
from astropy import units as u
import numpy as np
import pyspeckit

xaxis = np.linspace(-25, 25)*u.km/u.s
sigma = 3.0*u.km/u.s
data = 5*np.exp(-xaxis**2 /
                (2*sigma**2))*u.Jy
error = np.ones_like(data) * 0.2
sp = pyspeckit.Spectrum(xarr=xaxis,
                        data=data,
                        error=error)

sp.plotter(errstyle='fill')

sp.plotter.savefig("example_fig_1.pdf")
```

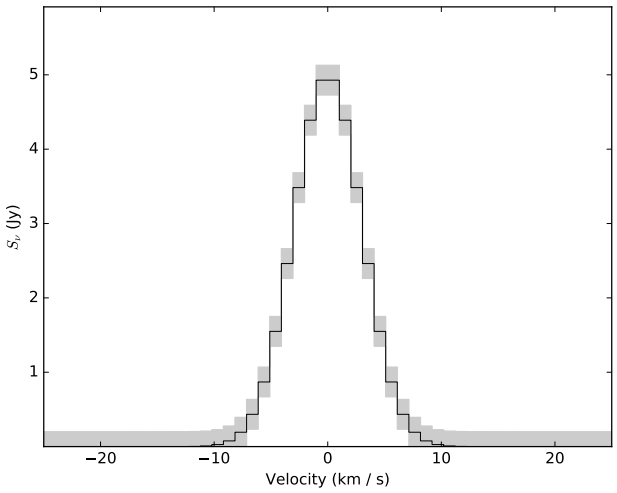

Figure 1. An example plotted spectrum showing the automated unit labeling and errors. The errors are shown with the `'fill'` style and represent symmetric $1-\sigma$ Gaussian errors.

2.6.1. *Automatic Error Estimation*

In the case where there are clear portions of the spectrum that have no significant emission, a common approach in spectroscopy is to estimate the errors from the standard deviation of those signal-free pixels. This approach 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. After a fit is performed with uninitialized errors, `pyspeckit` will automatically replace the errors with the standard deviation of the residuals; this means that performing a fit on the same data (without associated er-

rors) twice will result in the same parameter values both times, but different errors the second time.

2.6.2. *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 number of degrees of freedom is assumed to be equal to the number of free parameters, e.g., for a one-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 requirements are frequently not satisfied; see Andrae et al. (2010) and Andrae (2010) for details. We show a demonstration of this approximation process in Appendix A for the case of a simple Gaussian line profile.

3. GRAPHICAL DESIGN

3.1. *GUI development*

Many astronomers are familiar with IRAF's `splot` tool, which is useful for fitting Gaussian profiles to spectral lines. It uses keyboard interaction to specify the fitting region and guesses for fitting the line profile, but for most use-cases, these parameters could *only* be accessed through the GUI.

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

The GUI was built using `matplotlib`'s canvas interaction tools. These tools are limited to the 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` Beaumont et al. 2014).

3.2. *Plotting*

Plotting in `pyspeckit` is designed to provide a short path to publication-quality figures. The default plotting mode uses histogram-style line plots, which follows the radio and inteferometric standard, 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 LaTeX formatting. The errors on the parameters, if available, are also shown, and these uncertainties are used to decide on the number of significant figures to display.

4. MODELS

Many of `pyspeckit`'s internal functions are likely to replaced by the `astropy specutils` package in the future. However, the rich suite modeling in `pyspeckit` is likely to remain useful indefinitely. This model library 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 (e.g., ammonia and formaldehyde hyperfine models, the H_2 rotational ladder, and recombination line models).

In radio and millimeter astronomy, 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 these different features have different, known relative optical depths. In this case, the measured relative amplitudes of these different features allow the optical depth to be measured from a single spectrum. `pyspeckit` provides the `hyperfine` model class to handle this class of molecular line transitions, and it includes several molecular species implementations (HCN, N_2H^+, NH_3, H_2CO). Models can customizable and examples of registering a new or modified model in `pyspeckit` are included in the online documentation. TODO: ADD MODEL TABLE, maybe add demo of custom model? That might be in the docs already... If this is the beefiest part of pyspeckit, we should include a table listing the models and potentially an example for how to customize a model? **(Author note: I (JEP) agree, it would be really useful to include the list of models already included, and which ones could be used as templates for different types.)**

5. 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. Optical and infrared data cubes are also becoming more common from integral field units (IFUs) like MUSE on the VLT, OSIRIS on Keck, NIFS on Gemini, and NIRSpec and MIRI on JWST.

While many cube operations are handled well by `numpy`-based packages like `spectral-cube`[7], it is sometimes 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. Implementation examples can be found in the online documentation. This tool has seen significant use in custom made survey pipelines, (e.g. `?`, https://github.com/GBTAmmoniaSurvey/GAS), papers and it has been incorporated into other tools (e.g., `multicube`[8]).

6. SUMMARY

`pyspeckit` is a versatile tool for spectroscopic analysis in python and is one of the `astropy` affiliated packages. `pyspeckit` can interactively fit a model to a spectrum using the Levenberg-Marquardt optimization method via `mpfit` and `lmfit`, and wrappers for `pymc` and `emcee` are also available. There is also the option to fit a model to the many spectra in a spectral cube. We have described `pyspeckit`'s methods of error estimation for spectra with and without user-provided errors. `pyspeckit` has a library of models including Gaussian, Lorentzian, Voigt, and others for specific molecular species; user-created models can also be used with `pyspeckit`.

7. FINAL NOTE

This paper was collaboratively written using GitHub as a platform for discussion. Its version history and records of some discussions about its content can be found at https://github.com/pyspeckit/paper1.

REFERENCES

Alexov, A., Schellart, P., ter Veen, S., et al. 2012, in Astronomical Society of the Pacific Conference Series, Vol. 461, Astronomical Data Analysis Software and Systems XXI, ed. P. Ballester, D. Egret, & N. P. F. Lorente, 283

Andrae, R. 2010, ArXiv e-prints

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

Beaumont, C., Robitaille, T., & Borkin, M. 2014, Glue: Linked data visualizations across multiple files, Astrophysics Source Code Library

Foreman-Mackey, D., Hogg, D. W., Lang, D., & Goodman, J. 2013, PASP, 125, 306, doi: 10.1086/670067

Garwood, R. W. 2000, in Astronomical Society of the Pacific Conference Series, Vol. 216, Astronomical Data Analysis Software and Systems IX, ed. N. Manset, C. Veillet, & D. Crabtree, 243

Gildas Team. 2013, GILDAS: Grenoble Image and Line Data Analysis Software, Astrophysics Source Code Library

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

Levenberg, K. 1944, Quarterly of Applied Mathematics, 2, 164, doi: 10.1090/qam/10666

Marquardt, D. W. 1963, Journal of the Society for Industrial and Applied Mathematics, 11, 431, doi: 10.1137/0111030

Momcheva, I., & Tollerud, E. 2015, ArXiv e-prints

Pence, W. D., Chiappetti, L., Page, C. G., Shaw, R. A., & Stobie, E. 2010, A&A, 524, A42, doi: 10.1051/0004-6361/201015362

Price-Whelan, A. M., Sipőcz, B. M., Günther, H. M., et al. 2018, ArXiv e-prints

Tollerud, E. 2012, Astropysics: Astrophysics utilities for python, Astrophysics Source Code Library

Wells, D. C., Greisen, E. W., & Harten, R. H. 1981, A&AS, 44, 363

[7] spectral-cube.readthedocs.io

[8] https://github.com/vlas-sokolov/multicube

APPENDIX

A. PARAMETER ERROR ESTIMATION FOR A SIMPLE 1D GAUSSIAN PROFILE

As discussed in Section 2.6.2, parameter errors are estimated in `pyspeckit` by the underlying `lmfit` or `mpfit` tools using the approximation that the reduced chi-squared is unity, $\chi^2/n = 1$. We demonstrate here that, for a simple one-dimensional Gaussian profile, this approximation results in a reasonable, but not perfect, recovery of the underlying parameter errors.

In Figure 2, we show a synthetic spectrum with uniform Gaussian random noise and perfectly-measured uncorrelated data errors. The fitted model is a one-dimensional Gaussian profile with free parameters amplitude, center, and width. The fit results are given in the figure.

To produce a good error estimate under the $\chi^2/n = 1$ approximation, the error distribution must be Gaussian, the model must be linear in all parameters, and the model must be the correct underlying model (Andrae 2010).

Figure 3 shows the χ^2 values in parameter space surrounding the best-fit value. Along the diagonal, we show the χ^2 values for the individual parameters with all others marginalized over by taking the minimum χ^2 value over the explored parameter space. The vertical dashed lines show the estimated 1σ errors reported by the `mpfit` optimizer, while the horizontal dashed lines show the value $\Delta\chi^2 = 1$, which corresponds to the 68% confidence interval for that parameter. If the $\chi^2/n = 1$ approximation were perfect, the dashed lines would intersect with the solid lines.

Off of the diagonal of Figure 3, we show the two-dimensional marginal distributions. Contours are shown at $\Delta\chi^2 = 2.3, 6.2, 11.8$, corresponding to 68%, 95%, and 99.5% (1σ, 2σ, and 3σ for a normal distribution) confidence regions. The vertical and horizontal dashed lines show the estimates from the $\chi^2/n = 1$ approximation. The shift vs amplitude and shift vs width diagrams both show very good matches. However, the width vs amplitude plot indicates that the single-parameter errorbars underestimate the true errors because these parameters are significantly correlated. This information is captured in the covariance matrix that is used to compute the single-parameter errors, as it has significant values in the off-diagonal parts of the matrix. More broadly, this approach for estimating parameter uncertainties also relies on the analysis problem fulfilling the conditions for least-squares fitting, namely that the dependent variables are perfectly known and that the model would be the correct representation of perfectly known data.

The source code for this example can be found in the `pyspeckit` github repository in `examples/synthetic_spectrum_example_wi`

B. PARAMETER ERROR ESTIMATION FOR AMMONIA

In Section A, we showed the parameter estimation results in the case of a modeled 1-dimensional Gaussian. One of the most commonly used models in `pyspeckit` is the ammonia (NH_3) hyperfine model, which has several additional emission lines and several parameters governing those lines.

The ammonia inversion transitions are notable for having spectrally resolved hyperfine components, the relative weights of which are governed by quantum mechanics (**?**). The existence of these additional components often allows for direct estimates of the optical depth of the central line, which is optically thicker than the other components, thereby making column density estimates straightforward compared to other molecular species.

The model for these lines is more complicated than that for a single Gaussian. The model must include a simplified version of the radiative transfer equation and must simultaneously produce the predicted emission of several lines. Additionally, there are several approximations that are convenient to use in different circumstances, so `pyspeckit` implements several different variants of the NH_3 model.

In this section, we show parameter estimates analogous to those in Section A. We examine a case where the fitted lines are in local thermodynamic equilibrium (LTE), such that the ratios of the $(J, K) = (1, 1)$ to $(2, 2)$ line is governed by the rotational temperature $T_{\rm R}$ but the individual lines both have $T_{\rm ex} = T_{\rm R}$.

The free parameters in the ammonia model are the rotational temperature, $T_{\rm R}$, which governs the relative populations of the rotational states, the excitation temperature $T_{\rm ex}$, which governs the relative populations of the two levels within a single inversion transition, the column density, $N(NH_3)$, which specifies the total column density of NH_3 integrated over all states (note that this parameter enters the model as 10^N, i.e., we optimize the log of the column density), the line-of-sight velocity $v_{\rm LoS}$, the line width σ_v, and the ortho-to-para ratio parameterized as the fraction of ortho-NH_3 F_{ortho}. In the examples below, we fix $F_{ortho} = 0$ and treat only para-NH_3 lines.

The fit results from the first case are shown in Figures 4 and 5. The fit recovers the input parameters, but reveals one of the important caveats when using any optimization algorithm: in some models, parameters are degenerate, and therefore using the diagonal of the covariance matrix to estimate the variance can result in incorrect error estimates.

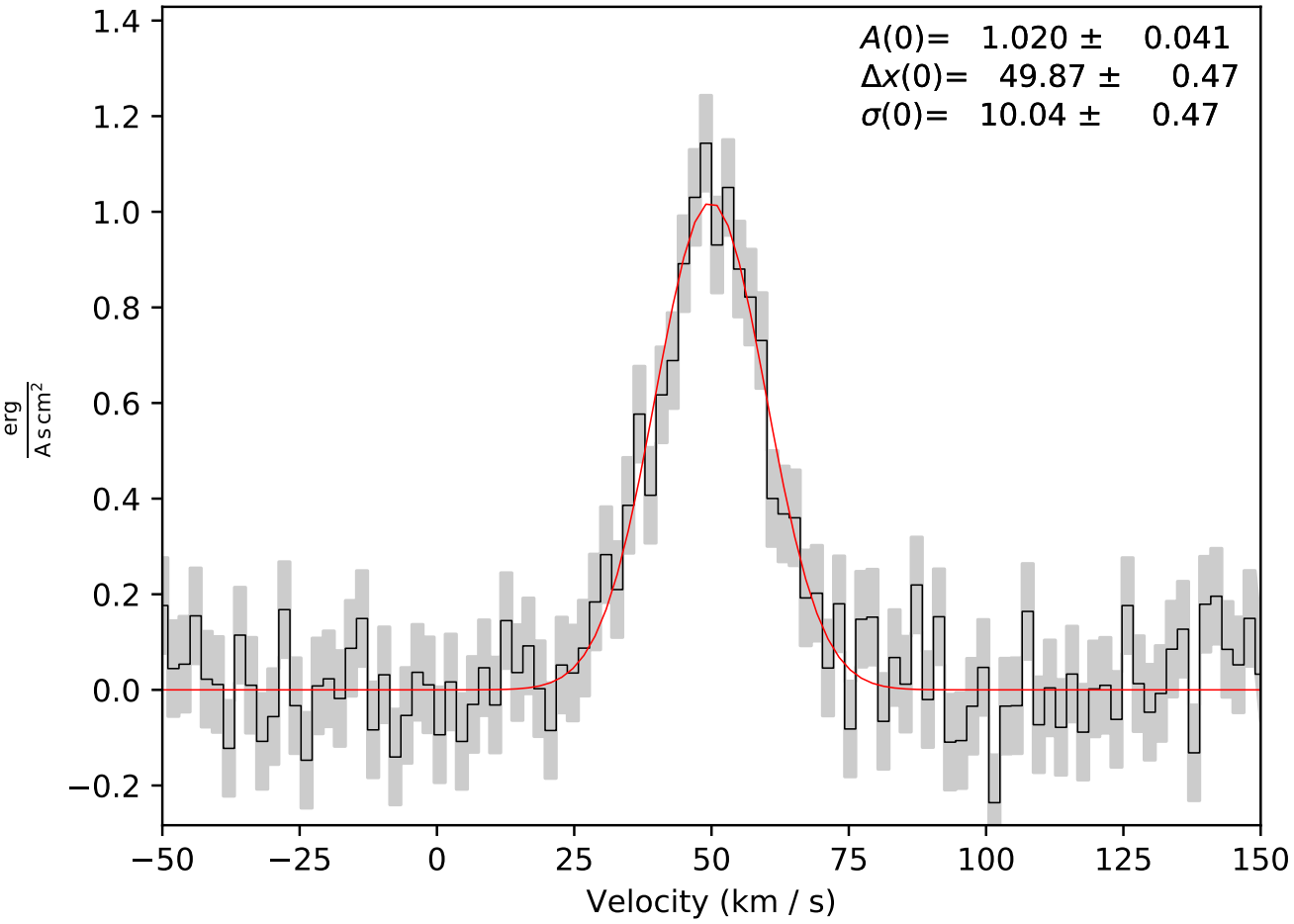

Figure 2. One-dimensional Gaussian profile fit to a synthetic spectrum. The parameter values and errors are shown in the upper right. The number of significant figures displayed in both the value and the error is automatically set to one digit more than the last significant digit in the error.

While the errors on most parameters appear reasonable, there is a very large error on the excitation temperature $T_{\rm ex}$, which is driven by the degeneracy of $T_{\rm ex}$ with $N_{\rm tot}$. The asymmetry of the error on $T_{\rm ex}$ is apparent in Figure 5, but it is not captured by the optimizer's reported error results; the asymmetry occurs because T_{ex} is in the exponent in the model equations.

In such situations, it can be beneficial to measure the parameter errors in different ways. Using the `emcee` and `pymc` wrappers can help do this. Examples of how to use these Monte Carlo samplers to acquire better parameter errors once an optimization has already been performed are available in the online documentation: see http://pyspeckit.readthedocs.io/en/latest/example_pymc.html.

More sophisticated examples, including fitting of a non-LTE ammonia spectrum in which $T_{\rm ex} < T_R$, are available in the example directory of `pyspeckit` (https://github.com/pyspeckit/pyspeckit/tree/master/examples), specifically https://github.com/pyspeckit/pyspeckit/tree/master/examples/synthetic_LTE_ammonia_spectrum_example_witherrorestimates.py and https://github.com/pyspeckit/pyspeckit/tree/master/examples/synthetic_nLTE_ammonia_spectrum_example_witherrorestimates.py. These examples also include demonstrations of how to force the optimizer to ignore nonphysical values.

C. COMPARISON OF N_2H^+ (1-0) RESULTS WITH CLASS

One of the most frequently used line transitions for the study of dense gas kinematics is N_2H^+ (1-0) at 93.17 GHz. The transition displays several hyperfine components with well determined relative frequencies and weights. The

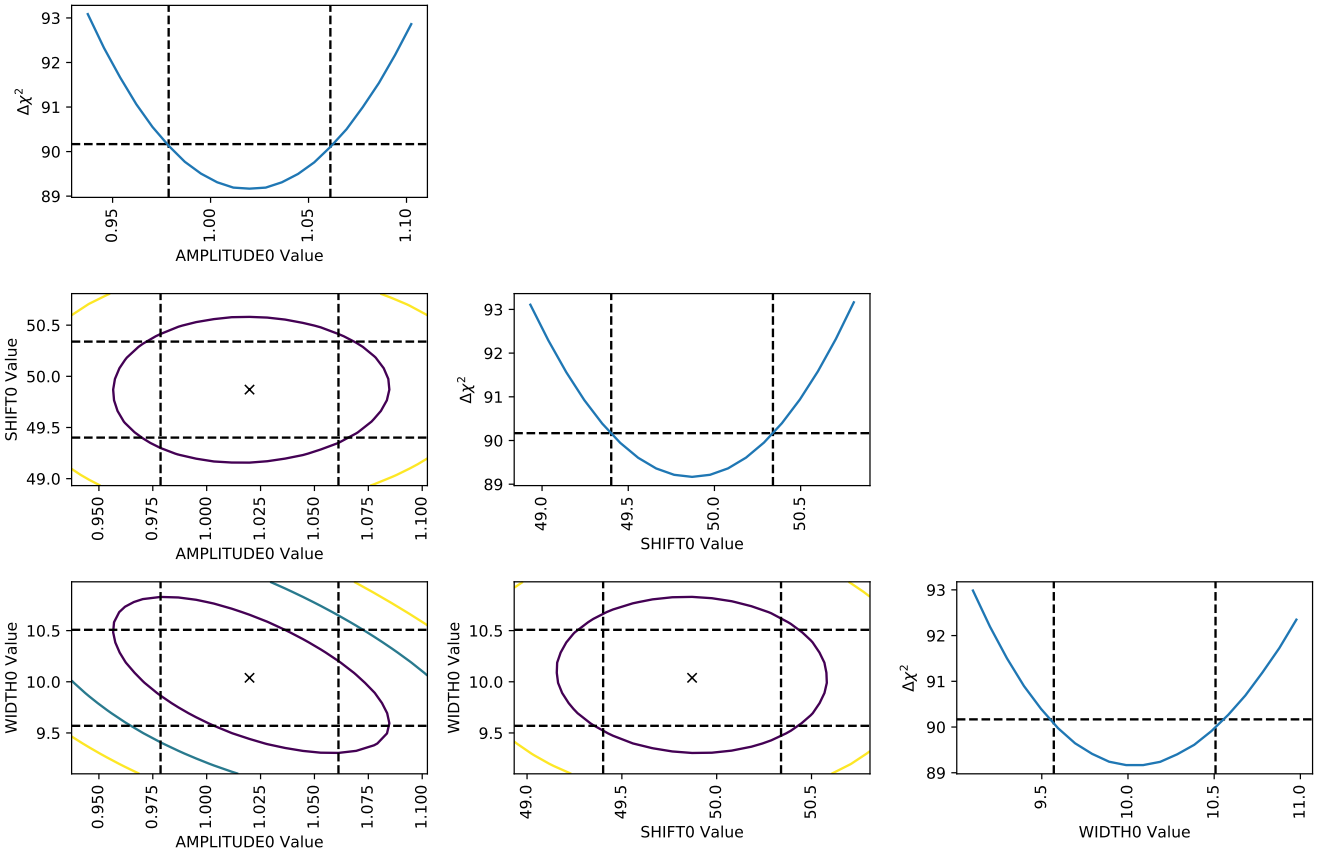

Figure 3. Error estimate figure. In all panels, the vertical dashed lines show the estimated 1σ errors from the optimizer, while the horizontal dashed lines show the value $\Delta\chi^2 = 1$, which corresponds to the 68% confidence interval for that parameter. In the off-diagonal panels, contours are shown at $\Delta\chi^2 = 2.3, 6.2, 11.8$, corresponding to 68%, 95%, and 99.5% (1σ, 2σ, and 3σ for a Gaussian) confidence regions. See Appendix A for details and interpretation.

standard approach for analyzing this line has been to use the `HFS` mode within CLASS. Here we show that using the N_2H^+ `hyperfine` model in `pyspeckit`, we obtain the same results in both optically thin and thick models.

The main difference between the CLASS and `pyspeckit` parametrization is that the former does not report excitation temperature (T_{ex}), but the area of the line profile. The reported area is $\tau \times T_{ant}$, where

$$T_{ant} = J(T_{ex}) - J(T_{background}) \ , \tag{C1}$$

where

$$J_\nu(T) = \frac{h\,\nu}{k_B} \frac{1}{(e^{h\,\nu/k_B\,T} - 1)} \ . \tag{C2}$$

We derive the equivalent T_{ex} from the reported line fit parameters. Moreover, in the optically thin case both fits are performed using the common assumption of $\tau = 0.1$ as a fixed parameter.

Tables C and 1 show the results of fitting an example spectrum in both CLASS and `pyspeckit`. The resulting fits differ by $< 1\%$ in most parameters, with a slightly greater discrepancy in the velocity centroid but consistent within the reported fit uncertainties.

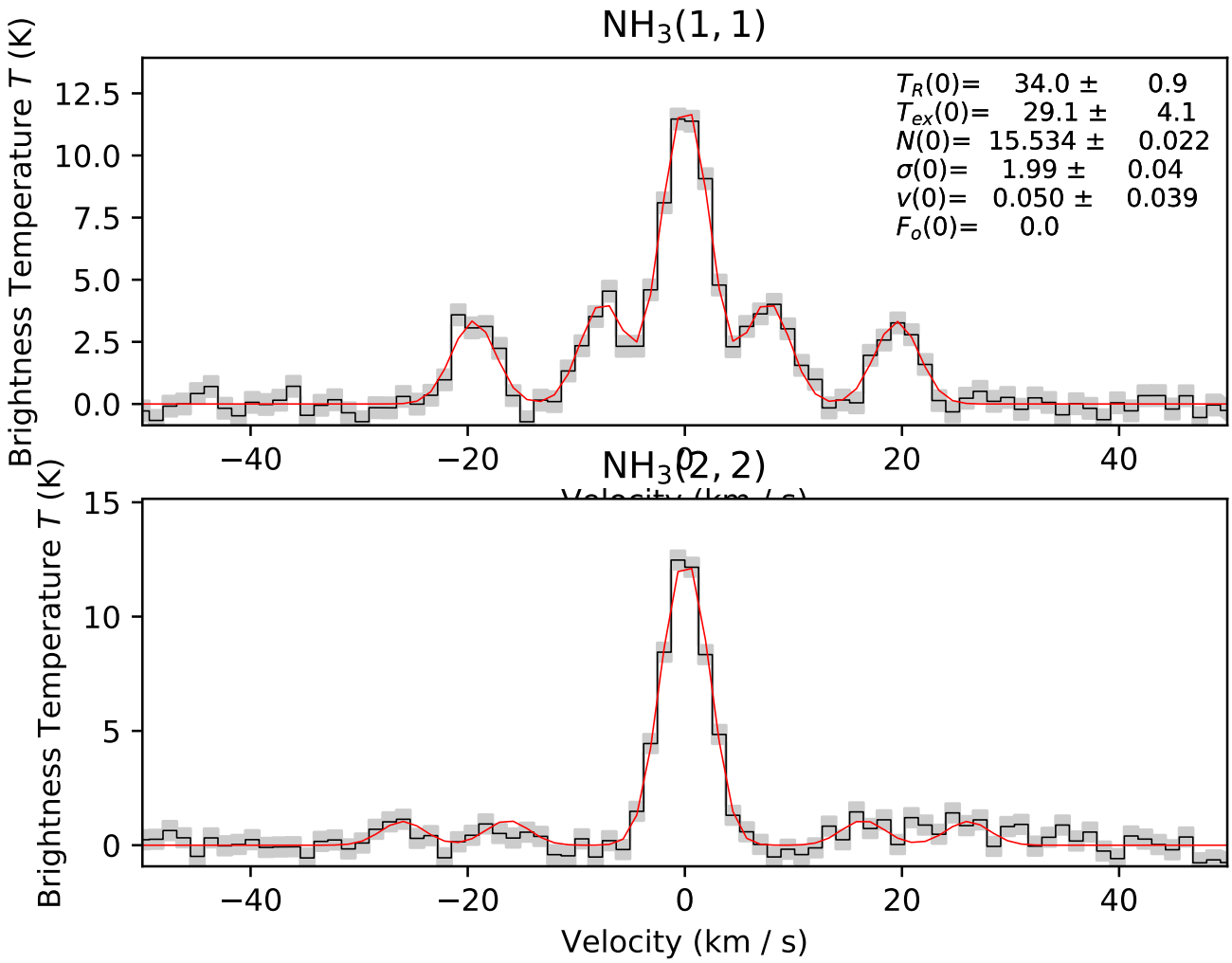

Figure 4. Ammonia model profile fit to a synthetic spectrum. The parameter values and errors are shown in the upper right. The associated error estimate triangle diagram is shown in Figure 5. The correct parameters are $T_{\rm R} = T_{\rm ex} = 35$, $N = 15$, $\sigma_v = 2$, and $v = 0$, all of which are reasonably recovered. However, note that $T_{\rm ex} > T_{\rm R}$ is generally nonphysical, yet the allowed parameter space for $T_{\rm ex}$ includes such values. EWR: Not showing the empty spectral region would make a nicer visualization, IMHO.

Table 1. Best fit parameters in optically thin model (3 parameters)

Parameter	Input value	pyspeckit fit	CLASS fit
T_{ex}	9.0	3.454±0.014	3.451
V_c	0.0	0.0016	0.0028±0.0068
σ_v	0.3	0.2942±0.0062	0.2930±0.0060
τ_{all}	0.01	0.1	0.1
Area			0.0607±0.0012

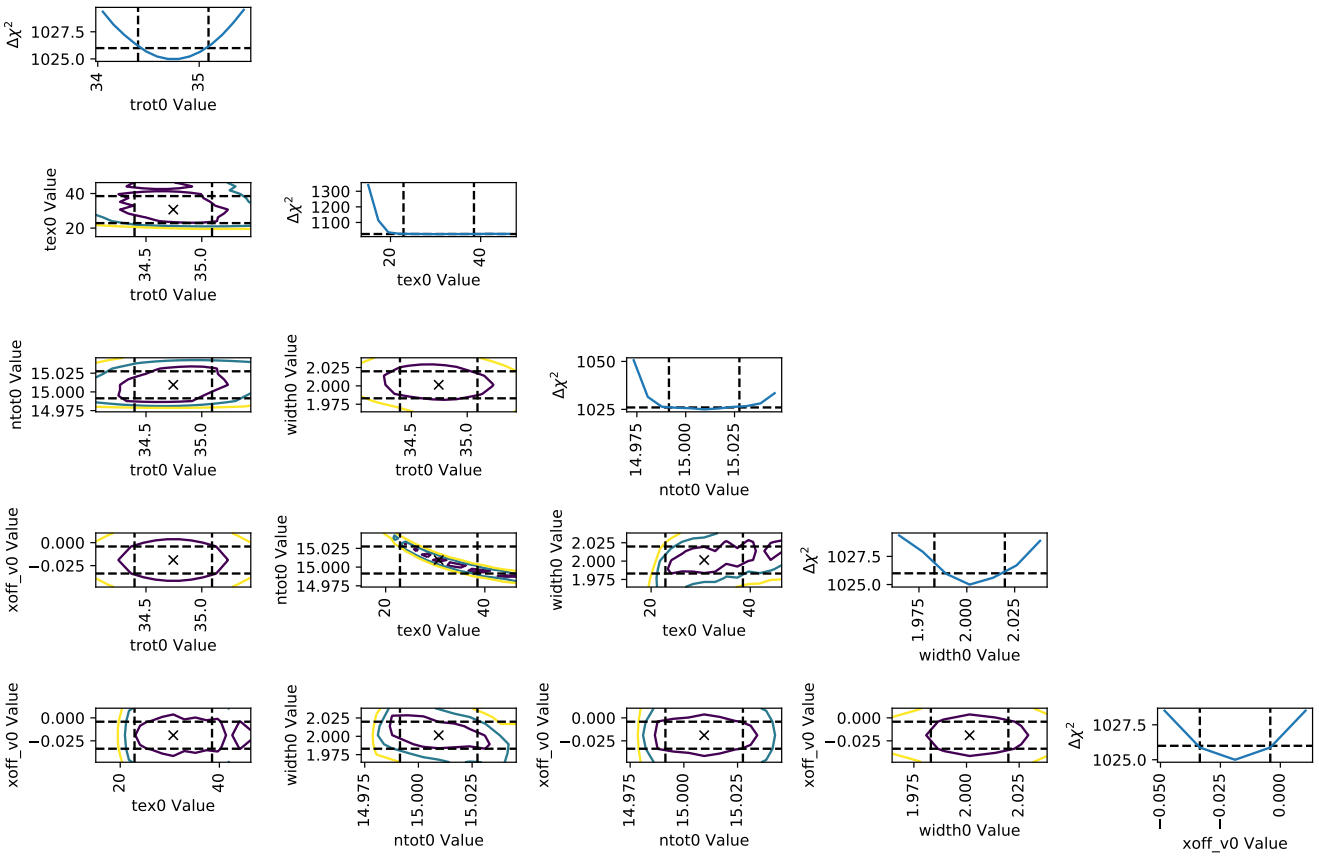

Figure 5. Error estimate figure for the default NH_3 model. The panels are labeled as in Figure 3. The most relevant panel is the ntot0 vs tex0 panel, which plots the column N against the excitation temperature $T_{\rm ex}$: both of these parameters govern the peak amplitude of the spectrum, so they are degenerate. The vertical dashed lines do not match the $\Delta\chi^2 = 1$ positions for $T_{\rm ex}$ or N partly because of this degeneracy; the fit errors reported by the Levenberg-Marquardt algorithm are larger than the directly computed marginal errors.

Table 2. Best fit parameters in optically thick model (4 parameters)

Parameter	Input value	pyspeckit fit	CLASS fit
T_{ex}	9.0	9.19±0.19	9.1833
V_c	0.0	0.00042	-0.000±0.0063
σ_v	0.3	0.3047±0.0062	0.3041±0.0063
τ_{all}	9.0	8.13±0.59	8.10±0.59
Area			49.0±2.25