

---

# **pyspeckit Documentation**

***Release 0.1***

**Adam Ginsburg and Jordan Mirocha**

January 27, 2012



# CONTENTS

<b>1</b>	<b>Classes</b>	<b>3</b>
<b>2</b>	<b>Features</b>	<b>5</b>
2.1	Installation and Requirements . . . . .	5
2.2	Models . . . . .	6
2.3	Classes . . . . .	11
2.4	Features . . . . .	11
2.5	Readers . . . . .	19
2.6	Wrappers . . . . .	23
	<b>Python Module Index</b>	<b>27</b>
	<b>Index</b>	<b>29</b>



An extensible spectroscopic analysis toolkit for astronomy.

If you're just getting started, see the [examples page](#).

Download or see [Installation and Requirements](#)

Supported file types and their formats:

- *FITS*
- *Plain Text*
- *hdf5*



# CLASSES

- `spectrum` can read a variety of individual spectra types
  - `Spectrum` The `Spectrum` class, which is the core of `pyspeckit`. The `__init__` procedure opens a spectrum file.
  - `Spectra` A group of `Spectrums`. Generally for when you have multiple wavelength observations you want to stitch together (e.g., two filterbanks on a heterodyne system, or the red/blue spectra from a multi-band spectrometer like the Double Imaging Spectrograph)
  - `ObsBlock` An Observation Block - multiple spectra of different objects or different times covering the same wavelength range
- `cubes` is used to deal with data cubes and has functionality similar to [GAIA](#) and [ds9](#).
  - `Cube` A Cube of Spectra. Has features to collapse the cube along the spectral axis and fit spectra to each element of the cube. Is meant to replicate [Starlink's GAIA](#) in some ways, but with less emphasis on speed and much greater emphasis on spectral line fitting.





# FEATURES

- *Baseline Fitting* describes baseline & continuum fitting.
- *Model Fitting* describes the general process of model fitting.
- *Measurements* is a toolkit for performing EQW, column, and other measurements...
- *Units* contains the all-important `SpectroscopicAxis` class that is used to deal with coordinate transformations
- *Registration* describes the extensible qualities of pyspeckit

## 2.1 Installation and Requirements

PySpecKit requires at least the basic scientific packages

- `numpy`
- `matplotlib`
- `mpfit` is included
- `scipy` is optional. It is only required for RADEX grid interpolation

You'll most likely want at least one of the following packages to enable file reading

- `pyfits`
- `atpy` (which depends on `asciitable` [\[github link\]](#) )
- `hdf5`

You can acquire the code with this clone command:

```
hg clone https://bitbucket.org/pyspeckit/pyspeckit.bitbucket.org pyspeckit
cd pyspeckit
python setup.py install
```

Or you can Download the latest tarball version, then extract and install using the standard python method:

```
wget https://bitbucket.org/pyspeckit/pyspeckit.bitbucket.org/get/tip.tar.gz
tar -xzf pyspeckit-pyspeckit.bitbucket.org-tip.tar.gz
cd pyspeckit-pyspeckit.bitbucket.org-tip
python setup.py install
```

You can also check out the [source code](#)

## 2.2 Models

The generic `SpectralModel` class is a wrapper for model functions. A model should take in an X-axis and some number of parameters. In order to declare a `SpectralModel`, you give `SpectralModel` the function name and the number of parameters it requires. The rest of the options are optional, though `parnames` & `shortvarnames` are strongly recommended. If you do not specify `fitunits`, your fitting code must deal with units internally.

Here are some examples of how to make your own fitters:

```
hill15_fitter = model.SpectralModel(hill15_model, 5,
    parnames=['tau', 'v_lsr', 'v_infall', 'sigma', 'tpeak'],
    parlimited=[(True,False), (False,False), (True,False), (True,False), (True,False)],
    parlimits=[(0,0), (0,0), (0,0), (0,0), (0,0)],
    # specify the parameter names (TeX is OK)
    shortvarnames=("\\tau", "v_{lsr}", "v_{infall}", "\\sigma", "T_{peak}"),
    fitunits='Hz' )

gaussfitter = model.SpectralModel(gaussian, 3,
    parnames=['amplitude', 'shift', 'width'],
    parlimited=[(False,False), (False,False), (True,False)],
    parlimits=[(0,0), (0,0), (0,0)],
    shortvarnames=('A', r'\Delta x', r'\sigma'),
    multisingle=multisingle,
    )
```

Then you can register these fitters.

### 2.2.1 Generic `SpectralModel` wrapper

```
class pyspeckit.spectrum.models.model.SpectralModel(modelfunc, npars, shortvar-
    names=('A', '\Delta x', '\sigma'),
    multisingle='multi', **kwargs)
```

A wrapper class for a spectra model. Includes internal functions to generate multi-component models, annotations, integrals, and individual components. The declaration can be complex, since you should name individual variables, set limits on them, set the units the fit will be performed in, and set the annotations to be used. Check out some of the hyperfine codes (hcn, n2hp) for examples.

`modelfunc`: the model function to be fitted. Should take an X-axis (spectroscopic axis) as an input, followed by input parameters. `npars` - number of parameters required by the model

`parnames` - a list or tuple of the parameter names

`parvalues` - the initial guesses for the input parameters (defaults to ZEROS)

`parlimits` - the upper/lower limits for each variable (defaults to ZEROS)

`parfixed` - Can declare any variables to be fixed (defaults to ZEROS)

`parerror` - technically an output parameter... hmm (defaults to ZEROS)

**partied - not the past tense of party. Can declare, via text, that** some parameters are tied to each other. Defaults to zeros like the others, but it's not clear if that's a sensible default

`fitunits` - convert X-axis to these units before passing to model

`parsteps` - minimum step size for each parameter (defaults to ZEROS)

`npeaks` - default number of peaks to assume when fitting (can be overridden)

`shortvarnames` - TeX names of the variables to use when annotating

**multisingle** - Are there multiple peaks (no background will be fit) or just a single peak (a background may/will be fit)

**annotations** (*shortvarnames=None*)  
Return a list of TeX-formatted labels

**components** (*xarr, pars*)  
Return a numpy ndarray of the independent components of the fits

**fitter** (*xax, data, err=None, quiet=True, veryverbose=False, debug=False, parinfo=None, \*\*kwargs*)  
Run the fitter. Must pass the x-axis and data. Can include error, parameter guesses, and a number of verbosity parameters.

**quiet** - pass to mpfit. If False, will print out the parameter values for each iteration of the fitter

**veryverbose** - print out a variety of mpfit output parameters

**debug** - raise an exception (rather than a warning) if  $\chi^2$  is nan

**accepts *tied, limits, limited, and fixed* as keyword arguments.** They must be lists of length `len(params)`

**parinfo** - You can override the class parinfo dict with this, though that largely defeats the point of having the wrapper class. This class does NO checking for whether the parinfo dict is valid.

kwargs are passed to mpfit after going through `_make_parinfo` to strip out things used by this class

**integral** (*modelpars, \*\*kwargs*)  
Extremely simple integrator: IGNORES modelpars; just sums self.model

**mpfitfun** (*x, y, err=None*)  
Wrapper function to compute the fit residuals in an mpfit-friendly format

**n\_modelfunc** (*pars, \*\*kwargs*)  
Simple wrapper to deal with N independent peaks for a given spectral model

**slope** (*xinp*)  
Find the local slope of the model at location x (x must be in xax's units)

## 2.2.2 Ammonia inversion transition TKIN fitter

Ammonia inversion transition TKIN fitter translated from Erik Rosolowsky's <http://svn.ok.ubc.ca/svn/signals/nh3fit/>

### Module API

```
pyspeckit.spectrum.models.ammonia.ammonia(xarr,          tkin=20,          tex=None,
                                             ntot=10000000000000.0,      width=1,
                                             xoff_v=0.0, fortho=0.0, tau=None, fillingfrac-
                                             tion=None, return_tau=False, thin=False,
                                             verbose=False, return_components=False,
                                             debug=False)
```

Generate a model Ammonia spectrum based on input temperatures, column, and gaussian parameters

ntot can be specified as a column density (e.g.,  $10^{15}$ ) or a log-column-density (e.g., 15)

**tex** can be specified or can be assumed LTE if unspecified, if `tex > tkin`, or if “thin” is specified

“thin” uses a different parameterization and requires only the optical depth, width, offset, and tkin to be specified. In the ‘thin’ approximation, tex is not used in computation of the partition function - LTE is implicitly assumed

If tau is specified, ntot is NOT fit but is set to a fixed value fillingfraction is an arbitrary scaling factor to apply to the model fortho is the ortho/(ortho+para) fraction. The default is to assume all ortho. xoff\_v is the velocity offset in km/s

tau refers to the optical depth of the 1-1 line. The optical depths of the other lines are fixed relative to tau\_oneone (not implemented) if tau is specified, ntot is ignored

### 2.2.3 SimpleFitter wrapper

Adds a variable height (background) component to any model

### 2.2.4 Formaldehyde cm-line fitter

This is a formaldehyde 1<sub>11</sub>-1<sub>10</sub> / 2<sub>12</sub>-2<sub>11</sub> fitter. It includes hyperfine components of the formaldehyde lines and has both LTE and RADEX LVG based models

```
pyspeckit.spectrum.models.formaldehyde.formaldehyde(xarr, amp=1.0,
                                                    xoff_v=0.0, width=1.0, re-
                                                    turn_components=False)
```

Generate a model Formaldehyde spectrum based on simple gaussian parameters

the “amplitude” is an essentially arbitrary parameter; we therefore define it to be Tex given tau=0.01 when passing to the fitter The final spectrum is then rescaled to that value

```
pyspeckit.spectrum.models.formaldehyde.formaldehyde_radex(xarr, density=4,
                                                            column=13,
                                                            xoff_v=0.0, width=1.0,
                                                            grid_vwidth=1.0,
                                                            grid_vwidth_scale=False,
                                                            texgrid=None, tau-
                                                            grid=None, hdr=None,
                                                            path_to_texgrid='',
                                                            path_to_taugrid='',
                                                            tempera-
                                                            ture_gridnumber=3,
                                                            debug=False, ver-
                                                            bose=False, **kwargs)
```

Use a grid of RADEX-computed models to make a model line spectrum

The RADEX models have to be available somewhere. OR they can be passed as arrays. If as arrays, the form should be: texgrid = ((minfreq1,maxfreq1,texgrid1),(minfreq2,maxfreq2,texgrid2))

xarr must be a SpectroscopicAxis instance xoff\_v, width are both in km/s

**grid\_vwidth is the velocity assumed when computing the grid in km/s** this is important because tau = mod-eltau / width (see, e.g., Draine 2011 textbook pgs 219-230)

grid\_vwidth\_scale is True or False: False for LVG, True for Sphere

### 2.2.5 Gaussian Fitter

The simplest and most useful model.

Until 12/23/2011, gaussian fitting used the complicated and somewhat bloated gaussfitter.py code. Now, this is a great example of how to make your own model! Just make a function like gaussian and plug it into the SpectralModel class.

```
pyspeckit.spectrum.models.inherited_gaussfitter.gaussian(x, A, dx, w, re-
    turn_components=False)
```

Returns a 1-dimensional gaussian of form  $H+A*\text{numpy.exp}(-(x-dx)**2/(2*w**2))$

[height,amplitude,center,width]

return\_components does nothing but is required by all fitters

```
pyspeckit.spectrum.models.inherited_gaussfitter.gaussian_fitter(multisingle='multi')
    Generator for Gaussian fitter class
```

## 2.2.6 HCN Hyperfine Fitter

This is an HCN fitter... ref for line params: <http://www.strw.leidenuniv.nl/~moldata/datafiles/hcn@hfs.dat>

```
pyspeckit.spectrum.models.hcn.aval_dict = {'10-01': 2.4074999999999999e-05, '12-01': 2.4074999999999999e-05,
```

Line strengths of the 15 hyperfine components in  $J = 1 - 0$  transition. The thickness of the lines indicates their relative weight compared to the others. Line strengths are normalized in such a way that summing over all initial  $J = 1$  levels gives the degeneracy of the  $J = 0$  levels, i.e., for  $JF1F = 012$ , three for  $JF1F = 011$ , and one for  $JF1F = 010$ . Thus, the sum over all 15 transitions gives the total spin degeneracy

```
pyspeckit.spectrum.models.hcn.hcn_radex(xarr,          density=4,          column=13,
    xoff_v=0.0,    width=1.0,    grid_vwidth=1.0,
    grid_vwidth_scale=False,    texgrid=None,    tau-
    grid=None,    hdr=None,    path_to_texgrid='',
    path_to_taugrid='',    temperature_gridnumber=3,
    debug=False, verbose=False, **kwargs)
```

Use a grid of RADEX-computed models to make a model line spectrum

The RADEX models have to be available somewhere. OR they can be passed as arrays. If as arrays, the form should be: `texgrid = ((minfreq1,maxfreq1,texgrid1),(minfreq2,maxfreq2,texgrid2))`

`xarr` must be a `SpectroscopicAxis` instance `xoff_v`, `width` are both in km/s

**grid\_vwidth is the velocity assumed when computing the grid in km/s** this is important because  $\tau = \text{mod-eltau} / \text{width}$  (see, e.g., Draine 2011 textbook pgs 219-230)

`grid_vwidth_scale` is True or False: False for LVG, True for Sphere

## 2.2.7 Hill5 analytic infall model

Code translated from: [https://bitbucket.org/devries/analytic\\_infall/overview](https://bitbucket.org/devries/analytic_infall/overview)

Original source: <http://adsabs.harvard.edu/abs/2005ApJ...620..800D>

```
pyspeckit.spectrum.models.hill5infall.hill5_model(xarr, tau, v_lsr, v_infall, sigma,
    tpeak, TBG=2.73)
```

The rest of this needs to be translated from C

```
pyspeckit.spectrum.models.hill5infall.jfunc(t, nu)
    t- kelvin nu - Hz?
```

## 2.2.8 Generalized hyperfine component fitter

```
class pyspeckit.spectrum.models.hyperfine.hyperfinemodel(line_names,
    voff_lines_dict,    freq_dict,
    line_strength_dict)
```

Wrapper for the hyperfine model class. Specify the offsets and relative strengths when initializing, then you've

got yourself a hyperfine modeler.

Initialize the various parameters defining the hyperfine transitions

line\_names is a LIST of the line names to be used as indices for the dictionaries

**voff\_lines\_dict** is a **linename:v\_off** dictionary of velocity offsets for the hyperfine components. Technically, this is redundant with freq\_dict

freq\_dict - frequencies of the individual transitions

**line\_strength\_dict** - Relative strengths of the hyperfine components, usually determined by their degeneracy and Einstein A coefficients

**hyperfine** (xarr, Tex=5.0, tau=0.10000000000000001, xoff\_v=0.0, width=1.0, return\_components=False, Tbackground=2.73, amp=None)

Generate a model spectrum given an excitation temperature, optical depth, offset velocity, and velocity width.

**hyperfine\_amp** (xarr, amp=None, xoff\_v=0.0, width=1.0, return\_components=False, Tbackground=2.73, Tex=5.0, tau=0.10000000000000001)

wrapper of self.hyperfine with order of arguments changed

## 2.2.9 Lorentzian Fitter

The simplest and most useful model.

Until 12/23/2011, lorentzian fitting used the complicated and somewhat bloated gaussfitter.py code. Now, this is a great example of how to make your own model!

pyspeckit.spectrum.models.inherited\_lorentzian.**lorentzian** (x, A, dx, w, return\_components=False)

Returns a 1-dimensional lorentzian of form  $A * 2 * \pi * w / ((x - dx)^2 + ((w/2)^2))$

[amplitude, center, width]

return\_components does nothing but is required by all fitters

pyspeckit.spectrum.models.inherited\_lorentzian.**lorentzian\_fitter** (multisingle='multi')

Generator for lorentzian fitter class

### 2.2.10 Model Grid

Fit a line based on parameters output from a grid of models

pyspeckit.spectrum.models.modelgrid.**gaussian\_line** (xax, maxamp, tau, offset, width)

A Gaussian line function in which the

pyspeckit.spectrum.models.modelgrid.**line\_model\_2par** (xax, center, width, gridval1, gridval2, griddim1, griddim2, maxampgrid, taugrid, linefunction=<function gaussian\_line at 0x10d64bc08>)

Returns the spectral line that matches the given x-axis

xax, center, width must be in the same units!

pyspeckit.spectrum.models.modelgrid.**line\_params\_2D** (gridval1, gridval2, griddim1, griddim2, valuegrid)

Given a 2D grid of modeled line values - the amplitude, e.g. excitation temperature, and the optical depth, tau - return the model spectrum

griddims contains the names of the axes and their values... it should have the same number of entries as gridpars

### 2.2.11 N2H+ fitter

Reference for line params: Daniel, F., Dubernet, M.-L., Meuwly, M., Cernicharo, J., Pagani, L. 2005, MNRAS 363, 1083 <http://www.strw.leidenuniv.nl/~moldata/N2H+.html> <http://adsabs.harvard.edu/abs/2005MNRAS.363.1083D>

```
pyspeckit.spectrum.models.n2hp.aval_dict = {'121-011': 1.512, '121-010': 2.0150000000000001, '121-012': 0.1010
```

Line strengths of the 15 hyperfine components in J=1-0 transition. The thickness of the lines indicates their relative weight compared to the others. Line strengths are normalized in such a way that summing over all initial J = 1 levels gives the degeneracy of the J = 0 levels, i.e., for JF1F 012, three for JF1F 011, and one for JF1F 010. Thus, the sum over all 15 transitions gives the total spin degeneracy

```
pyspeckit.spectrum.models.n2hp.n2hp_radex(xarr, density=4, column=13,
                                           xoff_v=0.0, width=1.0, grid_vwidth=1.0,
                                           grid_vwidth_scale=False, tex-
                                           grid=None, taugrid=None, hdr=None,
                                           path_to_texgrid='', path_to_taugrid='',
                                           temperature_gridnumber=3, debug=False,
                                           verbose=False, **kwargs)
```

Use a grid of RADEX-computed models to make a model line spectrum

The RADEX models have to be available somewhere. OR they can be passed as arrays. If as arrays, the form should be: texgrid = ((minfreq1,maxfreq1,texgrid1),(minfreq2,maxfreq2,texgrid2))

xarr must be a SpectroscopicAxis instance xoff\_v, width are both in km/s

**grid\_vwidth is the velocity assumed when computing the grid in km/s** this is important because  $\tau = \text{mod-eltau} / \text{width}$  (see, e.g., Draine 2011 textbook pgs 219-230)

grid\_vwidth\_scale is True or False: False for LVG, True for Sphere

### 2.2.12 Voigt Profile Fitter

```
pyspeckit.spectrum.models.inherited_voigtfitter.voigt(xarr, amp, xcen, Gfwhm,
                                                       Lfwhm)
```

voigt profile

$$V(x, \text{sig}, \text{gam}) = \text{Re}(w(z)) / (\text{sig} * \sqrt{2 * \pi}) \quad z = (x + i * \text{gam}) / (\text{sig} * \sqrt{2})$$

Converted from <http://mail.scipy.org/pipermail/scipy-user/2011-January/028327.html>

```
pyspeckit.spectrum.models.inherited_voigtfitter.voigt_fitter(multisingle='multi')
```

Generator for voigt fitter class

## 2.3 Classes

## 2.4 Features

### 2.4.1 Baseline Fitting

There are a number of cool features in baselining that aren't well-described below, partly due to Sphinx errors as of 12/22/2011.

`exclude` and `include` allow you to specify which parts of the spectrum to use for baseline fitting. Enter values as pairs of coordinates.

`Excludefit` makes use of an existing fit and excludes all points with signal above a (very low) threshold when fitting the baseline. Going back and forth between `baseline(excludefit=True)` and `specfit()` is a nice way to iteratively measure the baseline & emission/absorption line components.

## API

**class** `pyspeckit.spectrum.baseline.Baseline(Spectrum)`

Class to measure and subtract baselines from spectra.

While the term ‘baseline’ is generally used in the radio to refer to broad-band features in a spectrum not necessarily associated with a source, in this package it refers to general continuum fitting. In principle, there’s no reason to separate ‘continuum’ and ‘spectral feature’ fitting into different categories (both require some model, data, and optional weights when fitting). In practice, however, ‘continuum’ is frequently something to be removed and ignored, while spectral features are the desired measurable quantity. In order to accurately measure spectral features, it is necessary to allow baselines of varying complexity.

The `Baseline` class has both interactive and command-based data selection features. It can be used to fit both polynomial and power-law continua. Blackbody fitting is not yet implemented [12/21/2011]. Baseline fitting is a necessary prerequisite for Equivalent Width measurement.

As you may observe in the comments on this code, this has been one of the buggiest and least adequately tested components of `pyspeckit`. Bug reports are welcome. (as of 1/15/2012, a major change has probably fixed most of the bugs, and the code base is much simpler)

**\_\_call\_\_** (*\*args, \*\*kwargs*)

Fit and remove a polynomial from the spectrum. It will be saved in the variable “self.basespec” and the fit parameters will be saved in “self.order”

**function** `baseline(spectrum,xarr=None,xmin=None,xmax=None,order=1,quiet=True,exclude=None):`  
Subtract a baseline from a spectrum If `xmin,xmax` are not specified, defaults to ignoring first and last 10% of spectrum

`exclude` is a set of start/end indices to ignore when baseline fitting (ignored by setting error to infinite in fitting procedure)

`excludefit` creates a mask based on the fitted gaussian model (assuming that it has a zero-height) using an exclusion level of (`exclusionlevel`) \* the smallest gaussian peak that was fit

`fit_plotted_area` means that a fit will only be attempted on the region currently shown in the plot. Overridden by `interactive=True` or `exclude='interactive'`

if `fit_original` is set, “basespec” is added back to the spectrum before fitting so you can run this procedure multiple times without losing information

**\_\_init\_\_** (*Spectrum*)

**\_\_module\_\_** = ‘pyspeckit.spectrum.baseline’

**annotate** (*loc='upper left'*)

**button2action** (*event=None, debug=False, subtract=True, fit\_original=False, powerlaw=False, baseline\_fit\_color='orange', exclude=None, \*\*kwargs*)

Do the baseline fitting and save and plot the results.

Can specify a region to exclude using velocity units or pixel units

**button3action** (*event=None, debug=False, subtract=True, fit\_original=False, powerlaw=False, baseline\_fit\_color='orange', exclude=None, \*\*kwargs*)

Do the baseline fitting and save and plot the results.



Can specify a region to exclude using velocity units or pixel units

**clearlegend** ()

**crop** (*x1pix*, *x2pix*)

When `spectrum.crop` is called, this must be too

**downsample** (*factor*)

**plot\_baseline** (*annotate=True*, *baseline\_fit\_color='orange'*, *\*\*kwargs*)

Overplot the baseline fit

**savefit** ()

**unsubtract** ()

## 2.4.2 Model Fitting

**class** `pyspeckit.spectrum.fitters.Specfit` (*Spectrum*, *Registry=None*)

Bases: `pyspeckit.spectrum.interactive.Interactive`

**EQW** (*plot=False*, *plotcolor='g'*, *annotate=False*, *alpha=0.5*, *loc='lower left'*)

Returns the equivalent width (integral of “baseline” or “continuum” minus the spectrum) over the selected range

**annotate** (*loc='upper right'*, *labelspacing=0.25*, *markerscale=0.01*, *borderpad=0.10000000000000001*, *handlelength=0.10000000000000001*, *textpad=0.10000000000000001*, *frameon=False*, *\*\*kwargs*)

Add a legend to the plot showing the fitted parameters

`clearlegend()` will remove the legend

*kwargs* passed to legend

**button3action** (*event*, *debug=False*)

Disconnect the interactiveness Perform the fit (or die trying) Hide the guesses

**clear** (*legend=True*, *components=True*)

Remove the fitted model from the plot

Also removes the legend by default

**clear\_all\_connections** (*debug=False*)

Prevent overlapping interactive sessions

**clear\_highlights** ()

**crop** (*x1pix*, *x2pix*)

When `spectrum.crop` is called, this must be too

**downsample** (*factor*)

Downsample the model spectrum (and the spectrofit spectra) This should only be done when `Spectrum.smooth` is called

**event\_manager** (*event*, *debug=False*)

Decide what to do given input (click, keypress, etc.)

**firstclick\_guess** ()

Initialize `self.guesses`

**firstclick\_selection** (*include\_all=False*)

Initialize the include/exclude mask

**fullsizemodel** ()

If the gaussian was fit to a sub-region of the spectrum, expand it (with zeros) to fill the spectrum.

**guesspeakwidth** (*event*, *debug=False*)

Interactively guess the peak height and width from user input

Width is assumed to be half-width-half-max

**highlight\_fitregion** (*drawstyle='steps-mid'*, *color='g'*, *clear\_highlights=True*, *\*\*kwargs*)

Re-highlight the fitted region

kwargs are passed to *matplotlib.plot*

**integral** (*direct=False*, *threshold='auto'*, *integration\_limits=[]*, *return\_error=False*, *\*\*kwargs*)

Return the integral of the fitted spectrum

if *direct=True*, return the integral of the spectrum over a range defined by the threshold or integration limits if defined

note that *integration\_limits* will operate directly on the DATA, which means that if you've baselined without *subtract=True*, the baseline will be included in the integral

if *return\_error* is set, the error computed by  $\sigma = \sqrt{\sum(\sigma_i^2)} * dx$  will be returned as well

**moments** (*\*\*kwargs*)

Return the moments

see *moments*

**multifit** (*fitype=None*, *renormalize='auto'*, *annotate=None*, *color='k'*, *composite\_fit\_color='red'*, *component\_fit\_color='red'*, *lw=0.5*, *composite\_lw=0.75*, *component\_lw=0.75*, *show\_components=None*, *verbose=True*, *\*\*kwargs*)

Fit multiple gaussians (or other profiles)

**fitype** - What function will be fit? **fitype must have been Registryed in the** *singlefitters* dict. Uses default ('gaussian') if not specified

**renormalize** - if 'auto' or True, will attempt to rescale small data (<1e-9) to be closer to 1 (scales by the median) so that the fit converges better

**peakbgfit** (*usemoments=True*, *annotate=None*, *vheight=True*, *height=0*, *negamp=None*, *fitype=None*, *renormalize='auto'*, *color='k'*, *composite\_fit\_color='red'*, *component\_fit\_color='blue'*, *lw=1.0*, *composite\_lw=1.0*, *component\_lw=1.0*, *show\_components=None*, *debug=False*, *\*\*kwargs*)

Fit a single peak (plus a background)

**usemoments** - The initial guess will be set by the fitter's 'moments' function (this overrides 'guesses')

*annotate* - Make a legend? *vheight* - Fit a (constant) background as well as a peak? *height* - initial guess for background *negamp* - If True, assumes amplitude is negative. If False, assumes positive. If

None, can be either.

**fitype** - What function will be fit? **fitype must have been Registryed in the** *singlefitters* dict

**renormalize** - if 'auto' or True, will attempt to rescale small data (<1e-9) to be closer to 1 (scales by the median) so that the fit converges better

**plot\_fit** (*annotate=None*, *show\_components=None*, *color='k'*, *composite\_fit\_color='red'*, *component\_fit\_color='blue'*, *lw=0.5*, *composite\_lw=0.75*, *component\_lw=0.75*, *\*\*component\_kwargs*)

Plot the fit. Must have fitted something before calling this!

It will be automatically called whenever a spectrum is fit (assuming an axis for plotting exists)

kwargs are passed to the fitter's components attribute

**plotresiduals** (*fig=2, axis=None, clear=True, \*\*kwargs*)

Plot residuals of the fit. Specify a figure or axis; defaults to figure(2).

kwargs are passed to matplotlib plot

**print\_fit** (*print\_baseline=True, \*\*kwargs*)

Print the best-fit parameters to the command line

**register\_fitter** (*\*args, \*\*kwargs*)

Register a model fitter

Register a fitter function.

Required Arguments:

**name:** [ **string** ] The fit function name.

**function:** [ **function** ] The fitter function. Single-fitters should take npars + 1 input parameters, where the +1 is for a 0th order baseline fit. They should accept an X-axis and data and standard fitting-function inputs (see, e.g., gaussfitter). Multi-fitters should take N \* npars, but should also operate on X-axis and data arguments.

**npars:** [ **int** ] How many parameters does the function being fit accept?

Optional Keyword Arguments:

**multisingle:** [ **'multi'** | **'single'** ] Is the function a single-function fitter (with a background), or does it allow N copies of the fitting function?

**override:** [ **True** | **False** ] Whether to override any existing type if already present.

**key:** [ **char** ] Key to select the fitter in interactive mode

**savefit** ()

Save the fit parameters from a Gaussian fit to the FITS header **\*THESE SHOULD BE WRITTEN FOR EACH TYPE OF MODEL TO BE FIT\***

**selectregion** (*xmin=None, xmax=None, xtype='wcs', highlight=False, fit\_plotted\_area=True, reset=False, verbose=False, debug=False, \*\*kwargs*)

Pick a fitting region in either WCS units or pixel units

**selectregion\_interactive** (*event, mark\_include=True, debug=False, \*\*kwargs*)

select regions for baseline fitting

**seterrspec** (*usestd=None, useresiduals=True*)

Simple wrapper function to set the error spectrum; will either use the input spectrum or determine the error using the RMS of the residuals, depending on whether the residuals exist.

**setfitspec** ()

Set the spectrum that will be fit. This is primarily to remove NaNs from consideration: if you simply remove the data from both the X-axis and the Y-axis, it will not be considered for the fit, and a linear X-axis is not needed for fitting.

However, it may be possible to do this using masked arrays instead of setting errors to be 1e10....

**shift\_pars** (*frame=None*)

Shift the velocity / wavelength / frequency of the fitted parameters into a different frame

Right now this only takes care of redshift and only if redshift is defined. It should be extended to do other things later

```
start_interactive (debug=False, LoudDebug=False, print_message=True,  
                   clear_all_connections=True, **kwargs)
```

## 2.4.3 Measurements

```
class pyspeckit.spectrum.measurements.Measurements (Spectrum, z=None, d=None,  
                                                    xunits=None, fluxnorm=None,  
                                                    miscline=None, mistol=10, ig-  
                                                    nore=None, derive=True, de-  
                                                    bug=False)
```

Bases: object

This can be called after a fit is run. It will inherit the specfit object and derive as much as it can from modelpars. Just do: spec.measure(z, xunits, fluxnorm)

Notes: If z (redshift) or d (distance) are present, we can compute integrated line luminosities rather than just fluxes. Provide distance in cm.

**Currently will only work with Gaussians. To generalize:** 1. make sure we manipulate modelpars correctly, i.e. read in entries corresponding to wavelength/frequency/whatever correctly.

**misclines = dictionary** miscline = {{ 'name': H\_alpha, 'wavelength': 6565, 'etc': 0}, {} }

**Measurements.bisection** (*f, x\_guess*)

Find root of function using bisection method. Absolute tolerance of 1e-4 is being used.

**Measurements.bracket\_root** (*f, x\_guess, atol=0.0001*)

Bracket root by finding points where function goes from positive to negative.

**Measurements.compute\_amplitude** (*pars*)

Calculate amplitude of emission line. Should be easy - add multiple components if they exist. Currently assumes multiple components have the same centroid.

**Measurements.compute\_flux** (*pars*)

Calculate integrated flux of emission line. Works for multi-component fits too. Unnormalized.

**Measurements.compute\_fwhm** (*pars*)

Determine full-width at half maximum for multi-component fit numerically, or analytically if line has only a single component. Uses bisection technique for the former with absolute tolerance of 1e-4.

**Measurements.compute\_luminosity** (*pars*)

Determine luminosity of line (need distance and flux units).

**Measurements.derive** ()

Calculate luminosity and FWHM for all spectral lines.

**Measurements.identify** ()

Determine identity of lines in self.modelpars. Fill entries of self.lines dictionary.

Note: This method will be infinitely slow for more than 10 or so lines.

**Measurements.separate** ()

For multicomponent lines, separate into broad and narrow components (assume only one of components is narrow).

**Measurements.to\_tex** ()

Write out fit results to tex format.

## 2.4.4 Units

### Units and SpectroscopicAxes

Unit parsing and conversion tool. The SpectroscopicAxis class is meant to deal with unit conversion internally

**Open Questions: Are there other FITS-valid projection types, unit types, etc.** that should be included? What about for other fields (e.g., wavenumber?)

**class** `pyspeckit.spectrum.units.SpectroscopicAxis`

Bases: `numpy.ndarray`

A Spectroscopic Axis object to store the current units of the spectrum and allow conversion to other units and frames. Typically, units are velocity, wavelength, frequency, or redshift. Wavenumber is also hypothetically possible.

**WARNING:** If you index a SpectroscopicAxis, the resulting array will be a SpectroscopicAxis without a `dxarr` attribute! This can result in major problems; a workaround is being sought but subclassing numpy arrays is harder than I thought

**as\_unit** (*unit*, *frame=None*, *quiet=True*, *center\_frequency=None*, *center\_frequency\_units=None*, *\*\*kwargs*)

Convert the spectrum to the specified units. This is a wrapper function to convert between frequency/velocity/wavelength and simply change the units of the X axis. Frame conversion is... not necessarily implemented.

**unit** [ *string* ] What unit do you want to ‘view’ the array as?

**frame** [ *None* ] NOT IMPLEMENTED. When it is, it will allow you to convert between LSR, topocentric, heliocentric, rest, redshifted, and whatever other frames we can come up with. Right now the main holdup is finding a nice python interface to an LSR velocity calculator... and motivation.

*center\_frequency* [ *None* | *float* ] *center\_frequency\_units* [ *None* | *string* ]

If converting between velocity and any other spectroscopic type, need to specify the central frequency around which that velocity is calculated. I think this can also accept wavelengths....

**cdelt** (*tolerance=1e-08*)

Return the channel spacing if channels are linear

**change\_frame** (*frame*)

Change velocity frame

**convert\_to\_unit** (*unit*, *\*\*kwargs*)

Return the X-array in the specified units without changing it Uses `as_unit` for the conversion, but changes internal values rather than returning them.

**coord\_to\_x** (*xval*, *xunit*)

Given an X-value assumed to be in the coordinate axes, return that value converted to xunit e.g.: `xarr.units = ‘km/s’ xarr.refX = 5.0 xarr.refX_units = GHz xarr.coord_to_x(6000, ‘GHz’) == 5.1 # GHz`

**in\_frame** (*frame*)

Return a shifted xaxis

**make\_dxarr** (*coordinate\_location=‘center’*)

Create a “delta-x” array corresponding to the X array.

**coordinate\_location** [ *‘left’*, *‘center’*, *‘right’* ] Does the coordinate mark the left, center, or right edge of the pixel? If ‘center’ or ‘left’, the *last* pixel will have the same dx as the second to last pixel. If right, the *first* pixel will have the same dx as the second pixel.

**umax** (*units=None*)

Return the maximum value of the SpectroscopicAxis. If units specified, convert to those units first

**umin** (*units=None*)

Return the minimum value of the SpectroscopicAxis. If units specified, convert to those units first

**x\_in\_frame** (*xx, frame*)

Return the value 'x' shifted to the target frame

**x\_to\_coord** (*xval, xunit, verbose=False*)

Given a wavelength/frequency/velocity, return the value in the SpectroscopicAxis's units e.g.: `xarr.units = 'km/s' xarr.refX = 5.0 xarr.refX_units = GHz xarr.x_to_coord(5.1, 'GHz') == 6000 # km/s`

**x\_to\_pix** (*xval*)

Given an X coordinate in SpectroscopicAxis' units, return the corresponding pixel number

**class** `pyspeckit.spectrum.units.SpectroscopicAxes`

Bases: `pyspeckit.spectrum.units.SpectroscopicAxis`

Counterpart to Spectra: takes a list of SpectroscopicAxis's and concatenates them while checking for consistency and maintaining header parameters

## 2.4.5 Registration

PySpecKit is made extensible by allowing user-registered modules for reading, writing, and fitting data.

For examples of registration in use, look at the source code of `pyspeckit.spectrum.__init__` and `pyspeckit.spectrum.fitters`.

The registration functions can be accessed directly:

```
pyspeckit.register_reader
pyspeckit.register_writer
```

However, models are bound to individual instances of the Spectrum class, so they must be accessed via a specfit instance

```
sp = pyspeckit.Spectrum('myfile.fits')
sp.specfit.register_fitter
```

Alternatively, you can access and edit the default Registry

```
pyspeckit.fitters.default_Registry.add_fitter
```

If you've already loaded a Spectrum instance, but then you want to reload fitters from the default\_Registry, or if you want to make your own *Registry*, you can use the semi-private method

```
MyRegistry = pyspeckit.fitters.Registry()
sp._register_fitters(registry=MyRegistry)
```

## API

`pyspeckit.spectrum.__init__.register_reader` (*filetype, function, suffix, default=False*)

Register a reader function.

Required Arguments:

**filetype:** [ **string** ] The file type name

**function:** [ **function** ] The reader function. Should take a filename as input and return an X-axis object (see units.py), a spectrum, an error spectrum (initialize it to 0's if empty), and a pyfits header instance

**suffix:** [ **int** ] What suffix should the file have?

Optional Keyword Arguments:

`pyspeckit.spectrum.__init__.register_writer` (*filetype, function, suffix, default=False*)  
Register a writer function.

Required Arguments:

**filetype:** [ **string** ] The file type name

**function:** [ **function** ] The writer function. Will be an attribute of Spectrum object, and called as `spectrum.Spectrum.write_hdf5()`, for example.

**suffix:** [ **int** ] What suffix should the file have?

Optional Keyword Arguments:

**class** `pyspeckit.spectrum.fitters.Registry`  
This class is a simple wrapper to prevent fitter properties from being globals

**add\_fitter** (*name, function, npars, multisingle='single', override=False, key=None*)  
Register a fitter function.

Required Arguments:

**name:** [ **string** ] The fit function name.

**function:** [ **function** ] The fitter function. Single-fitters should take `npars + 1` input parameters, where the +1 is for a 0th order baseline fit. They should accept an X-axis and data and standard fitting-function inputs (see, e.g., `gaussfitter`). Multi-fitters should take `N * npars`, but should also operate on X-axis and data arguments.

**npars:** [ **int** ] How many parameters does the function being fit accept?

Optional Keyword Arguments:

**multisingle:** [ **'multi'** | **'single'** ] Is the function a single-function fitter (with a background), or does it allow N copies of the fitting function?

**override:** [ **True** | **False** ] Whether to override any existing type if already present.

**key:** [ **char** ] Key to select the fitter in interactive mode

## 2.5 Readers

### 2.5.1 Plain Text

Text files should be of the form:

```
wavelength flux err
3637.390 0.314 0.000
3638.227 0.717 0.000
3639.065 1.482 0.000
```

where there 'err' column is optional but the others are not. The most basic spectrum file allowed would have no header and two columns, e.g.:

```
1 0.5
2 1.5
3 0.1
```

If the X-axis is not monotonic, the data will be sorted so that the X-axis is in ascending order.

## API

### PySpecKit ASCII Reader

Routines for reading in ASCII format spectra. If atpy is not installed, will use a very simple routine for reading in the data.

```
pyspeckit.spectrum.readers.txt_reader.open_1d_txt(filename, xaxcol=0, datacol=1, errorcol=2, text_reader='simple', atpytype='ascii', **kwargs)
```

Attempt to read a 1D spectrum from a text file assuming wavelength as the first column, data as the second, and (optionally) error as the third.

Reading can be done either with atpy or a 'simple' reader. If you have an IPAC, CDS, or formally formatted table, you'll want to use atpy.

If you have a simply formatted file of the form, e.g. # name name # unit unit data data data data

kwargs are passed to atpy.Table

```
pyspeckit.spectrum.readers.txt_reader.simple_txt(filename, xaxcol=0, datacol=1, errorcol=2, skiplines=0, **kwargs)
```

Very simple method for reading columns from ASCII file.

## 2.5.2 FITS

A minimal header should look like this:

```
SIMPLE = T / conforms to FITS standard
BITPIX = -32 / array data type
NAXIS = 2 / number of array dimensions
NAXIS1 = 659
NAXIS2 = 2
CRPIX1 = 1.0
CRVAL1 = -4953.029632560421
CDELT1 = 212.5358581542998
CTYPE1 = 'VRAD-LSR'
CUNIT1 = 'm/s'
BUNIT = 'K'
RESTFRQ = 110.20137E9
SPECSYS = 'LSRK'
END
```

A fits file with a header as above should be easily readable without any user effort:

```
sp = pyspeckit.Spectrum('test.fits')
```

If you have multiple spectroscopic axes, e.g.

```
CRPIX1A = 1.0
CRVAL1A = 110.2031747948101
CTYPE1A = 'FREQ-LSR'
```



```
CUNIT1A = 'GHz'
RESTFRQA= 110.20137
```

you can load that axis with the 'wctype' keyword:

```
sp = pyspeckit.Spectrum('test.fits', wctype='A')
```

If you have a .fits file with a non-linear X-axis that is stored in the .fits file as data (as opposed to being implicitly included in a header), you can load it using a custom .fits reader. An example implementation is given in the `tspec_reader`. It can be registered using [Registration](#):

```
tspec_reader = check_reader(tspec_reader.tspec_reader)
pyspeckit.register_reader('tspec', tspec_reader, 'fits')
```

## API

`pyspeckit.spectrum.readers.fits_reader.open_1d_fits(filename, **kwargs)`  
Grabs all the relevant pieces of a simple FITS-compliant 1d spectrum

### Inputs:

**wctype** - the suffix on the WCS type to get to velocity/frequency/whatever

**specnum** - Which # spectrum, along the y-axis, is the data?

**errspecnum** - Which # spectrum, along the y-axis, is the error spectrum?

```
pyspeckit.spectrum.readers.fits_reader.open_1d_pyfits(pyfits_hdu, spec-
                                                         num=0, wctype='',
                                                         specaxis='1', errspec-
                                                         num=None, autofix=True,
                                                         scale_keyword=None,
                                                         scale_action=<built-in
                                                         function div>, verbose=False,
                                                         **kwargs)
```

This is `open_1d_fits` but for a `pyfits_hdu` so you don't necessarily have to open a fits file

`pyspeckit.spectrum.readers.fits_reader.read_echelle(pyfits_hdu, hdr)`  
Read an IRAF Echelle spectrum

<http://iraf.noao.edu/iraf/ftp/iraf/docs/specwcs.ps.Z>

## 2.5.3 hdf5

(work in progress)

## API

### PySpecKit HDF5 Reader

Routines for reading in spectra from HDF5 files.

Note: Current no routines for parsing HDF5 headers in `classes.py`.

```
pyspeckit.spectrum.readers.hdf5_reader.open_hdf5(filename, xaxkey='xarr',
                                                    datakey='data', errkey='error')
```

This reader expects three datasets to exist in the hdf5 file 'filename': 'xarr', 'data', and 'error', by default. Can specify other dataset names.

## 2.5.4 Gildas CLASS files

Pyspeckit is capable of reading files from some versions of CLASS. The CLASS developers have stated that the GILDAS file format is private and will remain so, and therefore there are no guarantees that the CLASS reader will work for your file.

Nonetheless, if you want to develop in python instead of SIC, the `read_class` module is probably the best way to access CLASS data.

The [CLASS file specification](#) is incomplete, so much of the data reading is hacked together. The code style is based off of Tom Robitaille's [idlsave](#) package.

An example usage. Note that `telescope` and `line` are NOT optional keyword arguments, they are just specified as such for clarity

```
n2hp = class_to_obsblocks(fn1, telescope=['SMT-F1M-HU', 'SMT-F1M-VU'],
                        line=['N2HP(3-2)', 'N2H+(3-2)'])
```

This will generate a `ObsBlock` from all data tagged with the 'telescope' flags listed and lines matching either of those above. The data selection is equivalent to a combination of

```
find /telescope SMT-F1M-HU
find /telescope SMT-F1M-VU
find /line N2HP(3-2)
find /line N2H+(3-2)
```

ALL of the data matching those criteria will be included in an `ObsBlock`. They will then be accessible through the `ObsBlock`'s `spectlist` attribute, or just by indexing the `ObsBlock` directly.

## An essentially undocumented API

### GILDAS CLASS file reader

Read a CLASS file into an `pyspeckit.spectrum.ObsBlock`

```
pyspeckit.spectrum.readers.read_class.class_to_obsblocks(*arg, **kwargs)
    Load an entire CLASS observing session into a list of ObsBlocks based on matches to the 'telescope' and 'line' names
```

```
pyspeckit.spectrum.readers.read_class.class_to_spectra(*arg, **kwargs)
    Load each individual spectrum within a CLASS file into a list of Spectrum objects
```

```
pyspeckit.spectrum.readers.read_class.make_axis(header)
    Create a pyspeckit.spectrum.units.SpectroscopicAxis from the CLASS "header"
```

```
pyspeckit.spectrum.readers.read_class.read_class(*arg, **kwargs)
    A hacked-together method to read a binary CLASS file. It is strongly dependent on the incomplete GILDAS CLASS file type Specification
```

## 2.6 Wrappers

These are wrappers to simplify some of the more complicated (and even some of the simpler) functions in PySpecKit

### 2.6.1 Cube Fitting

```
pyspeckit.wrappers.cube_fit.cube_fit(cubefilename, outfilename, errfilename=None,
                                     scale_keyword=None, vheight=False, verbose=False,
                                     signal_cut=3, verbose_level=2, clobber=True,
                                     **kwargs)
```

Light-weight wrapper for cube fitting

Takes a cube and error map (error will be computed naively if not given) and computes moments then fits for each spectrum in the cube. It then saves the fitted parameters to a reasonably descriptive output file whose header will look like

```
PLANE1  = 'amplitude'
PLANE2  = 'velocity'
PLANE3  = 'sigma'
PLANE4  = 'err_amplitude'
PLANE5  = 'err_velocity'
PLANE6  = 'err_sigma'
PLANE7  = 'integral'
PLANE8  = 'integral_error'
CDELT3  = 1
CTYPE3  = 'FITPAR'
CRVAL3  = 0
CRPIX3  = 1
```

Parameters:

**errfilename** [ **None** | **string name of .fits file** ] A two-dimensional error map to use for computing signal-to-noise cuts

**scale\_keyword** [ **None** | **Char** ] Keyword to pass to the data cube loader - multiplies cube by the number indexed by this header kwarg if it exists. e.g., if your cube is in T\_A units and you want T\_A\*

**vheight** [ **bool** ] Is there a background to be fit? Used in moment computation

**verbose** [ **bool** ] **verbose\_level** [ **int** ]

How loud will the fitting procedure be? Passed to momenteach and fiteach

**signal\_cut** [ **float** ] Signal-to-Noise ratio minimum. Spectra with a peak below this S/N ratio will not be fit and will be left blank in the output fit parameter cube

**clobber** [ **bool** ] Overwrite parameter .fits cube if it exists?

**kwargs** are passed to `pyspeckit.Spectrum.specfit`

## 2.6.2 Simple Gaussian Fitter

```
pyspeckit.wrappers.fit_gaussians_to_simple_spectra.fit_gaussians_to_simple_spectra(filename,
units='km/
do-
plot=True,
base-
line=True,
plotresid-
u-
als=False,
fig-
ure-
save-
name=None,
cro-
prange=None,
save-
name=None,
**kwargs)
```

As stated in the name title, will fit Gaussians to simple spectra!

kwargs will be passed to specfit

**figuresave** [ **None** | **string** ] After fitting, save the figure to this filename if specified

**croprange** [ **list of 2 floats** ] Crop the spectrum to (min,max) in the specified units

**save** [ **None** | **string** ] After fitting, save the spectrum to this filename

Note that this wrapper can be used from the command line:

```
python fit_gaussians_to_simple_spectra.py spectrum.fits
```

## 2.6.3 NH3 fitter wrapper

Wrapper to fit ammonia spectra. Generates a reasonable guess at the position and velocity using a gaussian fit

```
pyspeckit.wrappers.fitnh3.fitnh3tkin(input_dict, dobaseline=True, baselinekwargs={},
crop=False, guessline='twotwo', tex=15, tkin=20, col-
umn=15.0, fortho=0.66000000000000003, tau=None,
thin=False, quiet=False, doplot=True, fignum=1,
guessfignum=2, smooth=False, scale_keyword=None,
rebase=False, npeaks=1, guesses=None, **kwargs)
```

Given a dictionary of filenames and lines, fit them together e.g. { 'oneone': 'G000.000+00.000\_nh3\_11.fits' }

```
pyspeckit.wrappers.fitnh3.plot_nh3(spdict, spectra, fignum=1, show_components=False, resid-
fignum=None, **plotkwargs)
```

Plot the results from a multi-nh3 fit

## 2.6.4 N2H+ fitter wrapper

Wrapper to fit N2H+ using RADEX models. This is meant to be used from the command line, e.g.:

```
python n2hp_wrapper.py file.fits
```

and therefore has no independently defined functions.

In place of the actual contents of N2H+ fitter, here are the modules used to make the wrapper

`model.SpectralModel()`

A wrapper class for a spectra model. Includes internal functions to generate multi-component models, annotations, integrals, and individual components. The declaration can be complex, since you should name individual variables, set limits on them, set the units the fit will be performed in, and set the annotations to be used. Check out some of the hyperfine codes (hcn, n2hp) for examples.

```
static n2hp.n2hp_radex(xarr, density=4, column=13, xoff_v=0.0, width=1.0, grid_vwidth=1.0,
                      grid_vwidth_scale=False, texgrid=None, taugrid=None, hdr=None,
                      path_to_texgrid='', path_to_taugrid='', temperature_gridnumber=3, de-
                      bug=False, verbose=False, **kwargs)
```

Use a grid of RADEX-computed models to make a model line spectrum

The RADEX models have to be available somewhere. OR they can be passed as arrays. If as arrays, the form should be: `texgrid = ((minfreq1,maxfreq1,texgrid1),(minfreq2,maxfreq2,texgrid2))`

`xarr` must be a `SpectroscopicAxis` instance `xoff_v`, `width` are both in km/s

**grid\_vwidth is the velocity assumed when computing the grid in km/s** this is important because  $\tau = \text{model}\tau / \text{width}$  (see, e.g., Draine 2011 textbook pgs 219-230)

`grid_vwidth_scale` is True or False: False for LVG, True for Sphere



# PYTHON MODULE INDEX

## p

- `pyspeckit.spectrum.__init__`, 18
- `pyspeckit.spectrum.baseline`, 12
- `pyspeckit.spectrum.fitters`, 19
- `pyspeckit.spectrum.measurements`, 16
- `pyspeckit.spectrum.models`, 6
  - `pyspeckit.spectrum.models.ammonia`, 7
  - `pyspeckit.spectrum.models.fitter`, 8
  - `pyspeckit.spectrum.models.formaldehyde`, 8
  - `pyspeckit.spectrum.models.hcn`, 9
  - `pyspeckit.spectrum.models.hill5infall`, 9
  - `pyspeckit.spectrum.models.hyperfine`, 9
  - `pyspeckit.spectrum.models.inherited_gaussfitter`, 8
  - `pyspeckit.spectrum.models.inherited_lorentzian`, 10
  - `pyspeckit.spectrum.models.inherited_voigtfitter`, 11
  - `pyspeckit.spectrum.models.model`, 6
  - `pyspeckit.spectrum.models.modelgrid`, 10
  - `pyspeckit.spectrum.models.n2hp`, 11
- `pyspeckit.spectrum.readers.fits_reader`, 21
- `pyspeckit.spectrum.readers.hdf5_reader`, 21
- `pyspeckit.spectrum.readers.read_class`, 22
- `pyspeckit.spectrum.readers.txt_reader`, 20
- `pyspeckit.spectrum.units`, 17
- `pyspeckit.wrappers`, 23
  - `pyspeckit.wrappers.cube_fit`, 23
  - `pyspeckit.wrappers.fit_gaussians_to_simple_spectra`, 23
  - `pyspeckit.wrappers.fitnh3`, 24
  - `pyspeckit.wrappers.n2hp_wrapper`, 24





# INDEX

## Symbols

`__call__()` (pyspeckit.spectrum.baseline.Baseline method), 12  
`__init__()` (pyspeckit.spectrum.baseline.Baseline method), 12  
`__module__` (pyspeckit.spectrum.baseline.Baseline attribute), 12

## A

`add_fitter()` (pyspeckit.spectrum.fitters.Registry method), 19  
`ammonia()` (in module pyspeckit.spectrum.models.ammonia), 7  
`annotate()` (pyspeckit.spectrum.baseline.Baseline method), 12  
`annotate()` (pyspeckit.spectrum.fitters.Specfit method), 13  
`annotations()` (pyspeckit.spectrum.models.model.SpectralModel method), 7  
`as_unit()` (pyspeckit.spectrum.units.SpectroscopicAxis method), 17  
`aval_dict` (in module pyspeckit.spectrum.models.hcn), 9  
`aval_dict` (in module pyspeckit.spectrum.models.n2hp), 11

## B

`Baseline` (class in pyspeckit.spectrum.baseline), 12  
`button2action()` (pyspeckit.spectrum.baseline.Baseline method), 12  
`button3action()` (pyspeckit.spectrum.baseline.Baseline method), 12  
`button3action()` (pyspeckit.spectrum.fitters.Specfit method), 13

## C

`cdelt()` (pyspeckit.spectrum.units.SpectroscopicAxis method), 17  
`change_frame()` (pyspeckit.spectrum.units.SpectroscopicAxis method), 17  
`class_to_obsblocks()` (in module pyspeckit.spectrum.readers.read\_class), 22

`class_to_spectra()` (in module pyspeckit.spectrum.readers.read\_class), 22  
`clear()` (pyspeckit.spectrum.fitters.Specfit method), 13  
`clear_all_connections()` (pyspeckit.spectrum.fitters.Specfit method), 13  
`clear_highlights()` (pyspeckit.spectrum.fitters.Specfit method), 13  
`clearlegend()` (pyspeckit.spectrum.baseline.Baseline method), 13  
`components()` (pyspeckit.spectrum.models.model.SpectralModel method), 7  
`convert_to_unit()` (pyspeckit.spectrum.units.SpectroscopicAxis method), 17  
`coord_to_x()` (pyspeckit.spectrum.units.SpectroscopicAxis method), 17  
`crop()` (pyspeckit.spectrum.baseline.Baseline method), 13  
`crop()` (pyspeckit.spectrum.fitters.Specfit method), 13  
`cube_fit()` (in module pyspeckit.wrappers.cube\_fit), 23

## D

`downsample()` (pyspeckit.spectrum.baseline.Baseline method), 13  
`downsample()` (pyspeckit.spectrum.fitters.Specfit method), 13

## E

`EQW()` (pyspeckit.spectrum.fitters.Specfit method), 13  
`event_manager()` (pyspeckit.spectrum.fitters.Specfit method), 13

## F

`firstclick_guess()` (pyspeckit.spectrum.fitters.Specfit method), 13  
`firstclick_selection()` (pyspeckit.spectrum.fitters.Specfit method), 13  
`fit_gaussians_to_simple_spectra()` (in module pyspeckit.wrappers.fit\_gaussians\_to\_simple\_spectra), 24  
`fitnh3tkin()` (in module pyspeckit.wrappers.fitnh3), 24  
`fitter()` (pyspeckit.spectrum.models.model.SpectralModel method), 7

formaldehyde() (in module  
pyspeckit.spectrum.models.formaldehyde),  
8

formaldehyde\_radex() (in module  
pyspeckit.spectrum.models.formaldehyde),  
8

fullsizemodel() (pyspeckit.spectrum.fitters.Specfit  
method), 13

## G

gaussian() (in module  
pyspeckit.spectrum.models.inherited\_gaussfitter),  
8

gaussian\_fitter() (in module  
pyspeckit.spectrum.models.inherited\_gaussfitter),  
9

gaussian\_line() (in module  
pyspeckit.spectrum.models.modelgrid), 10

guesspeakwidth() (pyspeckit.spectrum.fitters.Specfit  
method), 14

## H

hcn\_radex() (in module pyspeckit.spectrum.models.hcn),  
9

highlight\_fitregion() (pyspeckit.spectrum.fitters.Specfit  
method), 14

hill5\_model() (in module  
pyspeckit.spectrum.models.hill5infall), 9

hyperfine() (pyspeckit.spectrum.models.hyperfine.hyperfine  
method), 10

hyperfine\_amp() (pyspeckit.spectrum.models.hyperfine.hyperfine  
method), 10

hyperfinemodel (class in  
pyspeckit.spectrum.models.hyperfine), 9

## I

in\_frame() (pyspeckit.spectrum.units.SpectroscopicAxis  
method), 17

integral() (pyspeckit.spectrum.fitters.Specfit method), 14

integral() (pyspeckit.spectrum.models.model.SpectralModel  
method), 7

## J

jfunc() (in module pyspeckit.spectrum.models.hill5infall),  
9

## L

line\_model\_2par() (in module  
pyspeckit.spectrum.models.modelgrid), 10

line\_params\_2D() (in module  
pyspeckit.spectrum.models.modelgrid), 10

lorentzian() (in module  
pyspeckit.spectrum.models.inherited\_lorentzian),  
10

lorentzian\_fitter() (in module  
pyspeckit.spectrum.models.inherited\_lorentzian),  
10

## M

make\_axis() (in module  
pyspeckit.spectrum.readers.read\_class), 22

make\_dxarr() (pyspeckit.spectrum.units.SpectroscopicAxis  
method), 17

moments() (pyspeckit.spectrum.fitters.Specfit method),  
14

mpfitfun() (pyspeckit.spectrum.models.model.SpectralModel  
method), 7

multifit() (pyspeckit.spectrum.fitters.Specfit method), 14

## N

n2hp\_radex() (in module  
pyspeckit.spectrum.models.n2hp), 11

n2hp\_radex() (pyspeckit.spectrum.models.n2hp static  
method), 25

n\_modelfunc() (pyspeckit.spectrum.models.model.SpectralModel  
method), 7

## O

open\_1d\_fits() (in module  
pyspeckit.spectrum.readers.fits\_reader), 21

open\_1d\_pyfits() (in module  
pyspeckit.spectrum.readers.fits\_reader), 21

open\_1d\_txt() (in module  
pyspeckit.spectrum.readers.txt\_reader), 20

open\_hdf5() (in module  
pyspeckit.spectrum.readers.hdf5\_reader),  
21

## P

peakbgfit() (pyspeckit.spectrum.fitters.Specfit method),  
14

plot\_baseline() (pyspeckit.spectrum.baseline.Baseline  
method), 13

plot\_fit() (pyspeckit.spectrum.fitters.Specfit method), 14

plot\_nh3() (in module pyspeckit.wrappers.fithn3), 24

plotresiduals() (pyspeckit.spectrum.fitters.Specfit  
method), 15

print\_fit() (pyspeckit.spectrum.fitters.Specfit method), 15

pyspeckit.spectrum.\_\_init\_\_ (module), 18

pyspeckit.spectrum.baseline (module), 12

pyspeckit.spectrum.fitters (module), 13, 19

pyspeckit.spectrum.measurements (module), 16

pyspeckit.spectrum.models (module), 6

pyspeckit.spectrum.models.ammonia (module), 7

pyspeckit.spectrum.models.fitter (module), 8

pyspeckit.spectrum.models.formaldehyde (module), 8

pyspeckit.spectrum.models.hcn (module), 9

pyspeckit.spectrum.models.hill5infall (module), 9

[pyspeckit.spectrum.models.hyperfine \(module\)](#), 9  
[pyspeckit.spectrum.models.inherited\\_gaussfitter \(module\)](#), 8  
[pyspeckit.spectrum.models.inherited\\_lorentzian \(module\)](#), 10  
[pyspeckit.spectrum.models.inherited\\_voigtfitter \(module\)](#), 11  
[pyspeckit.spectrum.models.model \(module\)](#), 6  
[pyspeckit.spectrum.models.modelgrid \(module\)](#), 10  
[pyspeckit.spectrum.models.n2hp \(module\)](#), 11  
[pyspeckit.spectrum.readers.fits\\_reader \(module\)](#), 21  
[pyspeckit.spectrum.readers.hdf5\\_reader \(module\)](#), 21  
[pyspeckit.spectrum.readers.read\\_class \(module\)](#), 22  
[pyspeckit.spectrum.readers.txt\\_reader \(module\)](#), 20  
[pyspeckit.spectrum.units \(module\)](#), 17  
[pyspeckit.wrappers \(module\)](#), 23  
[pyspeckit.wrappers.cube\\_fit \(module\)](#), 23  
[pyspeckit.wrappers.fit\\_gaussians\\_to\\_simple\\_spectra \(module\)](#), 23  
[pyspeckit.wrappers.fitnh3 \(module\)](#), 24  
[pyspeckit.wrappers.n2hp\\_wrapper \(module\)](#), 24

## R

[read\\_class\(\) \(in module pyspeckit.spectrum.readers.read\\_class\)](#), 22  
[read\\_echelle\(\) \(in module pyspeckit.spectrum.readers.fits\\_reader\)](#), 21  
[register\\_fitter\(\) \(pyspeckit.spectrum.fitters.Specfit method\)](#), 15  
[register\\_reader\(\) \(in module pyspeckit.spectrum.\\_\\_init\\_\\_\)](#), 18  
[register\\_writer\(\) \(in module pyspeckit.spectrum.\\_\\_init\\_\\_\)](#), 19  
[Registry \(class in pyspeckit.spectrum.fitters\)](#), 19

## S

[savefit\(\) \(pyspeckit.spectrum.baseline.Baseline method\)](#), 13  
[savefit\(\) \(pyspeckit.spectrum.fitters.Specfit method\)](#), 15  
[selectregion\(\) \(pyspeckit.spectrum.fitters.Specfit method\)](#), 15  
[selectregion\\_interactive\(\) \(pyspeckit.spectrum.fitters.Specfit method\)](#), 15  
[seterrspec\(\) \(pyspeckit.spectrum.fitters.Specfit method\)](#), 15  
[setfitspec\(\) \(pyspeckit.spectrum.fitters.Specfit method\)](#), 15  
[shift\\_pars\(\) \(pyspeckit.spectrum.fitters.Specfit method\)](#), 15  
[simple\\_txt\(\) \(in module pyspeckit.spectrum.readers.txt\\_reader\)](#), 20  
[slope\(\) \(pyspeckit.spectrum.models.model.SpectralModel method\)](#), 7  
[Specfit \(class in pyspeckit.spectrum.fitters\)](#), 13

[SpectralModel \(class in pyspeckit.spectrum.models.model\)](#), 6  
[SpectralModel\(\) \(pyspeckit.spectrum.models.model method\)](#), 25  
[SpectroscopicAxes \(class in pyspeckit.spectrum.units\)](#), 18  
[SpectroscopicAxis \(class in pyspeckit.spectrum.units\)](#), 17  
[start\\_interactive\(\) \(pyspeckit.spectrum.fitters.Specfit method\)](#), 15

## U

[umax\(\) \(pyspeckit.spectrum.units.SpectroscopicAxis method\)](#), 17  
[umin\(\) \(pyspeckit.spectrum.units.SpectroscopicAxis method\)](#), 18  
[unsubtract\(\) \(pyspeckit.spectrum.baseline.Baseline method\)](#), 13

## V

[voigt\(\) \(in module pyspeckit.spectrum.models.inherited\\_voigtfitter\)](#), 11  
[voigt\\_fitter\(\) \(in module pyspeckit.spectrum.models.inherited\\_voigtfitter\)](#), 11

## X

[x\\_in\\_frame\(\) \(pyspeckit.spectrum.units.SpectroscopicAxis method\)](#), 18  
[x\\_to\\_coord\(\) \(pyspeckit.spectrum.units.SpectroscopicAxis method\)](#), 18  
[x\\_to\\_pix\(\) \(pyspeckit.spectrum.units.SpectroscopicAxis method\)](#), 18