pyspeckit Documentation

Release 0.1

Adam Ginsburg and Jordan Mirocha

December 22, 2011

CONTENTS

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*

CHAPTER

ONE

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.

CHAPTER

TWO

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:

```
hill5_fitter = model.SpectralModel(hill5_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' )
```

2.2.1 Generic SpectralModel wrapper

class pyspeckit.spectrum.models.model.**SpectralModel**(*modelfunc*, *npars*, *parnames=None*, *parvalues=None*, *parlimits=None*, *parlimited=None*, *parfixed=None*, *parerror=None*, *partied=None*, *fitunits=None*, *parsteps=None*, *npeaks=1*, *shortvarnames=('A'*, *'v'*, *'\sigma')*, ***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 paremeter (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

annotations(*shortvarnames=None*)

Return a list of TeX-formatted labels

components(*xarr*, *pars*)

Return a numpy ndarray of the independent components of the fits

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

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=100000000000000.0*, *width=1*, *xoff_v=0.0*, *fortho=0.0*, *tau=None*, *fillingfraction=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 parametetrization 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_11-1_10 / 2_12-2_11 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*, *return_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*, *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 = modeltau / 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

Created 3/17/08

Original version available at http://code.google.com/p/agpy/source/browse/trunk/agpy/gaussfitter.py (the version below uses a Class instead of independent functions) 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**

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*, *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 = modeltau / width (see, e.g., Draine 2011 textbook pgs 219-230)

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

2.2.6 Hill5 analytic infall model

Code translated from: 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?

2.2.7 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 indvidual 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=0.1*, *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=0.1*)

wrapper of self.hyperfine with order of arguments changed

2.2.8 Lorentzian Fitter

2.2.9 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 0x1062b91b8>*)

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

2.2.10 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**

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*, *texgrid=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 = modeltau / width (see, e.g., Draine 2011 textbook pgs 219-230)

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

2.3 Classes

2.3.1 spectrum

class pyspeckit.spectrum.**Spectrum**(*filename=None*, *filetype=None*, *xarr=None*, *data=None*, *error=None*, *header=None*, *doplot=False*, *maskdata=True*, *plotkwargs={}*, *xarrkwargs={}*, ***kwargs*)

Bases: object

The core class for the spectroscopic toolkit. Contains the data and error arrays along with wavelength / frequency / velocity information in various formats.

Contains functions / classes to:

-read and write FITS-compliant spectroscopic data sets -read fits binaries? (not implemented)

-plot a spectrum -fit a spectrum both interactively and non-interactively

-with gaussian, voigt, lorentzian, and multiple (gvl) profiles

-fit a continuum "baseline" to a selected region with an n-order polynomial

-perform fourier transforms and operations in fourier space on a spectrum (not implemented)

Example usage:

automatically load a "compliant" (linear X-axis) FITS file

spec = pyspeckit.Spectrum('test.fits')

plot the spectrum

spec.plotter()

__init__ Initialize the Spectrum. Accepts files in the following formats:

- .fits
- .txt
- .hdf5

Must either pass in a filename or ALL of xarr, data, and header, plus optionally error.

doplot - if specified, will generate a plot when created

kwargs are passed to the reader, not the plotter

copy()
Create a copy of the spectrum with its own plotter, fitter, etc. Useful for, e.g., comparing smoothed to unsmoothed data

crop(*x1*, *x2*, *units=None*, ***kwargs*)
Replace the current spectrum with a subset from x1 to x2 in current units

Fixes CRPIX1 and baseline and model spectra to match cropped data spectrum

getlines(*linetype='radio'*, ***kwargs*)
Access a registered database of spectral lines. Will add an attribute with the name linetype, which then has properties defined by the speclines module (most likely, a table and a "show" function to display the lines)

interpnans(*spec*)
Interpolate over NAN values, replacing them with their neighbors...

measure(*z=None*, *d=None*, *fluxnorm=None*, *miscline=None*, *misctol=None*, *ignore=None*, *derive=True*)
Initialize the measurements class - only do this after you have run a fitter otherwise pyspeckit will be angry!

moments(*unit='km/s'*, ***kwargs*)

Return the moments of the spectrum. In order to assure that the 1st and 2nd moments are meaningful, a 'default' unit is set. If unit is not set, will use current unit.

Returns (height, amplitude, x, width_x)

the gaussian parameters of a 1D distribution by calculating its moments. Depending on the input parameters, will only output a subset of the above. "height" is the background level "amplitude" is the maximum (or minimum) of the data after background subtraction "x" is the first moment "width_x" is the second moment

If using masked arrays, pass estimator=numpy.ma.median 'estimator' is used to measure the background level (height)

negamp can be used to force the peak negative (True), positive (False), or it will be "autodetected" (negamp=None)

parse_hdf5_header(*hdr*)
HDF5 reader will create a hdr dictionary from HDF5 dataset attributes if they exist. This routine will convert that dict to a pyfits header instance.

parse_header(*hdr*, *specname=None*)
Parse parameters from a .fits header into required spectrum structure parameters

This should be moved to the FITSSpectrum subclass when that is available

parse_text_header(*Table*)
Grab relevant parameters from a table header (xaxis type, etc)

This function should only exist for Spectrum objects created from .txt or other atpy table type objects

shape()
Return the data shape

slice(*x1*, *x2*, *units=None*, *copy=True*)
Slice a spectrum. Defaults to the current units. Can specify 'pixels' if you want to just slice by index

smooth(*smooth*, ***kwargs*)

Smooth the spectrum by factor "smooth". sm.smooth doc:

Smooth and downsample the data array smoothtype - 'gaussian','hanning', or 'boxcar' downsample - Downsample the data? downsample_factor - Downsample by the smoothing factor, or something else? convmode - see numpy.convolve. 'same' returns an array of the same length as

'data' (assuming data is larger than the kernel)

stats(*statrange=()*, *interactive=False*)
Return some statistical measures in a dictionary (somewhat self-explanatory)

range - X-range over which to perform measures interactive - specify range interactively in plotter

write(*filename*, *type=None*, ***kwargs*)
Write the spectrum to a file. The available file types are listed in spectrum.writers.writers

type - what type of file to write to? If not specified, will attempt to determine type from suffix

class pyspeckit.spectrum.**Spectra**(*speclist*, *xunits='GHz'*, ***kwargs*)
Bases: pyspeckit.spectrum.classes.Spectrum

A list of individual Spectrum objects. Intended to be used for concatenating different wavelength observations of the SAME OBJECT. Can be operated on just like any Spectrum object, incuding fitting. Useful for fitting multiple lines on non-continguous axes simultaneously. Be wary of plotting these though...

Can be indexed like python lists.

X array is forcibly sorted in increasing order

fiteach(***kwargs*)
Fit each spectrum within the Spectra object

ploteach(*xunits=None*, *inherit_fit=False*, *plot_fit=True*, *plotfitkwargs={}*, ***plotkwargs*)
Plot each spectrum in its own window inherit_fit - if specified, will grab the fitter & fitter properties from Spectra

smooth(*smooth*, ***kwargs*)
Smooth the spectrum by factor "smooth". Options are defined in sm.smooth

because 'Spectra' does not have a header attribute, don't do anything to it...

class pyspeckit.spectrum.**ObsBlock**(*speclist*, *xtype='frequency'*, *xarr=None*, *force=False*, ***kwargs*)
Bases: pyspeckit.spectrum.classes.Spectra

An Observation Block

Consists of multiple spectra with a shared X-axis. Intended to hold groups of observations of the same object in the same setup for later averaging.

ObsBlocks can be indexed like python lists.

average (*weight=None*, *inverse_weight=False*, *error='erravgrtn'*)
Average all scans in an ObsBlock. Returns a single Spectrum object

weight - a header keyword to weight by error - estimate the error spectrum. Can be:

'scanrms' - the standard deviation of each pixel across all scans 'erravg' - the average of all input error spectra 'erravgrtn' - the average of all input error spectra divided by sqrt(n_obs)

smooth (*smooth*, ***kwargs*)
Smooth the spectrum by factor "smooth". Options are defined in sm.smooth

pyspeckit.spectrum.**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.**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:

2.3.2 Cubes

Cubes

Tools to deal with spectroscopic data cubes.

Many features in Cubes require additional packages: smoothing - requires agpy's smooth and parallel_map routines coords pyregion pywcs

The 'grunt work' is performed by the `cubes` module

class pyspeckit.cubes.SpectralCube.**Cube** (*filename*, *x0=0*, *y0=0*, ***kwargs*)
Bases: pyspeckit.spectrum.classes.Spectrum

Initialize the Cube. Accepts files in the following formats:

- .fits

x0,y0 - initial spectrum to use (defaults to lower-left corner)

copy()
Create a copy of the spectrum with its own plotter, fitter, etc. Useful for, e.g., comparing smoothed to unsmoothed data

crop(*x1*, *x2*, *units=None*, ***kwargs*)
Replace the current spectrum with a subset from x1 to x2 in current units

Fixes CRPIX1 and baseline and model spectra to match cropped data spectrum

fiteach(*errspec=None*, *errmap=None*, *guesses=()*, *verbose=True*, *verbose_level=1*, *quiet=True*, *signal_cut=3*, *usemomentcube=False*, *blank_value=0*, *integral=True*, *direct=False*, *absorption=False*, ***fitkwargs*)
Fit a spectrum to each valid pixel in the cube

get_apspec(*aperture*, *coordsys=None*)
Extract an aperture using cubes.extract_aperture (defaults to Cube coordinates)

get_spectrum(*x*, *y*)
Very simple: get the spectrum at coordinates x,y

Returns a SpectroscopicAxis instance

getlines(*linetype='radio'*, ***kwargs*)
Access a registered database of spectral lines. Will add an attribute with the name linetype, which then has properties defined by the speclines module (most likely, a table and a "show" function to display the lines)

interpnans(*spec*)
Interpolate over NAN values, replacing them with their neighbors...

measure(*z=None*, *d=None*, *fluxnorm=None*, *miscline=None*, *misctol=None*, *ignore=None*, *derive=True*)
Initialize the measurements class - only do this after you have run a fitter otherwise pyspeckit will be angry!

momenteach(*verbose=True*, *verbose_level=1*, ***kwargs*)
Return a cube of the moments of each pixel

moments(*unit='km/s'*, ***kwargs*)

Return the moments of the spectrum. In order to assure that the 1st and 2nd moments are meaningful, a 'default' unit is set. If unit is not set, will use current unit.

Returns (height, amplitude, x, width_x)

the gaussian parameters of a 1D distribution by calculating its moments. Depending on the input parameters, will only output a subset of the above. "height" is the background level "amplitude" is the maximum (or minimum) of the data after background subtraction "x" is the first moment "width_x" is the second moment

If using masked arrays, pass estimator=numpy.ma.median 'estimator' is used to measure the background level (height)

negamp can be used to force the peak negative (True), positive (False), or it will be "autodetected" (negamp=None)

parse_hdf5_header(*hdr*)
HDF5 reader will create a hdr dictionary from HDF5 dataset attributes if they exist. This routine will convert that dict to a pyfits header instance.

parse_header(*hdr*, *specname=None*)
Parse parameters from a .fits header into required spectrum structure parameters

This should be moved to the FITSSpectrum subclass when that is available

parse_text_header(*Table*)
Grab relevant parameters from a table header (xaxis type, etc)

This function should only exist for Spectrum objects created from .txt or other atpy table type objects

plot_apspec(*aperture*, *coordsys=None*, *reset_ylimits=True*, ***kwargs*)
Extract an aperture using cubes.extract_aperture (defaults to Cube coordinates)

plot_spectrum(*x*, *y*, ***kwargs*)
Fill the .data array with a real spectrum and plot it

set_apspec(*aperture*, *coordsys=None*)
Extract an aperture using cubes.extract_aperture (defaults to Cube coordinates)

shape()
Return the data shape

slice(*x1*, *x2*, *units=None*, *copy=True*)
Slice a spectrum. Defaults to the current units. Can specify 'pixels' if you want to just slice by index

smooth(*smooth*, ***kwargs*)

Smooth the spectrum by factor "smooth". sm.smooth doc:

Smooth and downsample the data array smoothtype - 'gaussian','hanning', or 'boxcar' downsample - Downsample the data? downsample_factor - Downsample by the smoothing factor, or something else? convmode - see numpy.convolve. 'same' returns an array of the same length as

'data' (assuming data is larger than the kernel)

stats(*statrange=()*, *interactive=False*)
Return some statistical measures in a dictionary (somewhat self-explanatory)

range - X-range over which to perform measures interactive - specify range interactively in plotter

write(*filename*, *type=None*, ***kwargs*)
Write the spectrum to a file. The available file types are listed in spectrum.writers.writers

type - what type of file to write to? If not specified, will attempt to determine type from suffix

class pyspeckit.cubes.SpectralCube.**CubeStack**(*cubelist*, *xunits='GHz'*, *x0=0*, *y0=0*, ***kwargs*)
Bases: pyspeckit.cubes.SpectralCube.Cube

The Cube equivalent of Spectra: for stitching multiple cubes with the same spatial grid but different frequencies together

Initialize the Cube. Accepts files in the following formats:

- .fits

x0,y0 - initial spectrum to use (defaults to lower-left corner)

cubes.py

From *agpy* <*http://code.google.com/p/agpy/source/browse/trunk/agpy/cubes.py*>, contains functions to perform various transformations on data cubes and their headers.

pyspeckit.cubes.cubes.**aper_world2pix**(*ap*, *wcs*, *coordsys='galactic'*, *wunit='arcsec'*)
Converts an elliptical aperture (x,y,width,height,PA) from WCS to pixel coordinates given an input wcs (an instance of the pywcs.WCS class). Must be a 2D WCS header.

pyspeckit.cubes.cubes.**coords_in_image**(*fitsfile*, *lon*, *lat*, *system='galactic'*)
Determine whether the coordinates are inside the image

pyspeckit.cubes.cubes.**extract_aperture**(*cube*, *ap*, *r_mask=False*, *wcs=None*, *coordsys='galactic'*, *wunit='arcsec'*, *debug=False*)
Extract an aperture from a data cube. E.g. to acquire a spectrum of an outflow that is extended.

Cube should have shape [z,y,x], e.g. cube = pyfits.getdata('datacube.fits')

Apertures are specified in PIXEL units with an origin of 0,0 (NOT the 1,1 fits standard!) unless wcs and coordsys are specified

INPUTS: wcs - a pywcs.WCS instance associated with the data cube coordsys - the coordinate system the aperture is specified in.

Options are 'celestial' and 'galactic'. Default is 'galactic'

wunit - units of width/height. default 'arcsec', options 'arcmin' and 'degree'

For a circular aperture, len(ap)=3: ap = [xcen,ycen,radius]

For an elliptical aperture, len(ap)=5: ap = [xcen,ycen,height,width,PA]

Optional inputs: r_mask - return mask in addition to spectrum (for error checking?)

pyspeckit.cubes.cubes.**flatten_header**(*header*)
Attempt to turn an N-dimensional fits header into a 2-dimensional header Turns all CRPIX[>2] etc. into new keywords with suffix 'A'

header must be a pyfits.Header instance

pyspeckit.cubes.cubes.**getspec**(*lon*, *lat*, *rad*, *cube*, *header*, *r_fits=True*, *inherit=True*, *wunit='arcsec'*)
Given a longitude, latitude, aperture radius (arcsec), and a cube file, return a .fits file or a spectrum.

lon,lat - longitude and latitude center of a circular aperture in WCS coordinates must be in coordinate system of the file

rad - radius (default degrees) of aperture

pyspeckit.cubes.cubes.**getspec_reg**(*cubefilename*, *region*, ***kwargs*)
Aperture extraction from a cube using a pyregion circle region

The region must be in the same coordinate system as the cube header

pyspeckit.cubes.cubes.**integ**(*file*, *vrange*, *xcen=None*, *xwidth=None*, *ycen=None*, *ywidth=None*, ***kwargs*)
wrapper of subimage_integ that defaults to using the full image

pyspeckit.cubes.cubes.**plane_smooth**(*cube*, *cubedim=0*, *parallel=True*, *numcores=None*, ***kwargs*)
parallel-map the smooth function

parallel - defaults True. Set to false if you want serial (for debug purposes?)

numcores - pass to parallel_map (None = use all available)

pyspeckit.cubes.cubes.**speccen_header**(*header*, *lon=None*, *lat=None*)
Turn a cube header into a spectrum header, retaining RA/Dec vals where possible (speccen is like flatten; specify would be better but, specify? nah)

Assumes 3rd axis is velocity

pyspeckit.cubes.cubes.**spectral_smooth**(*cube*, *smooth_factor*)
Smooth the cube along the spectral direction

pyspeckit.cubes.cubes.**subcube**(*cube*, *xcen*, *xwidth*, *ycen*, *ywidth*, *header=None*, *dvmult=False*, *return_HDU=False*, *units='pixels'*, *widthunits='pixels'*)

Crops a data cube

All units assumed to be pixel units

cube has dimensions (velocity, y, x)

xwidth and ywidth are "radius" values, i.e. half the length that will be extracted

if dvmult is set, multiple the average by DV (this is useful if you set average=sum and dvmul=True to get an integrated value)

pyspeckit.cubes.cubes.**subimage_integ**(*cube*, *xcen*, *xwidth*, *ycen*, *ywidth*, *vrange*, *header=None*, *average=<function mean at 0x102f14a28>*, *dvmult=False*, *return_HDU=False*, *units='pixels'*, *zunits=None*)

Returns a sub-image from a data cube integrated over the specified velocity range

All units assumed to be pixel units

cube has dimensions (velocity, y, x)

xwidth and ywidth are "radius" values, i.e. half the length that will be extracted

if dvmult is set, multiple the average by DV (this is useful if you set average=sum and dvmul=True to get an integrated value)

2.4 Features

2.4.1 Baseline Fitting

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.

annotate(*loc='upper left'*)

clearlegend()

crop(*x1pix*, *x2pix*)

When spectrum.crop is called, this must be too

dofit(*exclude=None*, *excludeunits='velo'*, *annotate=False*, *include=None*, *includeunits='velo'*, *LoudDebug=False*, *subtract=True*, *fit_original=False*, *powerlaw=False*, *xarr_fit_units='pixels'*, *baseline_fit_color='orange'*, ***kwargs*)

Do the baseline fitting and save and plot the results.

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

downsample(*factor*)

plot_baseline(*annotate=True*, *baseline_fit_color='orange'*)
Overplot the baseline fit

savefit()

selectregion(*xmin=None*, *xmax=None*, *xtype='wcs'*, *highlight=False*, ***kwargs*)
Pick a fitting region in either WCS units or pixel units

selectregion_interactive(*event*, *debug=False*, ***kwargs*)
select regions for baseline fitting

show_fitregion()

unsubtract()

2.4.2 Model Fitting

class pyspeckit.spectrum.fitters.**Specfit**(*Spectrum*, *Registry=None*)
Bases: object

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=0.1*, *handlelength=0.1*, *handletextpad=0.1*, *frameon=False*, ***kwargs*)
Add a legend to the plot showing the fitted parameters

clearlegend() will remove the legend

kwargs passed to legend

clear(*legend=True*, *components=True*)
Remove the fitted model from the plot

Also removes the legend by default

crop(*x1pix*, *x2pix*)
When spectrum.crop is called, this must be too

downsample(*factor*)
Downsample the model spectrum (and the spectofit spectra) This should only be done when Spectrum.smooth is called

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

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

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

For window-interactive use only! (i.e., you probably shouldn't call this from the command line or a script)

Given a set of clicks or button presses, sets the fit guesses

makeguess_debug(*event*)

multifit(*fittype=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)

fittype - What function will be fit? fittype 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*, *fittype=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.

fittype - What function will be fit? fittype 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

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'*, *reset=False*, *debug=False*, *verbose=True*, ***kwargs*)
Pick a fitting region in either WCS units or pixel units

reset - if true, overrides input xmin,xmax and selects the full range

selectregion_interactive(*event*, *debug=False*)
For window-interactive use only! (i.e., you probably shouldn't call this from the command line or a script)

Defines the fitting region of the spectrum

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....

2.4.3 Measurements

class pyspeckit.spectrum.measurements.**Measurements**(*Spectrum*, *z=None*, *d=None*, *xunits=None*, *fluxnorm=None*, *miscline=None*, *misctol=10*, *ignore=None*, *derive=True*)
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.fitpars. 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.

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

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

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_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*, ***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.

kwargs are passed to atpy.Table

pyspeckit.spectrum.readers.txt_reader.**simple_txt**(*filename*, *xaxcol=0*, *datacol=1*, *errorcol=2*, ***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 'wcstype' keyword:

```
sp = pyspeckit.Spectrum('test.fits',wcstype='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 heaer), 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:

wcstype - 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*, *specnum=0*, *wcstype=''*, *specaxis='1'*, *errspecnum=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

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 *speclist* 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=Non* *cro-* *prange=No* *save-* *name=Non* ***kwargs*)

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

kwargs will be passed to specfit

***figuresavename* [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

***savename* [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*, *column=15.0*, *fortho=0.66*, *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*, *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 = modeltau / 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

INDEX

R

S

U

W

X