
spectacle Documentation

Release 0.4.dev204

Nicholas Earl, Molly Peeples

Dec 13, 2019

CONTENTS

I Quick example	3
II Using Spectacle	7
1 Installation	9
2 Contributing	11
3 Getting Started	15
4 Modeling	17
5 Line Finding	23
6 Fitting	29
7 Registries	33
8 Analysis	35
III API	39
9 API	41
IV Indices and tables	55
Python Module Index	59
Index	61



Spectacle is an automated model generator for producing models that represent spectral data. It features the ability to reduce spectral data to its absorption components, fit features and continua, as well as allow for statistical analysis of spectral regions.

This package can also be used to generate analytical spectra from detailed characteristics, find potential line features, and simultaneously fit sets of absorption/emission lines.

Part I

Quick example

Include some setup imports for plotting and unit support.

```
>>> from astropy import units as u
>>> import numpy as np
>>> from matplotlib import pyplot as plt
>>> from astropy.visualization import quantity_support
>>> quantity_support() # for getting units on the axes below
```

Import the spectral model and profile model.

```
>>> from spectacle.modeling import Spectral1D, OpticalDepth1D
```

Create some HI lines to add to the spectral model. **Note** that all column densities are given in $\log 1/\text{cm}^2$.

```
>>> line = OpticalDepth1D("HI1216", v_doppler=50 * u.km/u.s, column_density=14)
>>> line2 = OpticalDepth1D("HI1216", delta_v=100 * u.km/u.s)
```

Create the multi-component spectral model, defining a rest wavelength and explicitly defining some redshift value.

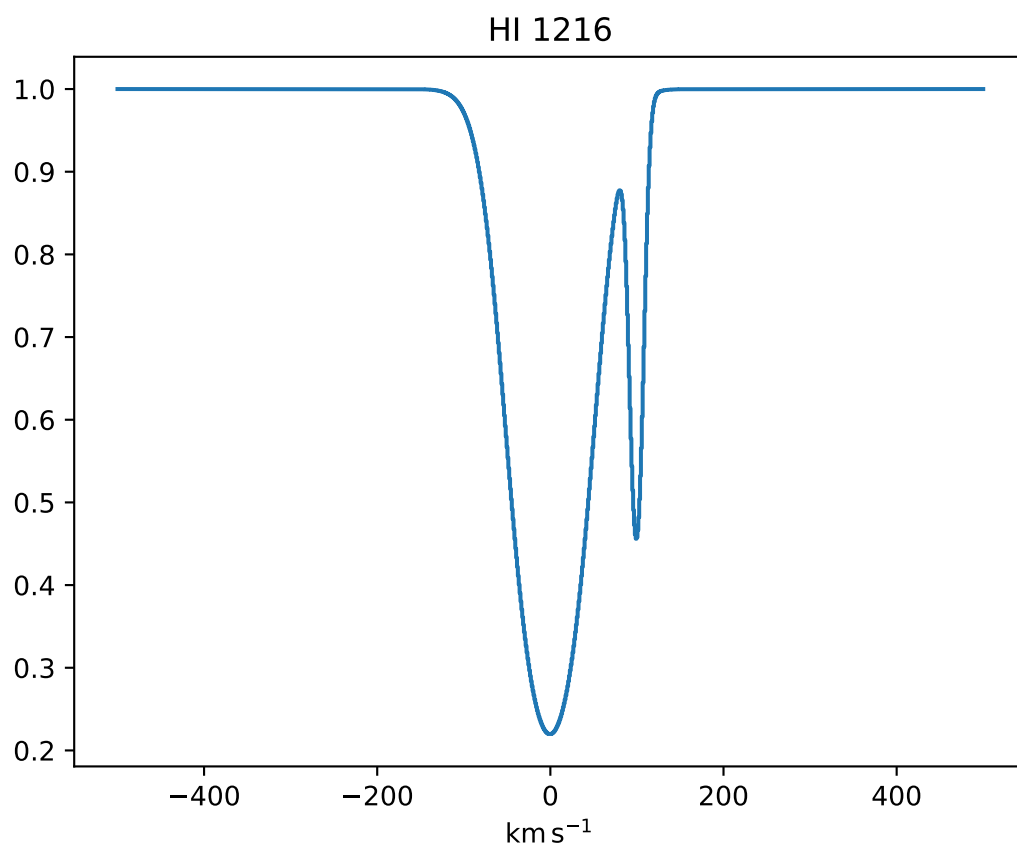
```
>>> spec_mod = Spectral1D([line, line2], continuum=1, z=0, output='flux')
```

Generate spectral data from the model.

```
>>> x = np.linspace(-500, 500, 1000) * u.Unit('km/s')
>>> y = spec_mod(x)
```

Plot the result.

```
>>> f, ax = plt.subplots()
>>> ax.set_title("HI 1216")
>>> ax.step(x, y)
```



Part II

Using Spectacle

INSTALLATION

1.1 Dependencies

The packages needed to run Spectacle should be installed automatically when the user installs the package. These dependencies include

- `astropy>=3.1`
- `specutils>=0.4`
- `emcee > 3.0`

1.2 Stable release

To install Spectacle, run this command in your terminal:

```
$ pip install spectacle
```

This is the preferred method to install Spectacle, as it will always install the most recent stable release.

If you don't have `pip` installed, this [Python installation guide](#) can guide you through the process.

1.3 From sources

The sources for Spectacle can be downloaded from the [Github repo](#).

You can either clone the public repository:

```
$ git clone git://github.com/MISTY-pipeline/spectacle
```

Or download the [tarball](#):

```
$ curl -OL https://github.com/MISTY-pipeline/spectacle/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```


CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given. You can contribute in many ways:

2.1 Types of Contributions

2.1.1 Report Bugs

Report bugs at <https://github.com/MISTY-pipeline/spectacle/issues>.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

2.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” and “help wanted” is open to whoever wants to implement it.

2.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with “enhancement” and “help wanted” is open to whoever wants to implement it.

2.1.4 Write Documentation

Spectacle could always use more documentation, whether as part of the official Spectacle docs, in docstrings, or even on the web in blog posts, articles, and such.

2.1.5 Submit Feedback

The best way to send feedback is to file an issue at <https://github.com/MISTY-pipeline/spectacle/issues>.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

2.2 Get Started!

Ready to contribute? Here's how to set up spectacle for local development.

1. Fork the spectacle repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/spectacle.git
```

3. Install your local copy into a virtualenv. Assuming you have Anaconda installed, this is how you set up your fork for local development:

```
$ conda create -n spectacle_env python=3
$ conda activate spectacle_env
$ cd spectacle/
$ python setup.py develop
```

Alternatively, **if** you have virtualenvwrapper installed::

```
$ mkvirtualenv spectacle
$ cd spectacle/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 spectacle tests
$ python setup.py test or py.test
$ tox
```

To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

2.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, 3.4 and 3.5, and for PyPy. Check https://travis-ci.org/MISTY-pipeline/spectacle/pull_requests and make sure that the tests pass for all supported Python versions.

2.4 Tips

To run a subset of tests:

```
$ py.test tests.test_spectacle
```


GETTING STARTED

Spectacle is designed to facilitate the creation of complex spectral models. It is built on the [Astropy modeling](#) module. It can create representations of spectral data in wavelength, frequency, or velocity space, with any number of line components. These models can then be fit to data for use in individual ion reduction, or characterization of spectra as a whole.

3.1 Creating a spectral model

The primary class is the [Spectral1D](#). This serves as the central object through which the user will design their model, and so exposes many parameters to the user.

[Spectral1D](#) models are initialized with several descriptive properties:

At-tribute	Description
z	This is the redshift of the spectrum. It can either be a single numerical value, or an array describing the redshift at each e.g. velocity bin. Note if the user provides an array, the length of the redshift array must match the length of the dispersion array used as input to the model. <i>Default: 0</i>
rest_wavelength	The rest wavelength at which transformations from wavelength space to velocity space will be performed. <i>Default: 0 Angstrom</i>
output	Describes the type of data the spectrum model will produce. This can be one of <code>flux</code> , <code>flux_decrement</code> , or <code>optical_depth</code> . <i>Default: 'flux'</i>
continuum	An FittableModel1D or single numeric value representing the continuum for the spectral model. <i>Default: 0</i>
lsf	The line spread function to be applied to the model. Users can provided an LSFModel , a Kernel1D , or a string indicating either <code>cos</code> for the COS LSF, or <code>gaussian</code> for a Gaussian LSF. In the latter case, the user should provide key word arguments as parameters for the Gaussian profile.
lines	The set of lines to be added to the spectrum. This information is passed to the OpticalDepth1D initializer. This can either be a single OpticalDepth1D instance, a list of OpticalDepth1D instances; a single string representing the name of an ion (e.g. "H1216"), a list of such strings; a single Quantity value representing the rest wavelength of an ion, or a list of such values.

The [Spectral1D](#) does not require that any lines be provided initially. In that case, it will just generate data only considering the continuum and other properties.

3.2 Providing lines to the initializer

As mentioned in the table above, lines can be added by passing a set of [OpticalDepth1D](#) instances, ion name strings, or rest wavelength [Quantity](#) objects to the initializer.

Using a set of OpticalDepth1D instances:

```
line = OpticalDepth1D("HI1216", v_doppler=50 * u.km/u.s, column_density=14)
line2 = OpticalDepth1D("HI1216", delta_v=100 * u.km/u.s)
spec_mod = Spectral1D([line, line2])
```

Using ion name strings:

```
spec_mod = Spectral1D(["HI1216", "OVI1032"])
```

Using rest wavelength Quantity objects:

```
spec_mod = Spectral1D([1216 * u.Angstrom, 1032 * u.Angstrom])
```

3.3 Adding lines after model creation

Likewise, the user can add a line to an already made spectral model by using the `with_line()` method, and provide to it information accepted by the OpticalDepth1D class

```
>>> from spectacle import Spectral1D
>>> import astropy.units as u
>>> spec_mod = Spectral1D([1216 * u.AA])
>>> spec_mod = spec_mod.with_line("HI1216", v_doppler=50 * u.km/u.s, column_density=14)
>>> print(spec_mod)
Model: Spectral1D
Inputs: ('x',)
Outputs: ('y',)
Model set size: 1
Parameters:
```

	amplitude_0	z_1	lambda_0_2	f_value_2	gamma_2	v_doppler_2	column_density_2	delta_v_2	delta_
↪	lambda_2	lambda_0_3	f_value_3	gamma_3	v_doppler_3	column_density_3	delta_v_3	delta_lambda_3	z_5
	Angstrom		Angstrom		km / s	km / s	km / s	Angstrom	
↪	0.0	0.0	1215.6701	0.4164	626500000.0	10.0	13.0	0.0	
↪	0.0	1215.6701	0.4164	626500000.0	50.0	14.0	0.0	0.0	0.0

MODELING

The purpose of Spectacle is to provide a descriptive model of spectral data, where each absorption or emission feature is characterized by an informative Voigt profile. To that end, there are several ways in which users can generate this model to more perfectly match their data, or the data they wish to create.

4.1 Defining output data

Support for three different types of output data exists: `flux`, `flux_decrement`, and `optical_depth`. This indicates the type of data that will be outputted when the model is run. Output type can be specified upon creation of a `spectacle.modeling.Spectral1D` object:

```
spec_mod = Spectral1D("HI1216", output='optical_depth')
```

Spectacle internally deals in optical depth space, and optical depth information is transformed into flux as a step in the compound model.

For flux transformations:

$$f(y) = np.exp(-y) - 1$$

And for flux decrement transformations:

$$f(y) = 1 - np.exp(-y) - 1$$

All output types use the continuum information when depositing absorption or emission data into the dispersion bins. Likewise, flux and flux_decrement will generate results that may be saturated.

```
>>> from astropy import units as u
>>> import numpy as np
>>> from matplotlib import pyplot as plt
>>> from spectacle.modeling import Spectral1D, OpticalDepth1D
```

```
>>> line = OpticalDepth1D("HI1216", v_doppler=50 * u.km/u.s, column_density=14)
>>> line2 = OpticalDepth1D("HI1216", delta_v=100 * u.km/u.s)
```

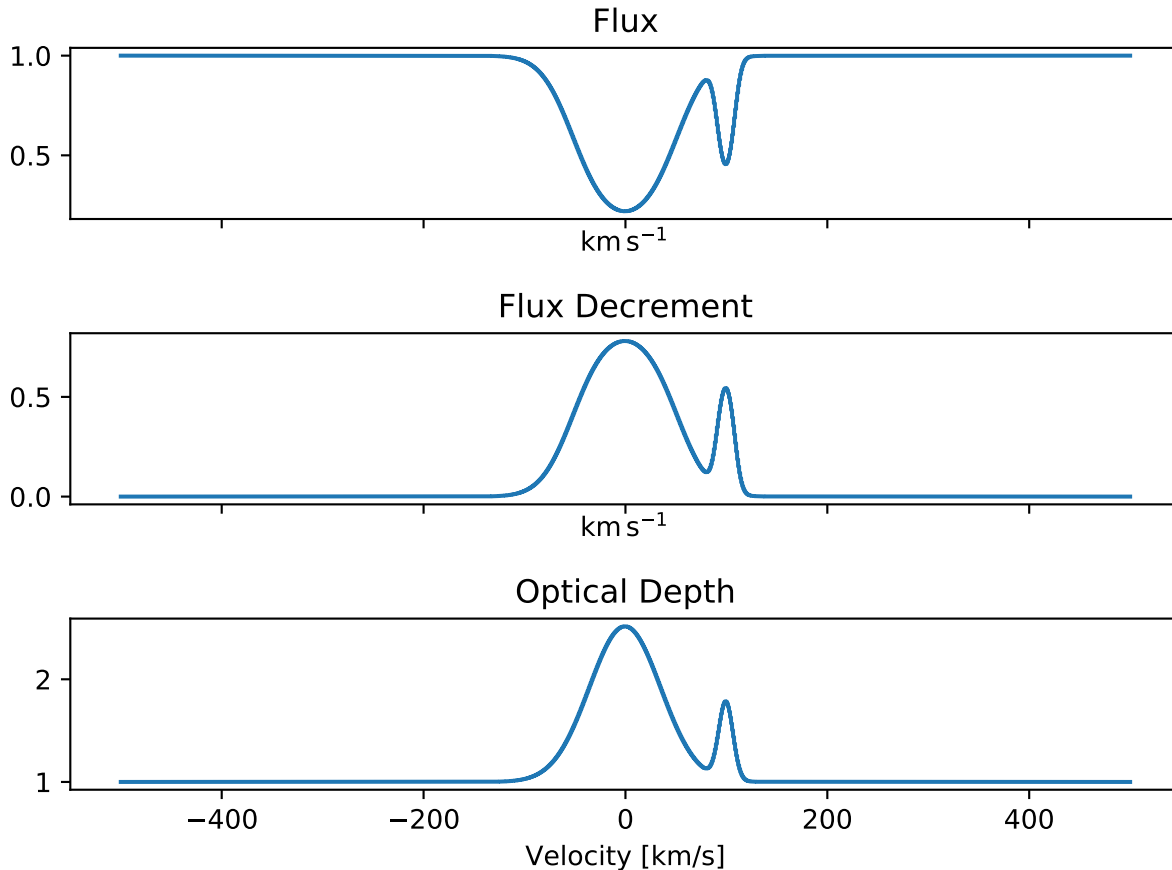
```
>>> spec_mod = Spectral1D([line, line2], continuum=1, output='flux')
>>> x = np.linspace(-500, 500, 1000) * u.Unit('km/s')
```

```
>>> flux = spec_mod(x)
>>> flux_dec = spec_mod.as_flux_decrement(x)
>>> tau = spec_mod.as_optical_depth(x)
```

```

>>> f, (ax1, ax2, ax3) = plt.subplots(3, 1, sharex=True)
>>> ax1.set_title("Flux")
>>> ax1.step(x, flux)
>>> ax2.set_title("Flux Decrement")
>>> ax2.step(x, flux_dec)
>>> ax3.set_title("Optical Depth")
>>> ax3.step(x, tau)
>>> ax3.set_xlabel('Velocity [km/s]')
>>> f.tight_layout()

```



4.2 Applying line spread functions

LSFs can be added to the `Spectral1D` model to generate data that more appropriately matches what one might expect from an instrument like, e.g., HST COS

```

>>> from astropy import units as u
>>> import numpy as np
>>> from matplotlib import pyplot as plt
>>> from spectacle.modeling import Spectral1D, OpticalDepth1D

```

```

>>> line1 = OpticalDepth1D("HI1216", v_doppler=500 * u.km/u.s, column_density=14)
>>> line2 = OpticalDepth1D("OVI1032", v_doppler=500 * u.km/u.s, column_density=15)

```

LSFs can either be applied directly during spectrum model creation:

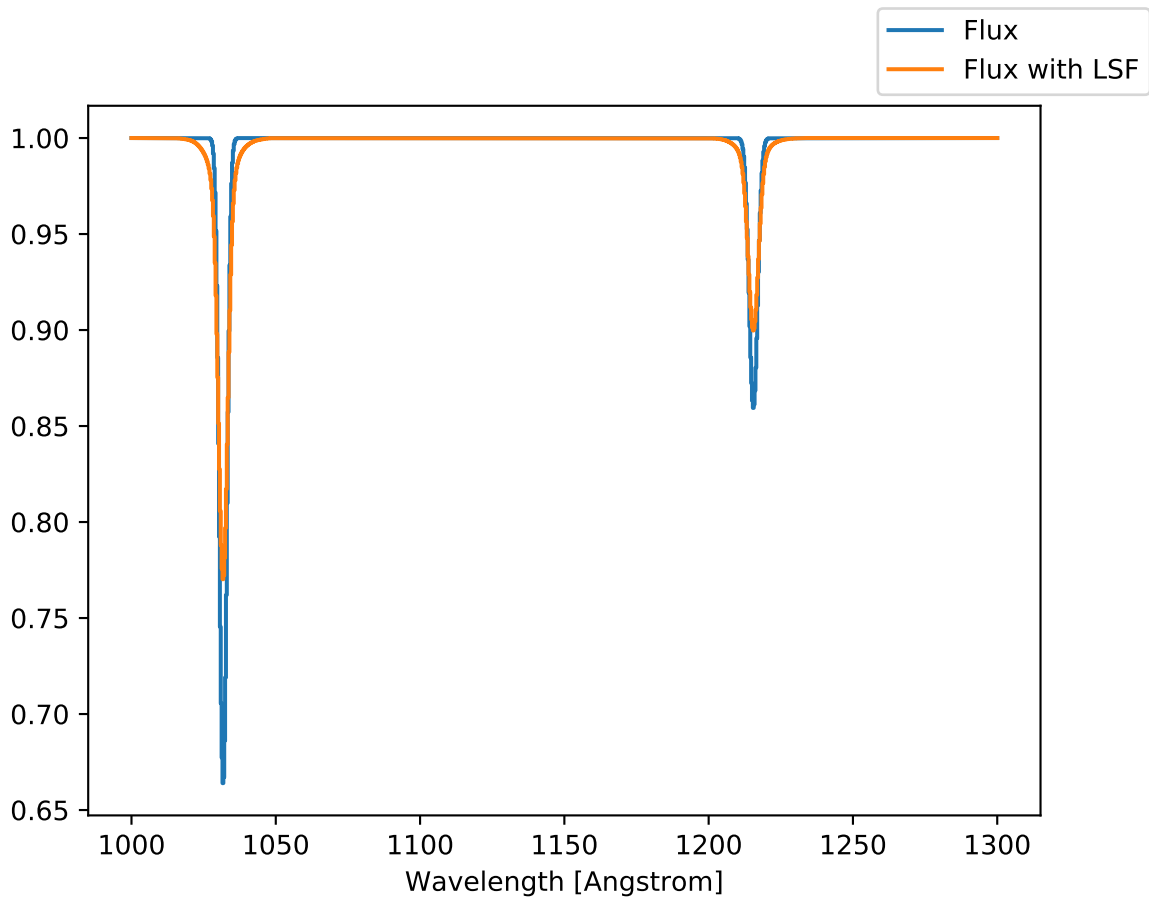
```
>>> spec_mod_with_lsf = Spectral1D([line1, line2], continuum=1, lsf='cos', output='flux')
```

or they can be applied after the fact:

```
>>> spec_mod = Spectral1D([line1, line2], continuum=1, output='flux')
>>> spec_mod_with_lsf = spec_mod.with_lsf('cos')
```

```
>>> x = np.linspace(1000, 1300, 1000) * u.Unit('Angstrom')
```

```
>>> f, ax = plt.subplots()
>>> ax.step(x, spec_mod(x), label="Flux")
>>> ax.step(x, spec_mod_with_lsf(x), label="Flux with LSF")
>>> ax.set_xlabel("Wavelength [Angstrom]")
>>> f.legend(loc=0)
```



4.2.1 Supplying custom LSF kernels

Spectacle provides two built-in LSF kernels: the HST COS kernel, and a Gaussian kernel. Both can be applied by simply passing in a string, and in the latter case, also supplying an additional `stddev` keyword argument:

```
.. code-block:: python
```

```
spec_mod = Spectral1D("HI1216", continuum=1, lsf='cos') spec_mod = Spectral1D("HI1216", continuum=1, lsf='gaussian', stddev=15)
```

Users may also supply their own kernels, or any [Astropy 1D kernel](#). The only restriction is that kernels must be a subclass of either `LSFModel`, or `Kernel1D`.

```
from astropy.convolution import Box1DKernel
kernel = Box1DKernel(width=10)

spec_mod_with_lsf = Spectral1D([line1, line2], continuum=1, lsf=kernel, output='flux')
```

4.3 Converting dispersions

Spectacle supports dispersions in either wavelength space or velocity space, and will implicitly deal with conversions internally as necessary. Conversion to velocity space is calculated using the relativistic doppler equation

$$V = c \frac{f_0^2 - f^2}{f_0^2 + f^2},$$
$$f(V) = f_0 \frac{(1 - (V/c)^2)^{1/2}}{(1 + V/c)}.$$

This of course makes the assumption that observed redshift is due to relativistic effects along the light of sight. At higher redshifts, however, the predominant source of observed redshift is due to the cosmological expansion of space, and not the source's velocity with respect to the observer.

It is possible to set the approximation used in wavelength/frequency to velocity conversions for Spectacle. Aside from the default relativistic calculation, users can choose the “optical definition”

$$V = c \frac{f_0 - f}{f}$$
$$f(V) = f_0(1 + V/c)^{-1}$$

or the “radio definition”

$$V = c \frac{f_0 - f}{f_0}$$
$$f(V) = f_0(1 - V/c).$$

This can be done upon instantiation of the `Spectral1D` model:

```
spec_mod = Spectral1D("HI1216", continuum=1, z=0, velocity_convention='optical')
```

The `velocity_convention` keyword supports one of either `relativistic`, `optical`, or `radio` to indicate the definition to be used in internal conversions.

4.4 Implementing redshift

When creating a `Spectral1D` model, the user can provide a redshift at which the output spectrum will deposit the lines by including a `z` parameter.

Note: When *fitting*, including the *z* parameter indicates the redshift of the *input* dispersion. Spectacle will de-redshift the data input using this value before performing any fits. Also, the provided continuum is *not* included in redshifting.

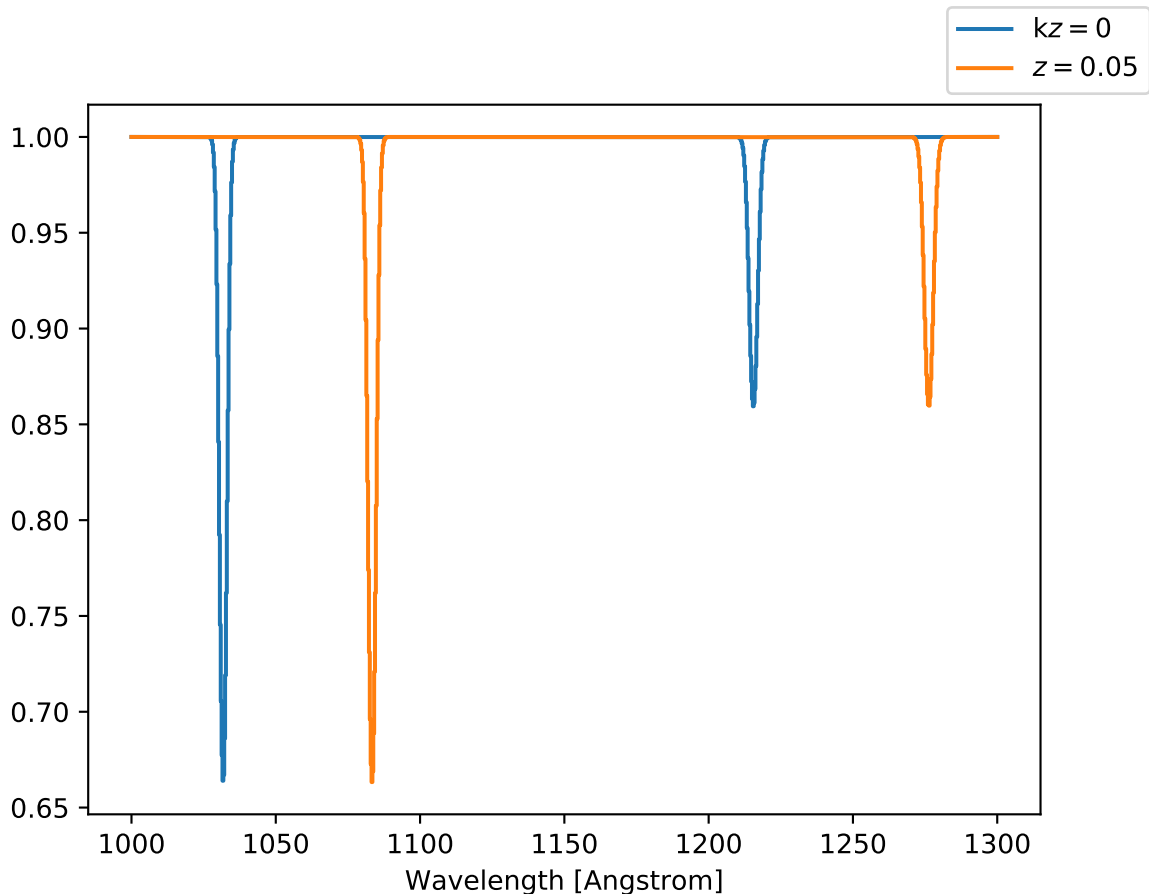
```
>>> from astropy import units as u
>>> import numpy as np
>>> from matplotlib import pyplot as plt
>>> from spectacle.modeling import Spectral1D, OpticalDepth1D
```

```
>>> line1 = OpticalDepth1D("HI1216", v_doppler=500 * u.km/u.s, column_density=14)
>>> line2 = OpticalDepth1D("OVI1032", v_doppler=500 * u.km/u.s, column_density=15)
```

```
>>> spec_mod = Spectral1D([line1, line2], continuum=1, z=0, output='flux')
>>> spec_mod_with_z = Spectral1D([line1, line2], continuum=1, z=0.05, output='flux')
```

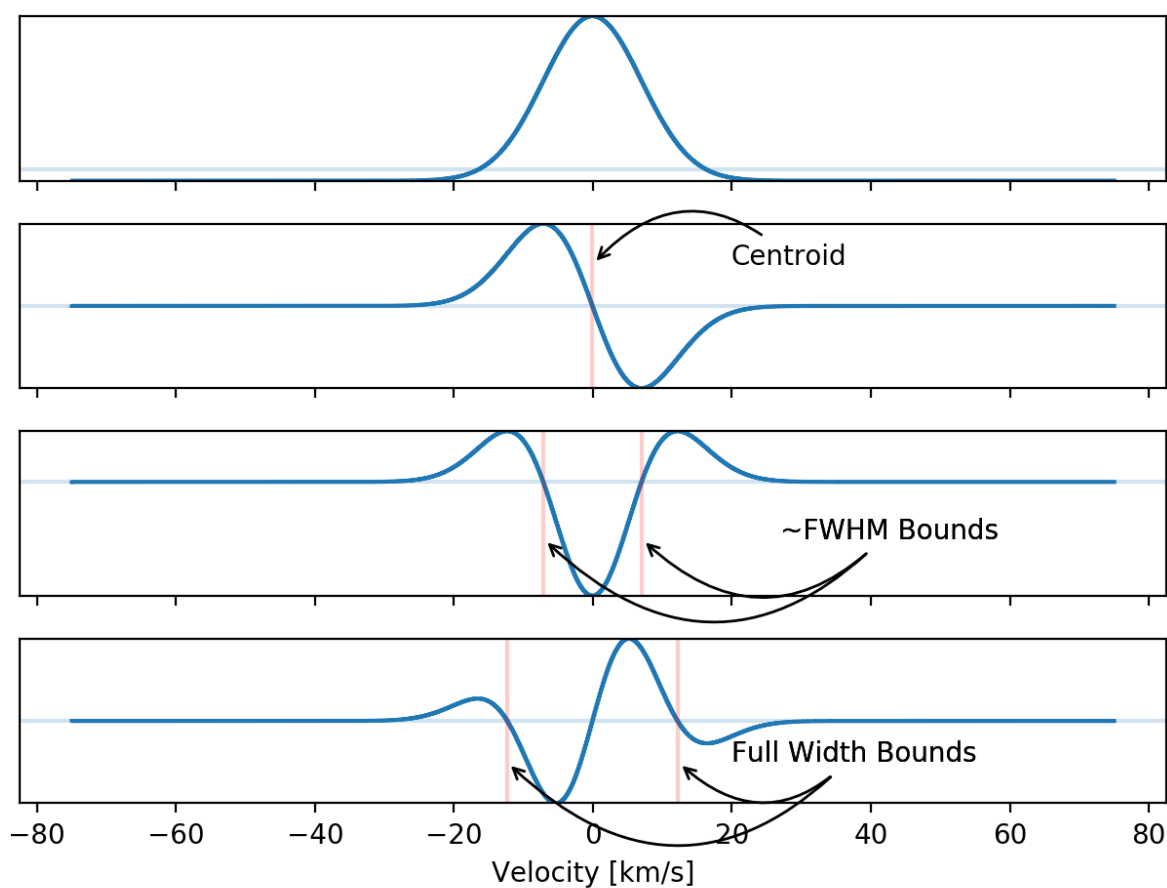
```
>>> x = np.linspace(1000, 1300, 1000) * u.Unit('Angstrom')
```

```
>>> f, ax = plt.subplots()
>>> ax.step(x, spec_mod(x), label="k$z=0$")
>>> ax.step(x, spec_mod_with_z(x), label="$z=0.05$")
>>> ax.set_xlabel("Wavelength [Angstrom]")
>>> f.legend(loc=0)
```



LINE FINDING

Spectacle has automated line-finding and characterization, including support for blended features. The line finder uses a series of convolutions to indicate where the slope of the spectrum changes, and uses these positions to characterize the shape of the absorption or emission lines. Such characterization includes finding the centroid, the width of the line, and the whether or not lines are buried within the region of another.



The line finder will return a generated spectral model from the found lines.

5.1 Auto-generate spectral model

The inputs to the `LineFinder` class help dictate the behavior of the finding routine and what lines the finder will consider when it believes it has found a feature.

The user may note that the line finder takes many of the same parameters accepted when creating a `Spectral1D` model explicitly. This is because these parameter are based onto the internal initialization of the spectral model. The accepted line finder arguments are summarized in the table below.

Argument	Description
ions	The subset of ion information to use when parsing potential profiles in the spectral data.
continuum	A <code>FittableModel1D</code> or single numeric value, representing the continuum for the spectral model. <i>Default: 0</i>
defaults	A dictionary describing the default arguments passed to internal <code>OpticalDepth1D</code> profiles.
auto_fit	Whether the line finder should attempt to automatically fit the resulting spectral model to the provided data.
velocity_convention	The velocity convention used in internal conversions between wavelength/frequency and velocity space.
output	Describes the type of data the spectrum model will produce. This can be one of <code>flux</code> , <code>flux_decrement</code> , or <code>optical_depth</code> . <i>Default: 'flux'</i>

The `ions` argument allows the user to select a subset of the entire ion table for the line finder to use when attempting to parse a particular profile. If no ions are provided, the entire ion table will be searched to find the ion whose λ_0 most closely aligns with the detected centroid.

Note: Currently, when running the line finder in velocity space, only one ion definition is supported at a time. This is because there is no way to disambiguate the kinematics of multiple ions simultaneously in velocity space.

As an example of the line finder functionality, let us define two lines within a set of “fake” data.

```
line1 = OpticalDepth1D("HI1216", v_doppler=20 * u.km/u.s, column_density=16)
line2 = OpticalDepth1D("OVI1032", v_doppler=60 * u.km/u.s, column_density=14)

spec_mod = Spectral1D([line1, line2], continuum=1, output='optical_depth')
```

We will describe the lines in wavelength space to disambiguate their emission profiles without considering kinematics.

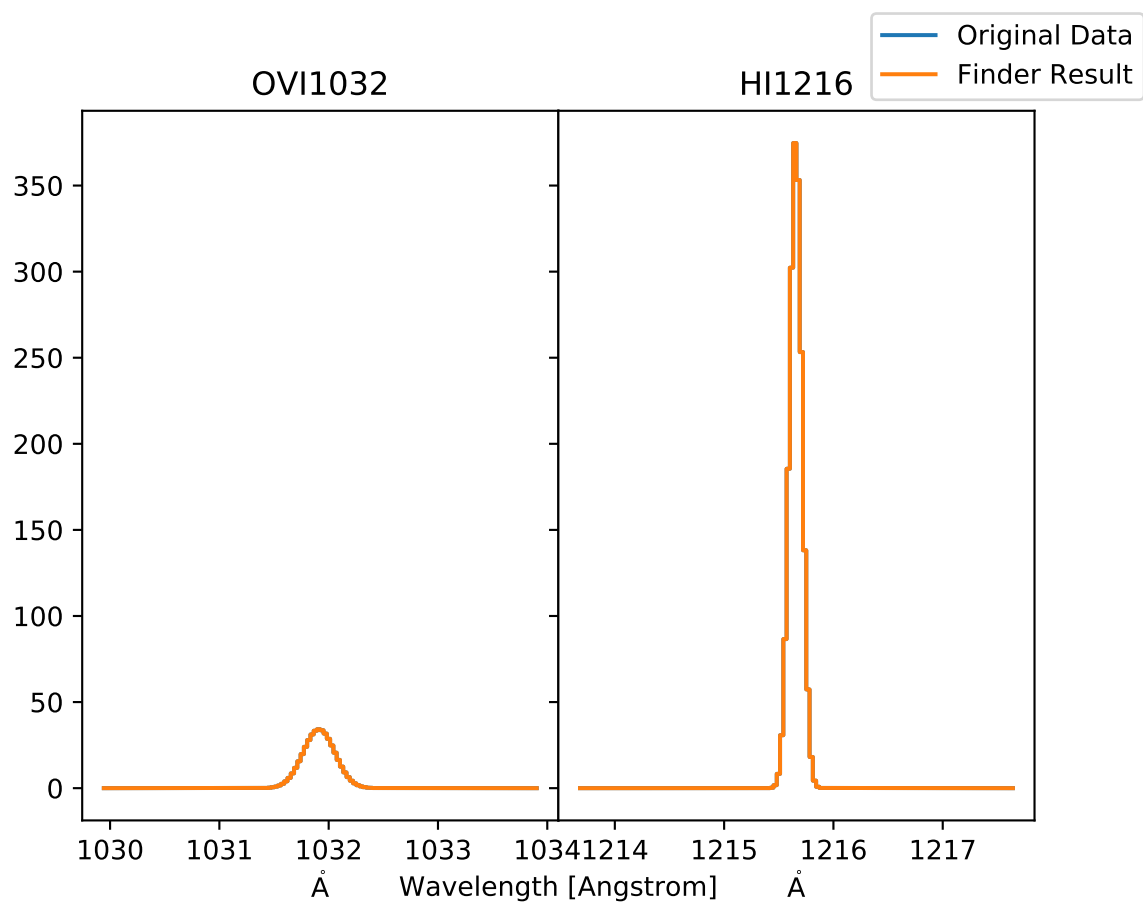
```
x = np.linspace(1000, 1300, 1000) * u.Unit('Angstrom')
y = spec_mod(x)
```

Now we tell the line finder that there are two possible lines that any feature it comes across could be. Doing so means that any `OpticalDepth1D` profiles generated will be given the correct attributes (e.g. `f_value`, `gamma`, etc.) when the line finder attempts to lookup the ion in the ion table.

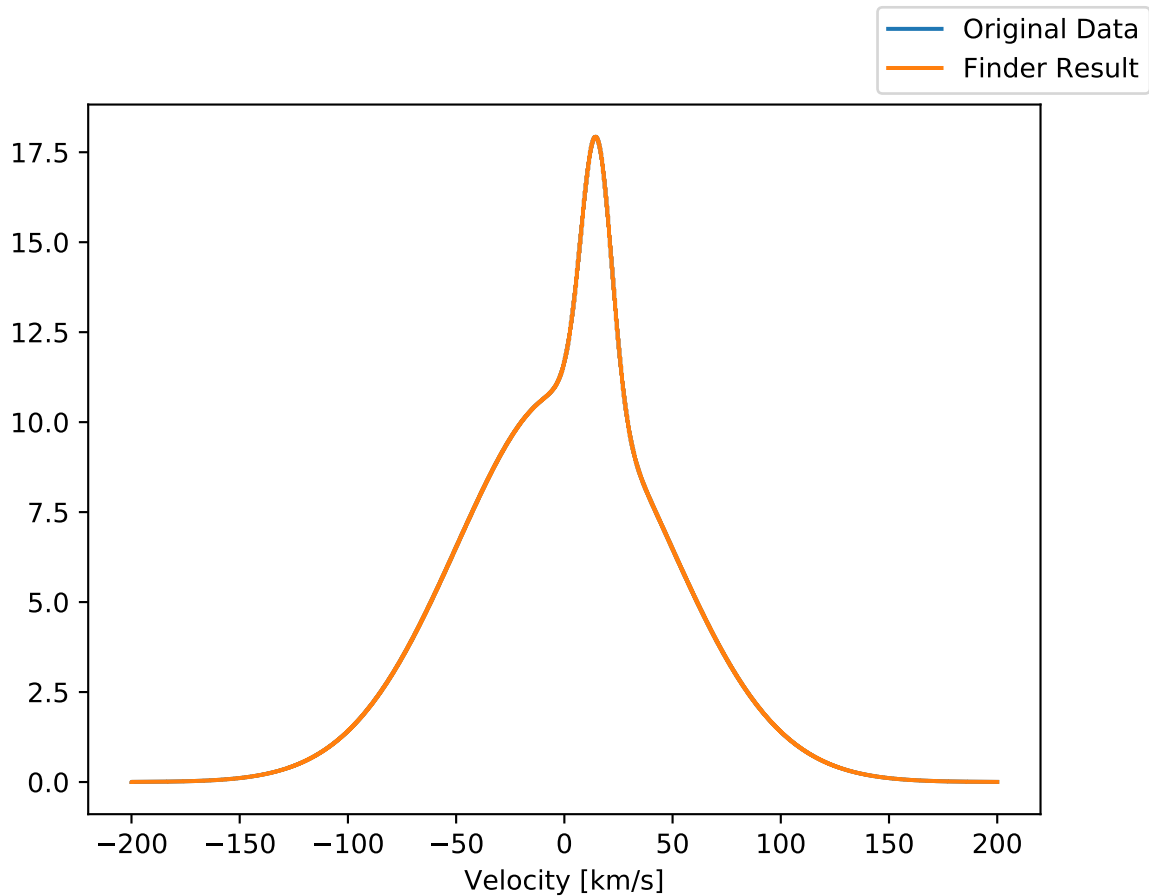
```
line_finder = LineFinder1D(ions=["HI1216", "OVI1032"], continuum=0, output='optical_depth')
finder_spec_mod = line_finder(x, y)
```

5.1.1 Dealing with buried lines

The line finder will implicitly deal with buried lines by considering the results of the series of convolutions and looking for characteristics that might indicate that the peak of a line is buried within the region of another profile.



The basic premise for determining the possibility of a buried line is to compare the bounds of the second differencing convolution with that of the third. A buried line is one which, in both cases, share the same found centroid, but two different sets of region bounds.



5.2 Defining default parameters

The line finder class can accept a defaults dictionary whose values will be applied when new `OpticalDepth1D` profiles are initialized. This is an easy way, for example, to manually set global parameter bounds information that would otherwise be up to Spectacle to determine.

```
defaults_dict = {
    'v_doppler': {
        'bounds': (-10, 10),
        'fixed': False
    },
    'column_density' = {
        'bounds': (13, 18)
    }
}

line_finder = LineFinder1D(ions=["HI1216"], defaults=defaults_dict, continuum=0, output='optical_depth')
```

5.3 Searching for ion subsets

As mentioned above, the line finder attempts to retrieve information about a potential profile by looking up the detected centroid in the ion table and selecting the nearest match. (A more extensive overview of the line registry can be found in the [line registry docs](#)). The user can provide a subset of ions that will help to narrow the possible options available to the line finder by passing in a list of ion names or λ_0 values. The default ion list is provided by Morton (2003).

Subsets behave by limiting the entire table of ions in the registry to some specified list:

```

1 >>> from spectacle.registries.lines import line_registry
2 >>> print(line_registry)
3
4 <LineRegistry length=329>
5   name      wave      osc_str      gamma
6   str9      float64    float64    float64
7   -----
8   HI1216 1215.6701    0.4164 626500000.0
9   HI1026 1025.7223    0.07912 189700000.0
10  HI973  972.5368      0.029 81270000.0
11  HI950  949.7431      0.01394 42040000.0
12  HI938  937.8035      0.007799 24500000.0
13  HI931  930.7483      0.004814 12360000.0
14  HI926  926.2257      0.003183 8255000.0
15  ...
16  ...
17  NiII1317 1317.217    0.07786 420500000.0
18  CuII1368 1367.9509    0.179 623000000.0
19  CuII1359 1358.773    0.3803 720000000.0
20  ZnII2063 2062.664      0.256 386000000.0
21  ZnII2026 2026.136      0.489 407000000.0
22  GeII1602 1602.4863    0.1436 990600000.0
23  GaII1414 1414.402      1.8 197000000.0
24
25 >>> subset = line_registry.subset(["HI1216", "NiII1468", "ZnII2026", "CoII1425"])
26 >>> print(subset)
27
28 <LineRegistry length=4>
29   name      wave      osc_str      gamma
30   str9      float64    float64    float64
31   -----
32   CoII1425 1424.7866    0.0109 35800000.0
33   HI1216 1215.6701    0.4164 626500000.0
34   NiII1468 1467.756      0.0099 23000000.0
35   ZnII2026 2026.136      0.489 407000000.0
36

```

Passing in a list to the `spectacle.fitting.LineFinder1D` will internally do this for the user

```

line_finder = LineFinder1D(ions=["HI1216", "NiII1468", "ZnII2026", "CoII1425"], continuum=0, output=
↳ 'optical_depth')

```

Warning: Only a single ion can be defined for the line finder if the user provides the dispersion in velocity space. This is because the line finder cannot disambiguate ions based on their kinematics.

FITTING

The core `Spectral1D` behaves exactly like an Astropy model and can be used with any of the supported non-linear Astropy fitters, as well as some not included in the Astropy library.

Spectacle provides a default Levenberg–Marquardt fitter in the `CurveFitter` class.

```
>>> from spectacle.fitting import CurveFitter
>>> from spectacle.modeling import Spectral1D, OpticalDepth1D
>>> import astropy.units as u
>>> from matplotlib import pyplot as plt
>>> import numpy as np
```

Generate some fake data to fit to:

```
>>> line1 = OpticalDepth1D("HI1216", v_doppler=10 * u.km/u.s, column_density=14)
>>> spec_mod = Spectral1D(line1, continuum=1, output='flux')
>>> x = np.linspace(-200, 200, 1000) * u.Unit('km/s')
>>> y = spec_mod(x) + (np.random.sample(1000) - 0.5) * 0.01
```

Instantiate the fitter and fit the model to the data:

```
>>> fitter = CurveFitter()
>>> fit_spec_mod = fitter(spec_mod, x, y)
```

Users can see the results of the fitted spectrum by printing the returned model object

```
>>> print(fit_spec_mod)
Model: Spectral1D
Inputs: ('x',)
Outputs: ('y',)
Model set size: 1
Parameters:
  amplitude_0 z_1 lambda_0_2 f_value_2 gamma_2 v_doppler_2 column_density_2 delta_v_
↪2          delta_lambda_2 z_4
          Angstrom
↪          Angstrom
-----
↪-----
          1.0 0.0 1215.6701 0.4164 626500000.0 10.010182187404824 13.998761432240995 1.
↪0052009119192702 -0.004063271434522016 0.0
```

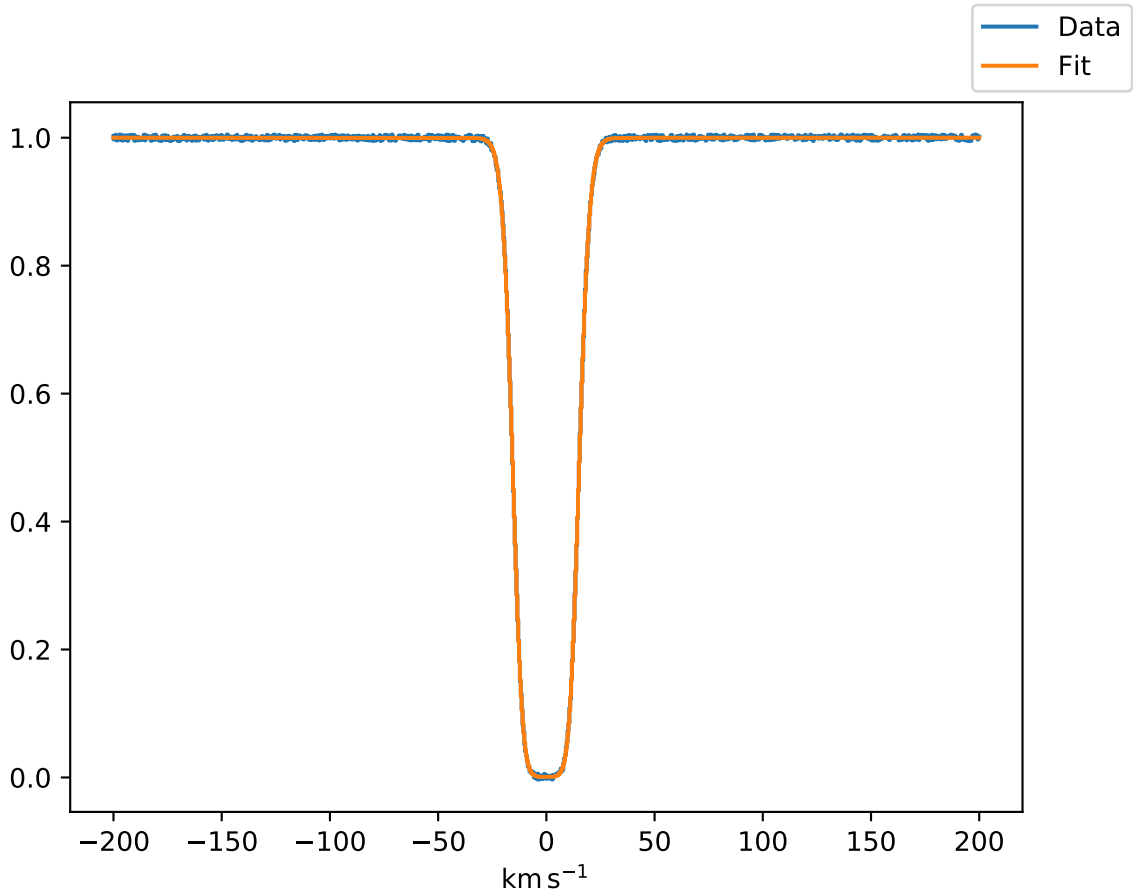
Plot the results:

```
>>> f, ax = plt.subplots()
>>> ax.step(x, y, label="Data")
```

(continues on next page)

(continued from previous page)

```
>>> ax.step(x, fit_spec_mod(x), label="Fit")
>>> f.legend()
```



On both the [CurveFitter](#) class and the [EmceeFitter](#) class described below, parameter uncertainties can be accessed through the `uncertainties` property of the instantiated fitter after the fitting routine has run.

```
>>> fitter.uncertainties
<QTable length=9>
  name          value          uncert          unit
  str16         float64         float64         object
-----
      z_0              0.0              0.0         None
  lambda_0_1      1215.6701              0.0   Angstrom
   f_value_1              0.4164              0.0         None
   gamma_1      626500000.0              0.0         None
 v_doppler_1  10.000013757295898  0.000957197044912263   km / s
column_density_1  14.000043173540684  3.589807779429899e-05         None
   delta_v_1  0.00011598087488537782  0.0006777042342563724   km / s
 delta_lambda_1              0.0              0.0   Angstrom
 amplitude_2              1.0              0.0         None
```

6.1 Using the MCMC fitter

Spectacle provides Bayesian fitting through the `emcee` package. This is implemented in the `EmceeFitter` class. The usage is similar above, but extra arguments can be provided to control the number of walkers and the number of iterations.

```
from spectacle.fitting import EmceeFitter
...

fitter = EmceeFitter()
fit_spec_mod = fitter(spec_mod, x, y, , nwalkers=250, steps=100, nprocs=8)
```

The fitted parameter results are given as the value at the 50th quantile of the distribution of walkers. The uncertainties on the values can be obtained through the `uncertainties` property on the fitter instance, and provide the 16th quantile and 80th quantile for the lower and upper bounds on the value, respectively.

Note: The MCMC fitter is a work in progress. Its results are dependent on how long the fitter runs and how many walkers are provided.

6.2 Custom fitters with the line finder

The `LineFinder1D` class can also be passed a fitter instance if the user wishes to use a specific type. If no explicit fitting class is passed, the default `CurveFitter` is used. Fitter-specific arguments can be passed into the `fitter_args` keyword as well.

```
1 line_finder = LineFinder1D(ions=["HI1216", "OVI1032"], continuum=0,
2                             output='optical_depth', fitter=LevMarLSQFitter(),
3                             fitter_args={'maxiter': 1000})
```

More information on using the line finder can be found in the [line finding documentation](#).

REGISTRIES

Spectacle uses an internal database of ions in order to look up relevant atomic line information (specifically, oscillator strength, gamma values, and rest-frame wavelength). Spectacle provides an extensive default ion registry taken from Morton 2003. However, it is possible for users to provide their own registries.

Spectacle searches the line registry for an atomic transition with the closest rest-frame wavelength to the line in question. Alternatively, users can provide a specific set of lines with associated atomic information for Spectacle to use.

```
>>> from spectacle.registries import line_registry
>>> import astropy.units as u
```

Users can query the registry by passing in the restframe wavelength, λ_0 , information for an ion

```
>>> from spectacle.registries import line_registry
>>> line_registry.with_lambda(1216 * u.AA)
<Row index=0>
  name      wave      osc_str      gamma
      Angstrom
  str9      float64      float64      float64
-----
HI1216 1215.6701  0.4164 626500000.0
```

Alternatively users can pass ion names in their queries. Spectacle will attempt to find the closest alpha-numerical match using an internal auto-correct:

```
>>> from spectacle.registries import line_registry
>>> line_registry.with_name("HI1215")
spectacle [INFO    ]: Found line with name 'HI1216' from given name 'HI1215'.
<Row index=0>
  name      wave      osc_str      gamma
      Angstrom
  str9      float64      float64      float64
-----
HI1216 1215.6701  0.4164 626500000.0
```

The default ion registry can be seen in its entirety [here](#).

7.1 Adding your own ion registry

Users can provide their own registries to replace the internal default ion database used by Spectacle. The caveat is that the file must be an *ECSV* file with four columns: name, wave, osc_str, gamma. The user's file can then be loaded by importing and the `spectacle.registries.LineRegistry` and declaring the `line_registry` variable

```
>>> from spectacle.registries import LineRegistry
>>> line_registry = LineRegistry.read("/path/to/ion_file.ecsv")
```

ANALYSIS

Spectacle comes with several statistics on the line profiles of models. These can be accessed via the `line_stats()` method on the spectrum object, and will display basic parameter information such as the centroid, column density, doppler velocity, etc. of each line in the spectrum.

Three additional statistics are added to this table as well: the equivalent width (ew), the velocity delta covering 90% of the flux value (dv90), and the full width half maximum (fwhm) of the feature.

```
>>> spec_mod.line_stats(x)

<QTable length=2>
  name      wave   col_dens  v_dop  delta_v  delta_lambda      ew      dv90
  fwhm
      Angstrom      km / s  km / s  Angstrom  Angstrom      km / s
  Angstrom
bytes10 float64 float64 float64 float64 float64 float64 float64
  float64
-----
  HI1216 1215.6701    13.0    7.0    0.0      0.0 0.050400091274475814 15.21521521521521 0.
  048709144294889484
  HI1216 1215.6701    13.0   12.0   30.0      0.0 0.08632151159859157 26.426426426426445 0.
  08119004034051613
```

The dv90 statistic is calculated following the “Velocity Interval Test” formulation defined in Prochaska & Wolf (1997). In this case, the optical depth of the profile is calculated in velocity space, and 5% of the total is trimmed from each side, leaving the velocity width encompassing the central 90%.

8.1 Arbitrary line region statistics

It is also possible to perform statistical operations over arbitrary absorption or emission line regions. Regions are defined as contiguous portions of the profile beyond some threshold.

```
>>> from spectacle.modeling import Spectral1D, OpticalDepth1D
>>> import astropy.units as u
>>> from matplotlib import pyplot as plt
>>> import numpy as np
>>> from astropy.visualization import quantity_support
>>> quantity_support()
```

We’ll go ahead and compose a spectrum in velocity space of two HI line profiles, with one offset by $30 \frac{km}{s}$.

```
>>> line1 = OpticalDepth1D(lambda_0=1216 * u.AA, v_doppler=7 * u.km/u.s, column_density=13, delta_v=0 * u.km/u.s)
>>> line2 = OpticalDepth1D("HI1216", delta_v=30 * u.km/u.s, v_doppler=12 * u.km/u.s, column_density=13)
```

```
>>> spec_mod = Spectral1D([line1, line2], continuum=1, output='flux')
```

```
>>> x = np.linspace(-200, 200, 1000) * u.km / u.s
>>> y = spec_mod(x)
```

Now, print the statistics on this absorption region.

```
>>> region_stats = spec_mod.region_stats(x, rest_wavelength=1216 * u.AA, abs_tol=0.05)
>>> print(region_stats)
```

region_start	region_end	rest_wavelength	ew	dv90	dv50
km / s	km / s	Angstrom	Angstrom	km / s	km / s
-12.612612612612594	48.648648648648674	1216.0	0.12297380252108686	46.44644644644646	0.056842794376279926

Plot to show the found bounds of the contiguous absorption region.

```
>>> f, ax = plt.subplots()
>>> ax.axhline(0.95, linestyle='--', color='k', alpha=0.5)
>>> for row in region_stats:
...     ax.axvline(row['region_start'].value, color='r', alpha=0.5)
...     ax.axvline(row['region_end'].value, color='r', alpha=0.5)
>>> ax.step(x, y)
```

8.2 Re-sampling dispersion grids

Spectacle provides a means of doing flux-conserving re-sampling by generating a re-sampling matrix based on an input spectral dispersion grid and a desired output grid.

```
>>> from spectacle.modeling import Spectral1D, OpticalDepth1D
>>> from spectacle.analysis import Resample
>>> import astropy.units as u
>>> from matplotlib import pyplot as plt
>>> import numpy as np
>>> from astropy.visualization import quantity_support
>>> quantity_support()
```

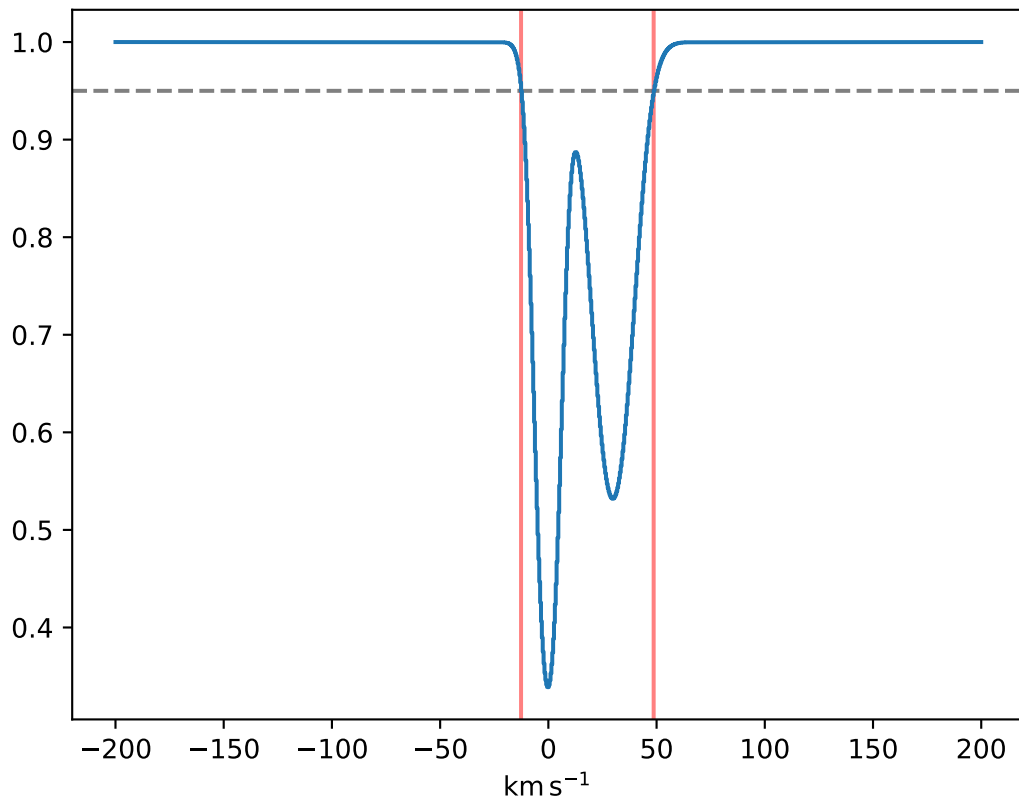
Create a basic spectral model.

```
>>> line1 = OpticalDepth1D("HI1216")
>>> spec_mod = Spectral1D(line1)
```

Define our original, highly-sampled dispersion grid.

```
>>> vel = np.linspace(-50, 50, 1000) * u.km / u.s
>>> tau = spec_mod(vel)
```

Define a new, lower-sampled dispersion grid we want to re-sample to.



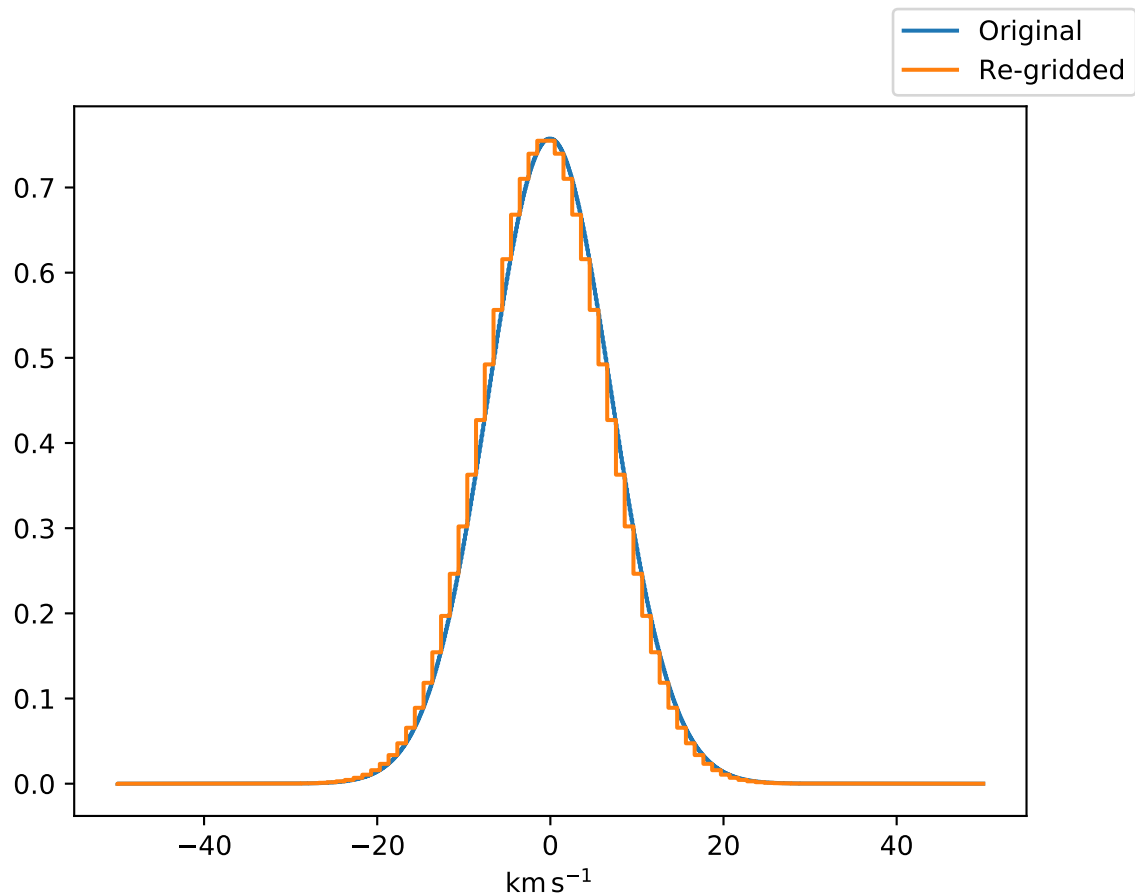
```
>>> new_vel = np.linspace(-50, 50, 100) * u.km / u.s
```

Generate the resampling matrix and apply it to the original data.

```
>>> resample = Resample(new_vel)
>>> new_tau = resample(vel, tau)
```

Plot the results.

```
>>> f, ax = plt.subplots()
>>> ax.step(vel, tau, label="Original")
>>> ax.step(new_vel, new_tau, label="Re-gridded")
>>> f.legend()
```



Part III

API

9.1 Modeling

9.2 spectacle.modeling Package

9.2.1 Classes

<code>OpticalDepth1D([name, line_list])</code>	Implements a Voigt profile astropy model.
<code>Spectral1D(*args, **kwargs)</code>	Base representation of a compound model containing a variable number of <code>OpticalDepth1D</code> line model features.

OpticalDepth1D

class `spectacle.modeling.OpticalDepth1D`(*name=None, line_list=None, *args, **kwargs*)

Bases: `astropy.modeling.Fittable1DModel`

Implements a Voigt profile astropy model. This model generates optical depth profiles for absorption features.

Parameters

lambda_0

[float] Central wavelength in Angstroms.

f_value

[float] Absorption line oscillator strength.

gamma

[float] Absorption line gamma value.

v_doppler

[float] Doppler b-parameter in km/s.

column_density

[float] Column density in cm⁻².

delta_v

[float] Velocity offset from lambda_0 in km/s. Default: None (no shift).

delta_lambda

[float] Wavelength offset in Angstrom. Default: None (no shift).

lambda_bins

[array-like] Wavelength array for line deposition in Angstroms. If None, one will created using `n_lambda` and `dlambda`. Default: None.

n_lambda

[int] Size of lambda bins to create if `lambda_bins` is None. Default: 12000.

dlambda

[float] Lambda bin width in Angstroms if `lambda_bins` is None. Default: 0.01.

Returns**tau_phi**

[array-like] An array of optical depth values.

Attributes Summary

<code>column_density</code>	
<code>delta_lambda</code>	
<code>delta_v</code>	
<code>f_value</code>	
<code>gamma</code>	
<code>input_units</code>	This property is used to indicate what units or sets of units the evaluate method expects, and returns a dictionary mapping inputs to units (or <code>None</code> if any units are accepted).
<code>inputs</code>	
<code>lambda_0</code>	
<code>outputs</code>	
<code>param_names</code>	
<code>v_doppler</code>	

Methods Summary

<code>__call__(self, x[, model_set_axis, ...])</code>	Evaluate this model using the given input(s) and the parameter values that were specified when the model was instantiated.
<code>evaluate(self, x, lambda_0, f_value, gamma, ...)</code>	Evaluate the model on some input variables.
<code>fit_deriv(x, x_0, b, gamma, f)</code>	
<code>full_width_half_max(self, x)</code>	
<code>voigt(a, u)</code>	

Attributes Documentation

column_density

delta_lambda

delta_v

f_value

gamma

input_units

This property is used to indicate what units or sets of units the evaluate method expects, and returns a dictionary mapping inputs to units (or `None` if any units are accepted).

Model sub-classes can also use function annotations in evaluate to indicate valid input units, in which case this property should not be overridden since it will return the input units based on the annotations.

inputs = ('x',)

lambda_0

outputs = ('y',)

param_names = ('lambda_0', 'f_value', 'gamma', 'v_doppler', 'column_density', 'delta_v', 'delta_lambda')

v_doppler

Methods Documentation

__call__(self, x, model_set_axis=None, with_bounding_box=False, fill_value=nan, equivalencies=None)

Evaluate this model using the given input(s) and the parameter values that were specified when the model was instantiated.

evaluate(self, x, lambda_0, f_value, gamma, v_doppler, column_density, delta_v, delta_lambda)

Evaluate the model on some input variables.

static fit_deriv(x, x_0, b, gamma, f)

full_width_half_max(self, x)

static voigt(a, u)

Spectral1D

class `spectacle.modeling.Spectral1D(*args, **kwargs)`

Bases: `astropy.modeling.Fittable1DModel`

Base representation of a compound model containing a variable number of `OpticalDepth1D` line model features.

Parameters

lines

[str, `OpticalDepth1D`, list] The line information used to compose the spectral model. This

can be either a string, in which case the line information is retrieve from the ion registry; an instance of `OpticalDepth1D`; or a list of either of the previous two types.

continuum

[`Fittable1DModel`, optional] An astropy model representing the continuum of the spectrum. If not provided, a `Const1D` model is used.

z

[float, optional] The redshift applied to the spectral model. Default = 0.

lsf

[`LSFModel`, `Kernel1D`, str, optional] The line spread function applied to the spectral model. It can be a pre-defined kernel model, or a convolution kernel, or a string referencing the built-in Hubble COS lsf, or a Gaussian lsf. Optional keyword arguments can be passed through.

Attributes Summary

<code>as_flux</code>	New spectral model that produces flux output.
<code>as_flux_decrement</code>	New spectral model that produces flux decrement output.
<code>as_optical_depth</code>	New spectral model that produces optical depth output.
<code>continuum</code>	
<code>input_units</code>	This property is used to indicate what units or sets of units the evaluate method expects, and returns a dictionary mapping inputs to units (or <code>None</code> if any units are accepted).
<code>input_units_allow_dimensionless</code>	Allow dimensionless input (and corresponding output).
<code>input_units_equivalencies</code>	
<code>inputs</code>	
<code>is_single_ion</code>	Whether this spectrum represents a collection of single ions or a collection of multiple different ions.
<code>lines</code>	The collection of profiles representing the absorption or emission lines in the spectrum model.
<code>lsf_kernel</code>	
<code>output_type</code>	The data output of this spectral model.
<code>outputs</code>	
<code>redshift</code>	The redshift at which the data given to.
<code>rest_wavelength</code>	
<code>velocity_convention</code>	

Methods Summary

<code>__call__(self, x, *args, **kwargs)</code>	Override the default call function to handle fixed parameters based on the unit of the provided dispersion.
<code>evaluate(self, x, *args, **kwargs)</code>	Evaluate the model on some input variables.
<code>line_stats(self, x)</code>	Calculate statistics over individual line profiles.
<code>region_stats(self, x, rest_wavelength[, ...])</code>	Calculate statistics over arbitrary line regions given some tolerance from the continuum.

Continued on next page

Table 5 – continued from previous page

<code>rejection_criteria(self, x, y[, auto_fit])</code>	Implementation of the Akaike Information Criteria with Correction (AICC) (Akaike 1974; Liddle 2007; King et al.
<code>with_continuum(self, continuum)</code>	New spectral model defined with a different continuum.
<code>with_line(self, *args[, reset])</code>	Add a new line to the spectral model.
<code>with_lines(self, lines[, reset])</code>	Create a new spectral model with the added lines.
<code>with_lsf(self[, kernel])</code>	New spectral model with a line spread function.
<code>with_redshift(self, value)</code>	Generate a new spectral model with the given redshift.

Attributes Documentation

`as_flux`

New spectral model that produces flux output.

`as_flux_decrement`

New spectral model that produces flux decrement output.

`as_optical_depth`

New spectral model that produces optical depth output.

`continuum`

`input_units`

This property is used to indicate what units or sets of units the evaluate method expects, and returns a dictionary mapping inputs to units (or `None` if any units are accepted).

Model sub-classes can also use function annotations in evaluate to indicate valid input units, in which case this property should not be overridden since it will return the input units based on the annotations.

`input_units_allow_dimensionless`

Allow dimensionless input (and corresponding output). If this is `True`, input values to evaluate will gain the units specified in `input_units`. If this is a dictionary then it should map input name to a bool to allow dimensionless numbers for that input. Only has an effect if `input_units` is defined.

`input_units_equivalencies`

`inputs = ('x',)`

`is_single_ion`

Whether this spectrum represents a collection of single ions or a collection of multiple different ions.

Returns

: bool

Is the spectrum composed of a collection of single ions.

`lines`

The collection of profiles representing the absorption or emission lines in the spectrum model.

Returns

: list

A list of Voigt1D models.

lsf_kernel

output_type

The data output of this spectral model. It could one of 'flux', 'flux_decrement', or 'optical_depth'.

Returns

: str

The output type of the model.

outputs = ('y',)

redshift

The redshift at which the data given to.

Returns

: float

The redshift value.

rest_wavelength

velocity_convention

Methods Documentation

__call__(self, x, *args, **kwargs)

Override the default call function to handle fixed parameters based on the unit of the provided dispersion.

Notes

The initial call to this method from fitter will provide unit information on the dispersion. Subsequent calls do not. It is assumed that subsequent calls provide dispersion as km/s.

evaluate(self, x, *args, **kwargs)

Evaluate the model on some input variables.

line_stats(self, x)

Calculate statistics over individual line profiles.

Parameters

x

[Quantity] The input dispersion in either wavelength/frequency or velocity space.

Returns

tab

[QTable] A table detailing the calculated statistics.

region_stats(self, x, rest_wavelength, rel_tol=0.01, abs_tol=1e-05)

Calculate statistics over arbitrary line regions given some tolerance from the continuum.

Parameters**x**

[Quantity] The input dispersion in either wavelength/frequency or velocity space.

rest_wavelength

[Quantity] The rest frame wavelength used in conversions between wavelength/ frequency and velocity space.

rel_tol

[float] The relative tolerance parameter.

abs_tol

[float] The absolute tolerance parameter.

Returns**tab**

[QTable] A table detailing the calculated statistics.

rejection_criteria(*self*, *x*, *y*, *auto_fit=True*)

Implementation of the Akaike Information Criteria with Correction (AICC) (Akaike 1974; Liddle 2007; King et al. 2011). Used to determine whether lines can be safely removed from the compound model without loss of information.

Parameters**x**

[Quantity] The dispersion data.

y

[array-like] The expected flux or tau data.

auto_fit

[bool] Whether the model fit should be re-evaluated for every removed line.

Returns**final_model**

[Spectral1D] The new spectral model with the least complexity.

with_continuum(*self*, *continuum*)

New spectral model defined with a different continuum.

with_line(*self*, **args*, *reset=False*, ***kwargs*)

Add a new line to the spectral model.

Returns**: Spectral1D**

The new spectral model.

with_lines(*self*, *lines*, *reset=False*)

Create a new spectral model with the added lines.

Parameters

lines

[list] List of `OpticalDepth1D` line objects.

Returns

: `Spectral1D`

The new spectral model.

`with_1sf(self, kernel=None, **kwargs)`

New spectral model with a line spread function.

`with_redshift(self, value)`

Generate a new spectral model with the given redshift.

9.3 Fitting

9.4 spectacle.fitting Package

9.4.1 Classes

`CurveFitter()`

`EmceeFitter()`

`LineFinder1D([ions, continuum, defaults, z, ...])`

CurveFitter

class `spectacle.fitting.CurveFitter`

Bases: `astropy.modeling.fitting.LevMarLSQFitter`

Attributes Summary

`uncertainties`

Methods Summary

`__call__(self, *args[, method])`

Fit data to this model.

Attributes Documentation

`uncertainties`

Methods Documentation

`__call__(self, *args, method='curve', **kwargs)`

Fit data to this model.

Parameters

model

[[FittableModel](#)] model to fit to x, y, z

x

[array] input coordinates

y

[array] input coordinates

z

[array (optional)] input coordinates

weights

[array (optional)] Weights for fitting. For data with Gaussian uncertainties, the weights should be 1/sigma.

maxiter

[int] maximum number of iterations

acc

[float] Relative error desired in the approximate solution

epsilon

[float] A suitable step length for the forward-difference approximation of the Jacobian (if `model.fjac=None`). If `epsfcn` is less than the machine precision, it is assumed that the relative errors in the functions are of the order of the machine precision.

estimate_jacobian

[bool] If False (default) and if the model has a `fit_deriv` method, it will be used. Otherwise the Jacobian will be estimated. If True, the Jacobian will be estimated in any case.

equivalencies

[list or None, optional and keyword-only argument] List of *additional* equivalencies that are should be applied in case x, y and/or z have units. Default is None.

Returns

model_copy

[[FittableModel](#)] a copy of the input model with parameters set by the fitter

EmceeFitter

```
class spectacle.fitting.EmceeFitter
```

Bases: [object](#)

Attributes Summary

[errors](#)

[uncertainties](#)

Methods Summary

<code>__call__(self, model, x, y[, yerr, ...])</code>	Call self as a function.
---	--------------------------

Attributes Documentation

errors

uncertainties

Methods Documentation

`__call__(self, model, x, y, yerr=None, nwalkers=500, steps=200, nprocs=1)`
Call self as a function.

LineFinder1D

```
class spectacle.fitting.LineFinder1D(ions=None, continuum=None, defaults=None, z=None,
                                     auto_fit=True, velocity_convention='relativistic', output='flux',
                                     fitter=None, with_rejection=False, fitter_args=None, *args,
                                     **kwargs)
Bases: astropy.modeling.Fittable2DModel
```

Attributes Summary

<code>fitter</code>	
<code>input_units_allow_dimensionless</code>	Allow dimensionless input (and corresponding output).
<code>inputs</code>	
<code>min_distance</code>	
<code>model_result</code>	
<code>outputs</code>	
<code>param_names</code>	
<code>threshold</code>	

Methods Summary

<code>__call__(self, x, *args[, auto_fit])</code>	Evaluate this model using the given input(s) and the parameter values that were specified when the model was instantiated.
<code>evaluate(self, x, y, threshold, ...)</code>	Evaluate the model on some input variables.

Attributes Documentation

fitter

input_units_allow_dimensionless
Allow dimensionless input (and corresponding output). If this is True, input values to evaluate will gain

the units specified in `input_units`. If this is a dictionary then it should map input name to a bool to allow dimensionless numbers for that input. Only has an effect if `input_units` is defined.

```
inputs = ('x', 'y')
```

```
min_distance
```

```
model_result
```

```
outputs = ('y',)
```

```
param_names = ('threshold', 'min_distance')
```

```
threshold
```

Methods Documentation

```
__call__(self, x, *args, auto_fit=None, **kwargs)
```

Evaluate this model using the given input(s) and the parameter values that were specified when the model was instantiated.

```
evaluate(self, x, y, threshold, min_distance, *args, **kwargs)
```

Evaluate the model on some input variables.

9.5 Analysis

9.6 spectacle.analysis Package

9.6.1 Functions

<code>delta_v_90(x, y)</code>	Calculate the dispersion that encompasses the central 90 percent of the apparant optical depth.
<code>equivalent_width(x, y[, continuum])</code>	
<code>full_width_half_max(x, y)</code>	Use a univariate spline to fit the line feature, taking its roots as representative of the full width at half maximum.

`delta_v_90`

`spectacle.analysis.delta_v_90(x, y)`

Calculate the dispersion that encompasses the central 90 percent of the apparant optical depth. Follows the formulation defined in Prochaska & Wolf (1997).

Parameters

x

[Quantity] The dispersion axis. Can be either wavelength or velocity space.

y
[array-like] Flux or optical depth array. Note that the calculation assumes that the data is optical depth. If providing flux, it will be converted to apparant optical depth.

equivalent_width

`spectacle.analysis.equivalent_width(x, y, continuum=None)`

full_width_half_max

`spectacle.analysis.full_width_half_max(x, y)`

Use a univariate spline to fit the line feature, taking its roots as representative of the full width at half maximum.

Parameters

x
[`astropy.units.Quantity`] The dispersion array.

y
[`np.ndarray`] The data array.

Returns

float
The full width at half maximum.

9.6.2 Classes

<code>Resample(new_dispersion)</code>	Resample model which can be used with compound model objects.
---------------------------------------	---

Resample

class `spectacle.analysis.Resample(new_dispersion)`

Bases: `object`

Resample model which can be used with compound model objects.

Methods Summary

<code>__call__(self, x, y)</code>	Call self as a function.
-----------------------------------	--------------------------

Methods Documentation

`__call__(self, x, y)`
Call self as a function.

9.7 Registries

9.8 spectacle.registries Package

9.8.1 Classes

`LineRegistry(*args, **kwargs)`

LineRegistry

```
class spectacle.registries.LineRegistry(*args, **kwargs)
    Bases: astropy.table.QTable
```

Methods Summary

`correct(self, name)`

`load(self, path)`

`load_default(self)`

`subset(self, ions)`

`with_lambda(self, lambda_0)`

`with_name(self, name)`

Methods Documentation

correct(*self*, *name*)

load(*self*, *path*)

load_default(*self*)

subset(*self*, *ions*)

with_lambda(*self*, *lambda_0*)

with_name(*self*, *name*)

Part IV

Indices and tables

- [genindex](#)
- [modindex](#)
- [search](#)

PYTHON MODULE INDEX

S

spectacle.analysis, [51](#)
spectacle.fitting, [48](#)
spectacle.modeling, [41](#)
spectacle.registries, [53](#)

Symbols

__call__() (*spectacle.analysis.Resample* method), 52
 __call__() (*spectacle.fitting.CurveFitter* method), 48
 __call__() (*spectacle.fitting.EmceeFitter* method), 50
 __call__() (*spectacle.fitting.LineFinder1D* method), 51
 __call__() (*spectacle.modeling.OpticalDepth1D* method), 43
 __call__() (*spectacle.modeling.Spectral1D* method), 46

A

as_flux (*spectacle.modeling.Spectral1D* attribute), 45
 as_flux_decrement (*spectacle.modeling.Spectral1D* attribute), 45
 as_optical_depth (*spectacle.modeling.Spectral1D* attribute), 45

C

column_density (*spectacle.modeling.OpticalDepth1D* attribute), 42
 continuum (*spectacle.modeling.Spectral1D* attribute), 45
 correct() (*spectacle.registries.LineRegistry* method), 53
 CurveFitter (*class in spectacle.fitting*), 48

D

delta_lambda (*spectacle.modeling.OpticalDepth1D* attribute), 42
 delta_v (*spectacle.modeling.OpticalDepth1D* attribute), 42
 delta_v_90() (*in module spectacle.analysis*), 51

E

EmceeFitter (*class in spectacle.fitting*), 49
 equivalent_width() (*in module spectacle.analysis*), 52
 errors (*spectacle.fitting.EmceeFitter* attribute), 50
 evaluate() (*spectacle.fitting.LineFinder1D* method), 51
 evaluate() (*spectacle.modeling.OpticalDepth1D* method), 43
 evaluate() (*spectacle.modeling.Spectral1D* method), 46

F

f_value (*spectacle.modeling.OpticalDepth1D* attribute), 42

fit_deriv() (*spectacle.modeling.OpticalDepth1D* static method), 43
 fitter (*spectacle.fitting.LineFinder1D* attribute), 50
 full_width_half_max() (*in module spectacle.analysis*), 52
 full_width_half_max() (*spectacle.modeling.OpticalDepth1D* method), 43

G

gamma (*spectacle.modeling.OpticalDepth1D* attribute), 43

I

input_units (*spectacle.modeling.OpticalDepth1D* attribute), 43
 input_units (*spectacle.modeling.Spectral1D* attribute), 45
 input_units_allow_dimensionless (*spectacle.fitting.LineFinder1D* attribute), 50
 input_units_allow_dimensionless (*spectacle.modeling.Spectral1D* attribute), 45
 input_units_equivalencies (*spectacle.modeling.Spectral1D* attribute), 45
 inputs (*spectacle.fitting.LineFinder1D* attribute), 51
 inputs (*spectacle.modeling.OpticalDepth1D* attribute), 43
 inputs (*spectacle.modeling.Spectral1D* attribute), 45
 is_single_ion (*spectacle.modeling.Spectral1D* attribute), 45

L

lambda_0 (*spectacle.modeling.OpticalDepth1D* attribute), 43
 line_stats() (*spectacle.modeling.Spectral1D* method), 46
 LineFinder1D (*class in spectacle.fitting*), 50
 LineRegistry (*class in spectacle.registries*), 53
 lines (*spectacle.modeling.Spectral1D* attribute), 45
 load() (*spectacle.registries.LineRegistry* method), 53
 load_default() (*spectacle.registries.LineRegistry* method), 53

lsf_kernel (*spectacle.modeling.Spectral1D* attribute), 46

M

min_distance (*spectacle.fitting.LineFinder1D* attribute), 51

model_result (*spectacle.fitting.LineFinder1D* attribute), 51

O

OpticalDepth1D (*class in spectacle.modeling*), 41

output_type (*spectacle.modeling.Spectral1D* attribute), 46

outputs (*spectacle.fitting.LineFinder1D* attribute), 51

outputs (*spectacle.modeling.OpticalDepth1D* attribute), 43

outputs (*spectacle.modeling.Spectral1D* attribute), 46

P

param_names (*spectacle.fitting.LineFinder1D* attribute), 51

param_names (*spectacle.modeling.OpticalDepth1D* attribute), 43

R

redshift (*spectacle.modeling.Spectral1D* attribute), 46

region_stats() (*spectacle.modeling.Spectral1D* method), 46

rejection_criteria() (*spectacle.modeling.Spectral1D* method), 47

Resample (*class in spectacle.analysis*), 52

rest_wavelength (*spectacle.modeling.Spectral1D* attribute), 46

S

spectacle.analysis (*module*), 51

spectacle.fitting (*module*), 48

spectacle.modeling (*module*), 41

spectacle.registries (*module*), 53

Spectral1D (*class in spectacle.modeling*), 43

subset() (*spectacle.registries.LineRegistry* method), 53

T

threshold (*spectacle.fitting.LineFinder1D* attribute), 51

U

uncertainties (*spectacle.fitting.CurveFitter* attribute), 48

uncertainties (*spectacle.fitting.EmceeFitter* attribute), 50

V

v_doppler (*spectacle.modeling.OpticalDepth1D* attribute), 43

velocity_convention (*spectacle.modeling.Spectral1D* attribute), 46

voigt() (*spectacle.modeling.OpticalDepth1D* static method), 43

W

with_continuum() (*spectacle.modeling.Spectral1D* method), 47

with_lambda() (*spectacle.registries.LineRegistry* method), 53

with_line() (*spectacle.modeling.Spectral1D* method), 47

with_lines() (*spectacle.modeling.Spectral1D* method), 47

with_lsf() (*spectacle.modeling.Spectral1D* method), 48

with_name() (*spectacle.registries.LineRegistry* method), 53

with_redshift() (*spectacle.modeling.Spectral1D* method), 48