I Installation and Setup
1 Installation 5
2 Launching SpecViz 9

II Using SpecViz
3 Getting Started 13
4 Unit Conversion 17
5 Arithmetic Widget 21
6 Loader Wizard 25
7 Model Fitting 27
8 Spectrum Statistics 29
SpecViz is a tool for visualization and quick-look analysis of 1D astronomical spectra. It is written in the Python programming language, and therefore can be run anywhere Python is supported (see Installation). SpecViz is built on top of the SpecUtils Astropy-affiliated python library, providing a visual, interactive interface to the analysis capabilities in that library.

SpecViz allows spectra to be easily plotted and examined. It supports flexible spectral unit conversions, custom plotting attributes, interactive selections, multiple plots, and other features.

SpecViz notably includes a measurement tool for spectral lines which enables the user, with a few mouse actions, to perform and record measurements. It has a model fitting capability that enables the user to create simple (e.g., single Gaussian) or multi-component models (e.g., multiple Gaussians for emission and absorption lines in addition to regions of flat continua). A typical data-analysis workflow might involve data exploration using SpecViz and then scripting to create more complex measurements or modeling workflows using SpecUtils.

SpecViz will soon include the ability to:

- Measure the average of multiple spectra, detrending, and apply Fourier filters.
- Interactively renormalize data from spectral templates.
- And more…
Part I

Installation and Setup
As a Python package, SpecViz is installable via any approach that is available for installing Python packages. In practice the easiest way is often using the Anaconda package manager, but several other options are available. These are detailed below distributed through the Anaconda package manager. Specifically, it lives within Space Telescope Science Institute's AstroConda channel.

### 1.1 Install via Anaconda

If you do not have Anaconda, please follow the instructions here to install it, or scroll down for manual installation of SpecViz.

If you are using the Space Telescope Science Institute’s AstroConda channel, then all you have to do to install SpecViz is simply type the following at any Bash terminal prompt:

```bash
c$ conda install specviz
```

If you do not have AstroConda installed, you can still install SpecViz from AstroConda by specifying the channel in your install command:

```bash
c$ conda install --channel http://ssb.stsci.edu/astroconda specviz
```

At this point, you’re done! You can launch SpecViz by typing the following at any terminal:

```bash
c$ specviz
```

### 1.1.1 Uninstalling

To uninstall via Anaconda, simply type the following at a command line:
## 1.2 Install via source

SpecViz can also be installed manually using the source code followint the instructions below. The dependencies are listed in the `setup.cfg` file, and therefore most of them will be handled automatically by the setup functions, the exception to this is the exception of PyQt, which may require manual installation.

### 1.2.1 By using pip

Clone the SpecViz repository somewhere on your system, and install locally using `pip`. If you are using an Anaconda virtual environment, please be sure to activate it first before installing:

```bash
$ source activate <environment_name>
```

```bash
$ pip install git+http://github.com/spacetelescope/specviz.git@v0.4.4
```

This uses the pip installation system, so please note that

1. You need to have pip installed (included in most Python installations).
2. You do not need to run `python setup.py install`.
3. You do not need to install the dependencies by hand (except for PyQt).

Likewise, the pip command will use your default Python to install. You can specify by using `pip2` or `pip3`, if you’re not using a virtual environment.

### 1.2.2 By cloning

You may also install by cloning the repository directly

```bash
$ git clone https://github.com/spacetelescope/specviz.git
$ cd specviz
$ git checkout tags/v0.3.0
$ python setup.py install
```

### 1.2.3 PyQt bindings

SpecViz requires PyQt. Currently, only python environments with 3.5 or higher installed can use pip to install PyQt5, in which case simply type:

```bash
$ pip install pyqt5
```

to install it on your system.

In any other case, PyQt can be installed via anaconda:

```bash
$ conda install pyqt
```
1.2.4 Uninstalling

To uninstall via pip, simply type the following at a command line:

```
$ pip uninstall specviz
```

1.3 Known Issues

On a Mac with Qt5, depending on exactly how you have set up Anaconda, you might see the following error after following the above instructions:

```
This application failed to start because it could not find or load the Qt platform plugin “cocoa”.
Reinstalling the application may fix this problem.
```

If you see this message, you have encountered an incompatibility between Anaconda’s packaging of Qt4 and Qt5. The workaround is to uninstall Qt4 with the following command:

```
$ conda uninstall pyqt qt
```

and SpecViz should now happily run.

Conversely, if you’ve had PyQt5 installed previously and you wish to run the PyQt4 version, you may run into a similar error:

```
$ RuntimeError: the PyQt4.QtCore and PyQt5.QtCore modules both wrap the QObject class
```

This issue can be solved with the following command:

```
$ conda uninstall pyqt5 qt5
```
Launching SpecViz

Once the user has installed SpecViz, they can launch it via the command line:

```
$ specviz
```

If the user wishes to inspect a single file, they can also pass in the filename as a command line argument along with the `-F` flag as follows:

```
$ specviz -F <filename>
```

In the above case, the loader registry functionality will attempt to select the best available loader for the data. If the user wishes to specify a specific loader for their data, a `-L` flag can be passed with the loader name

```
$ specviz -F <filename> -L <loader_name>
```

To get further help with the command line options, simply type `specviz --help`. 

Part II

Using SpecViz
3.1 Loading a Basic Spectrum

Let's start by loading a simple, known spectrum. If you followed the steps in *Launching SpecViz*, you should see the specviz window up, but without a spectrum.

There are two options to load a spectrum data file:

1. Load the spectrum with a pre-defined data loader, or a custom user-defined data loader.
2. Utilize the *Loader Wizard* to create a new custom loader, and use that to load your data.

Either way, the specific loader for your data can be selected from the drop down list in the open file dialog.
Once the file has been loaded, you will be greeted with a plotted spectrum.

### 3.2 Exporting Spectra

A user can export a given spectrum in the data list by highlighting the spectrum and clicking the Export Data button in the main toolbar. This will provide the user with a save file dialog where they may choose where to save the exported spectrum file.

**Note:** ECSV is currently the only supported export format. This will change in the future as more exporting formats are supported in the specutils package.

### 3.3 Workspaces and Plots

Loaded data is added to the global data list for the entire workspace. A workspace is an instance of the SpecViz application. Several can be open at once and more can be added by selecting File > New Workspace from the menu bar. Workspaces are completely independent of one another, and the user can have as many workspaces as they wish.

Plots are the visual representation of a selection of data items within a particular workspace. A user can have as many plots as they wish, and new plots can be created by clicking the New Plot button in the main toolbar. Likewise, plot windows can be removed by clicking the X button in the plot tab.

Plots all share the same data list within a workspace. The caveat is that plots can only show data that have equivalent units. Any data that cannot be added to a plot will show as disabled in the data list.

Clicking the checkbox next to an item in the data list will plot the data item on the current plot.
3.4 Using Regions

Interesting regions can be defined by adding an ROI to the plot. This is done via the Add Region button in the plot toolbar. Multiple regions can be used when defining e.g. areas the user might wish to use for model fitting.

A blue highlighted region is considered the _active_ region, and will in cases where it is imported (e.g. statistics), will be used to define the region used in the calculations.

**Note:** Currently, only one region can be the “selected” region. This will change in the future to allow a subset of plotted region to be defined as “selected”.

Removing a region is accomplished by clicking the Remove Region button. Only the currently selected region is removed.
4.1 How Unit Conversion Works

Unit Conversion uses the astropy.units module in order to convert spectral axis and flux units.

```python
import astropy.units as u

potential_X_axis_units = u.Unit(original_spectral_axis_units).find_equivalent_units(equivalencies=u.spectral())

potential_Y_axis_units = u.Unit(original_flux_units).find_equivalent_units(equivalencies=u.spectral_density(spectral_axis_element))
```

The GUI comboboxes are then populated by potential_X_axis_units and potential_Y_axis_units, respectively. The user can then select any of those options - as well as a “Custom” option - and the changes will be reflected in the plot. If the user selects the “Custom” option, they can type in their own units, and if the units are accepted by astropy.units the unit conversion dialog box will allow the units to be parsed. However, “Custom” units are checked to see if the conversion is allowed before any conversion occurs and if False, the units will not be changed.
4.1. How Unit Conversion Works
The Arithmetic widget allows users to perform basic arithmetic on their spectra that are loaded into Specviz. The arithmetic editor accepts data loaded into the Specviz session and only outputs type `specutils.Spectrum1D`. This means the editor allows users to perform the same arithmatic operations that are available with `specutils.Spectrum1D` objects.

### 5.1 Launching Arithmetic Editor

Lets start by loading the arithmetic layer widget located on the tool bar. Upon clicking you will be prompted with the arithmetic dialog.
From this dialog you can add, edit or remove arithmetic items from the editor. We will start by clicking the New Arithmetic Attribute button located in top left hand corner of the Editor dialog. Upon clicking you will be prompted with the editor dialog.
5.2 Adding Arithmetic

Once the Arithmetic widget is launched, spectra and their components can be added here by typing the names directly surrounded by ‘{}’ or by selecting the spectrum in the dropdown bar and clicking insert. We are going to take the preloaded spectrum (science_spectrum) and create a new spectrum that is double the flux of science_spectrum and call it double science_spectrum.

To validate arithmetic, click the OK button located at the bottom right hand corner of the dialog box.

**Warning:** If the python syntax is invalid, the editor will not allow you to continue! The editor can only return SpecUtils Spectrum1D objects.

Now, there will be a new data item located in the data collection called double science_spectrum.
To show the result in the plotting window, select the data item by clicking the box next double science_spectrum.

This is a very simple example of the arithmetic you can perform with the SpecViz arithmetic editor. There are more example expressions located at the bottom of the equation editor dialog.
The loader wizard will assist users in generating a custom data loader. After working through the wizard a new loader python file can be saved to a directory of the user’s choice (default is the ~/.specutils directory). The written loader will then be automatically added to the specutils loader registry and accessible from the file open dialog.
The top field of the wizard requires additional parameters that may be needed by `astropy.table.Table.read()` to open the data file. The syntax of this field should match the syntax following the filename parameter in an `astropy.table.Table.read()` call. For example, the grayed out text in the above figure shows an additional parameter of `format="ascii"`. For multiple additional parameters you should include commas, i.e. `format = "ascii", comment = "*"`. See the `astropy.table.Table` documentation for the available options. Once this field is filled out, you can press the Refresh Data button and the wizard will reload the table from the provided file. If the `read` command was successful, the remaining sections of the wizard will be populated.

The bottom left field for the Loader Name determines the label assigned to the saved loader. When the Save to .py button is clicked, it will open a new dialog for saving the loader file on your machine. The loader label name and the filename can be different.

You will not be able to save your new loader without valid units defined for the data section and the dispersion section. You can see a preview of the generated loader wizard by clicking on the Preview .py button.

The wizard is only able to handle table data, and will not properly handle FITS image data. Additionally, although not currently implemented, we plan on supporting bit masks in a future release.
SpecViz utilizes Astropy Models and SpecUtils Fitting to fit models to its spectra. For example, you can fit one model to the continuum, another to an emission line of interest, and yet another to an absorption line.

Currently, the following models are available:

<table>
<thead>
<tr>
<th>SpecViz Model Name</th>
<th>Astropy Model Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Const</td>
<td>Const1D</td>
</tr>
<tr>
<td>Gaussian</td>
<td>Gaussian1D</td>
</tr>
<tr>
<td>Linear</td>
<td>Linear1D</td>
</tr>
<tr>
<td>Lorentz</td>
<td>Lorentz1D</td>
</tr>
<tr>
<td>Voigt</td>
<td>Voigt1D</td>
</tr>
<tr>
<td>Polynomial</td>
<td>Polynomial1D</td>
</tr>
</tbody>
</table>

The models can be fitted with the following fitters:

<table>
<thead>
<tr>
<th>SpecViz Fitter Name</th>
<th>Astropy Fitter Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Levenberg-Marquardt</td>
<td>LevMarLSQFitter</td>
</tr>
<tr>
<td>Simplex</td>
<td>SimplexLSQFitter</td>
</tr>
</tbody>
</table>

To use a model:

1. Create a new model data item by clicking on the New Model button on the WorkSpace toolbar.
2. Select the layer you wish to operate on from the combination box at the bottom of the fitting window. For example, you can choose the layer containing your emission or absorption line.
3. Create and position a region of interest (ROI). Multiple ROIs can be used. SpecViz fits the data under all the ROIs.
4. Select the desired model from the green Add Model drop-down box to add it to Current Models.
5. If desired, repeat the above step to add additional models.

To edit model parameters or enter a better first estimate of the model parameters:
1. If desired, double-click on the model name to rename it. When you see a blinking cursor, enter its new name and press “Enter”.
2. Expand the model listing under the model name.
3. Double-click on the desired model parameter value in the listing. When you see a blinking cursor, enter the new value and press Enter.

To fit a model:
1. Select the layer you wish to operate on from the combination box at the bottom of the fitting window.
2. Adjust model parameter values to approximate fit.
3. Click the lock icon next to any parameter to choose whether it should be kept fixed (closed lock) or allowed to vary (open lock) during fitting.
4. Click on the settings icon at the bottom of the model fitting window select options such as the desired fitter and maximum iterations.
5. Check the model’s Equation Editor by clicking the calculator button. It will pop up with the current model arithmetic. Review, edit and press OK when done.
6. Click the blue Fit Model button at the bottom of the fitting window.
7. The associated model parameters will be adjusted accordingly.

### 7.1 Equation Editor

The Equation Editor text box is used to define the relationship between different models for the same model data item. The editor can be launched by clicking the calculator button at the bottom of the model fitting window. If nothing is defined, the default is to add all the models together. To describe a non-default model relationship, adjust the math operators, as shown in the examples below and then press Enter to produce the compound model:

<table>
<thead>
<tr>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear1 + Gaussian1</td>
</tr>
<tr>
<td>Linear1 * Gaussian1</td>
</tr>
<tr>
<td>Gaussian1 - Gaussian2</td>
</tr>
</tbody>
</table>

The entity that results from lumping together all the models, and combining them either using the arithmetic behavior expression, or just adding them all together, is called a “compound model”.

### 7.2 Model names

When added to the Current Models list, a model will receive a default name that is generated from the model type (as listed in the drop down model selector) plus a running numerical suffix.

These names can be changed by clicking on the default name and entering a new name. Note that changing model names will require that any expression in the Equation Editor text box be edited accordingly.
11 statistic/analysis functions are calculated using the input spectrum or specified region of interest, depending on what is selected in the left side bar. If a region of interest is selected, the statistic calculations are updated when that region of interest is changed.

Calculations are done using the following functions:

- **Mean**
  ```python
  astropy.units.Quantity.mean
  ```

- **Median**
  ```python
  numpy.median
  ```

- **Std Dev**
  ```python
  astropy.units.Quantity.std
  ```

- **Centroid**
  ```python
  specutils.analysis.centroid()
  ```

- **RMS**
  ```python
  numpy.sqrt(flux.dot(flux) / len(flux))
  ```

- **SNR**
  ```python
  specutils.analysis.snr()
  ```

- **FWHM**
  ```python
  specutils.analysis.fwhm()
  ```

- **Eq Width**
  ```python
  specutils.analysis.equivalent_width()
  ```

- **Max**
  ```python
  astropy.units.quantity.Quantity.max
  ```

- **Min**
  ```python
  astropy.units.quantity.Quantity.min
  ```

- **Count Total**
  ```python
  specutils.analysis.line_flux()
  ```