specdal Documentation

Release 2.0.0

Young Lee

Mar 21, 2018

Contents:

CHAPTER 1

Introduction

SpecDAL is a Python package for loading and manipulating field spectroscopy data. It currently supports readers for ASD, SVC, and PSR spectrometers. SpecDAL provides useful functions and command line scripts for processing and aggregating the data.

1.1 Interface

There are three options for using SpecDAL.

1. Python interface

   The lowest level interface is for users to import `specdal` as a Python module. Functions in `specdal` are written to operate directly on Pandas Series and DataFrames. `specdal` also provides classes that wrap around Pandas objects for convenience to users not familiar with Pandas.

   Users at this level are encouraged to check out the *data model*, Notebook examples , and the *API*.

2. Command line interface

   Alternatively, users can utilize the command line scripts that `specdal` provides. The following scripts are currently distributed:

   - *specdal_info*: displays key information in a spectral file
   - *specdal_pipeline*: converts a directory of spectral files into .csv files and figures

3. Graphical User Interface (GUI)

   At the highest level, `SpecDAL` provides a GUI that requires no programming. GUI can be handy for tasks such as outlier detection. GUI is provided as an executable, `specdal_gui` on Linux/Mac and `specdal_gui.exe` on Windows.

1.2 Examples

Check out the example Notebooks here.

CHAPTER 2

Installation

SpecDAL is available via pip (`pip install specdal`) or on Github. This page provides detailed walkthrough of the installation process intended for users who are not comfortable in Python environment.

2.1 Prerequisites

- python3
- pip3

2.1.1 Setting up the virtual environment (recommended)

Although not necessary, it is good practice to install Python packages in a virtual environment. Virtual environments provide an isolated and self-contained environment for your Python session, which can help prevent conflicts across packages. We will walk through the process of creating one on Ubuntu Linux for demonstration.

- Install virtualenv using pip installer.

```
$ pip install --user virtualenv
```

- Create a directory for storing virtual environments.

```
$ mkdir ~/venv
```

- Create a new virtual environment called `specdal_env` running python3 by default.

```
$ virtualenv -p python3 ~/venv/specdal_env
```

  If you're curious, you can navigate to that directory and find all the components that make up a Python environment. For example, packages are installed in `~/venv/specdal_env/lib` and binaries are stored in `~/venv/specdal_env/bin`.

- Before starting a Python session, we can activate the virtual environment as follows.

```
$ source ~/venv/specdal/bin/activate
```

Note: On windows, there should be an executable `~/venv/specdal/bin/activate.exe` with a similar effect.

You'll notice the name of your virtual environment in parentheses.

```
(specdal_env) $
```

- Once in this environment, we can install and use `SpecDAL` or other packages.

```
(specdal_env) $ ... # install specdal
(specdal_env) $ ... # write and run programs
```

- When we're done, we can exit the virtual environment.

```
$ deactivate
```

2.2 Install via pip

- Stable version

```
$ pip3 install specdal --upgrade
```

- Latest development version

```
$ pip3 install specdal --pre
```

2.3 Install from Github

SpecDAL can be found on Enspec's Github repo. Stable release can be found on `master` branch and the development version on `dev` branch.

2.3.1 Github walkthrough

1. Open terminal or Git-bash and navigate to the desired directory, `~/specdal` for this demo.

   `cd ~/specdal`

2. The following command will clone the SpecDAL's Github repository.

   ```
   $ git clone https://github.com/EnSpec/SpecDAL.git
   ```

   You'll notice a new subdirectory `SpecDAL` with the source code.

3. Install SpecDAL.

   ```
   $ cd ./SpecDAL
   $ python setup.py install
   ```

2.3.2 Install in development mode

If you'd like to modify SpecDAL's source, it's useful to install the package in development mode.

- Install in development mode

```
$ python setup.py develop
```

- Modify the source and run/test it.
- Uninstall development mode

```
$ python setup.py develop --uninstall
```

CHAPTER 3

Data Model

SpecDAL relies on Pandas data structures to represent spectroscopy measurements. A single measurement is stored in pandas.Series while a collection of measurements is stored in pandas.DataFrame. SpecDAL provides Spectrum and Collection classes that wraps Series and DataFrames along with spectral metadata. Spectral operators, such as interpolation, are provided as functions on pandas objects or as methods of specdal's classes.

3.1 Pandas Representation of Spectra

3.1.1 Series - single spectrum

3.1.2 DataFrame - collection of spectra

3.2 Spectrum and Collection Classes

3.2.1 Spectrum - single spectrum

3.2.2 Collection - collection of spectra

3.3 Operators

CHAPTER 4

API Reference

This is the class and function reference page of SpecDAL.

4.1 Operators

Specdal's operators perform on both pandas and specdal objects. In the following operations, pandas series and dataframes correspond to specdal's spectrum and collection, respectively (except get_column_types - TODO: move this function to utils module).

specdal.operator.**derivative**(*series*)
Calculate the spectral derivative. Not Implemented Yet.

specdal.operator.**get_column_types**(*df*)
Returns a tuple (wvl_cols, meta_cols), given a dataframe.

Notes

Wavelength column is defined as columns with a numerical name (i.e. decimal). Everything else is considered metadata column.

specdal.operator.**interpolate**(*series*, *spacing=1*, *method='slinear'*)
Interpolate the array into given spacing

Parameters **series: pandas.Series object**

specdal.operator.**jump_correct**(*series*, *splices*, *reference*, *method='additive'*)
Correct for jumps in non-overlapping wavelengths

Parameters **splices: list**

list of wavelength values where jumps occur

reference: int

position of the reference band (0-based)

specdal.operator.**jump_correct_additive**(*series*, *splices*, *reference*)
Perform additive jump correction (ASD)

specdal.operator.**proximal_join**(*base_df*, *rover_df*, *on='gps_time_tgt'*, *direction='nearest'*)
Perform proximal join and return a new dataframe.

Returns proximal: pandas.DataFrame object

proximally processed dataset (rover_df / base_df)

Notes

As a side-effect, the rover dataframe is sorted by the key Both base_df and rover_df must have the column specified by on. This column must be the same type in base and rover.

specdal.operator.**stitch**(*series*, *method='max'*)
Stitch the regions with overlapping wavelength

Parameters **series: pandas.Series object**

4.2 Spectrum

class specdal.spectrum.**Spectrum**(*name=None*, *filepath=None*, *measurement=None*, *measure_type='pct_reflect'*, *metadata=None*, *interpolated=False*, *stitched=False*, *jump_corrected=False*, *verbose=False*)
Class that represents a single spectrum

Parameters **name: string**

Name of the spectrum.

filepath: string (optional)

Path to the file to read from.

measurement: pandas.Series

Spectral measurement

metadata: OrderedDict

Metadata associated with spectrum

Notes

Spectrum object stores a single spectral measurement using pandas.Series with index named: "wavelength".

Methods

interpolate(*spacing=1*, *method='slinear'*)

jump_correct(*splices*, *reference*, *method='additive'*)

read(*filepath*, *measure_type*, *verbose=False*)
Read measurement from a file.

stitch(*method='mean'*)

4.3 Collection

class specdal.collection.**Collection**(*name*, *directory=None*, *spectra=None*, *measure_type='pct_reflect'*, *metadata=None*, *flags=None*)

Represents a dataset consisting of a collection of spectra

Attributes

Methods

append(*spectrum*)
insert spectrum to the collection

data
Get measurements as a Pandas.DataFrame

data_with_meta(*data=True*, *fields=None*)
Get dataframe with additional columns for metadata fields

Parameters **data: boolean**

whether to return the measurement data or not

fields: list

names of metadata fields to include as columns. If None, all the metadata will be included.

Returns pd.DataFrame: self.data with additional columns

flags
A dict of flags for each spectrum in the collection

groupby(*separator*, *indices*, *filler=None*)
Group the spectra using a separator pattern

Returns OrderedDict consisting of specdal.Collection objects for each group

key: group name value: collection object

interpolate(*spacing=1*, *method='slinear'*)

jump_correct(*splices*, *reference*, *method='additive'*)

max(*append=False*)

mean(*append=False*)

median(*append=False*)

min(*append=False*)

plot(**args*, ***kwargs*)

read(*directory, measure_type='pct_reflect', ext=['.asd', '.sed', '.sig', '.pico', '.light'], recursive=False, verbose=False*)
read all files in a path matching extension

spectra
A list of Spectrum objects in the collection

std(*append=False*)

stitch(*method='max'*)

to_csv(**args*, ***kwargs*)

CHAPTER 5

Specdal Pipeline Script

Specdal provides a command line script ‘’specdal_pipeline” for batch processing of spectral data files in a directory. A typical input to ‘’specdal_pipeline” is a directory containing spectral files (i.e. .asd files), which will be converted into .csv files and figures of spectra. User can provide arguments to customize the processing operations (i.e. jump correction, groupby) and output (i.e. .csv file of group means). This page describes the usage and provides examples.

5.1 Usage

5.2 Example

CHAPTER 6

Specdal Info Script

6.1 Usage

6.2 Example

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index