SED Fitter Documentation

Release 1.2

Thomas Robitaille

Jun 16, 2018

Contents

This package is an experimental Python port of the SED Fitting tool described in Robitaille et al. (2007) and is still under active development.

Important: **No guarantees are made** about the accuracy of the results, and users are responsible for ensuring that the results are reasonable and that they understand all the limitations inherent to SED fitting.

Please report any issues with the package here (not by email).

CHAPTER 1

User documentation

1.1 Introduction

The concept of the SED fitter is very simple: consider a set of sources to be studied. For each source, the SED fitter can fit models, such as model stellar photospheres, YSO model SEDs, as well as galaxy and AGB templates, to the multi-wavelength photometry measurements of this particular source using linear regression.

The scale factor S (which is related to the luminosity and distance of the sources) and the extinction $A_{\rm V}$ are used as free parameters in the fitting process. The result for any given source is a value for the goodness of fit and best-fit values S and $A_{\rm V}$ for every single model. These fits can then be analyzed to derive properties of the source.

To quantify the goodness/badness of each fit to each source, we calculate the χ^2 value:

$$\chi^2 = \sum_{i=1}^{N} \left(\frac{\langle \log_{10}[F_\nu(\lambda_i)] \rangle - \log_{10}[M_\nu(\lambda_i)]}{\sigma(\langle \log_{10}[F_\nu](\lambda_i) \rangle)} \right)^2$$

where $\langle \log_{10}[F_\nu(\lambda_i)] \rangle$ are the mean fluxes in log space, flux values at a given wavelength λ_i, $\sigma(\langle \log_{10}[F_\nu](\lambda_i) \rangle)$ are the flux uncertainties in log space, and $\log_{10}[M_\nu(\lambda_i)]$ are the extincted and scaled model log fluxes. For more details, see Robitaille et al (2007). In this manual we sometimes refer to the χ^2 per datapoint, $\chi^2/n_{\rm data}$, where the number of datapoints does *not* include upper and lower limits.

The SED fitter uses the concept of *model packages*, which are single directories containing SEDs, convolved fluxes, parameters, and a description of the models, all in a common format. There are two kinds of models that can be used with the SED fitter:

- Models for which the absolute distance cannot be determined from the fit (e.g. unscaled stellar photosphere models). This is usually used for the purpose of filtering out a certain class of sources, for example foreground/background stars.
- Models that are absolutely scaled in flux, to a distance of 1kpc. In this case, it is also possible to specify fluxes as a function of aperture - in this case, the fitting procedure is more complex as it involves computing the aperture-dependent SEDs for a fine grid of distances, and optionally removing models that would clearly be extended relative to the aperture chosen. This is described in more detail in Robitaille et al (2007).

1.2 Installation

1.2.1 Requirements

The SED fitter requires Python 2.6, 2.7, 3.2, or 3.3, and the following Python packages to be installed:

- Numpy
- Scipy
- Matplotlib
- Astropy 0.3 or later

First-time Python users may want to consider an all-in-one Python installation package, such as the Anaconda Python Distribution which provides all of the above dependencies.

1.2.2 Installation

You can install the SED fitting package using:

```
pip install sedfitter
```

or you can get the latest tar file from PyPI.

If you want to install the latest developer version, you will need to clone the git repository:

```
git clone http://github.com/astrofrog/sedfitter
```

You can then install the package with:

```
cd sedfitter
python setup.py install
```

1.2.3 Testing

If you want to check that all the tests are running correctly with your Python configuration, you can also run:

```
python setup.py test
```

in the source directory. If there are no errors, you are good to go!

1.3 Fitting SEDs to sources

1.3.1 Typical workflow

Once the data has been prepared following the description in *Data format*, and that you have a set of models in the format described in *Creating model packages* the typical workflow to follow when using the SED fitting tool is:

- Fit a given set of models to the data
- Optionally split the output between well- and badly-fit sources
- Make plots of the SED fits, or of the parameter distribution

Each of these steps can be performed by one ore more functions in the SED Fitting tool, which are described below in detail:

1.3.2 Note on quantities and units

The Python SED fitting tool makes use of the `astropy.units` package to handle units and unit conversions. Several of the options that need to be specified in the functions described below require `Quantity` instances. Defining quantities is straightforward:

```
from astropy import units as u

# Define scalar quantity
q1 = 3. * u.kpc

# Define array quantity using a list
q2 = [1., 2., 3.] * u.arcsec

# Define array quantity using a Numpy array
q3 = np.array([1., 2., 3.]) * u.cm ** 2 / u.g
```

1.3.3 Fitting the models to the data

In order to fit the data, you will need to make use of the *`fit()`* function, which can be imported using:

```
from sedfitter import fit
```

At a minimum, you will need to call the function with the following input:

- The file containing the data (see *Data format*).
- The list of filters and aperture radii for which the data is given (see *Common filter names*).
- The directory containing the models (see *Available Model Packages* and *Creating model packages*).
- The name of the output file
- The range of Av and distances to assume for the fits
- An extinction law, given as a opacities to extinction (in cm^2/g or equivalent) versus wavelength.

The *`fit()`* page describes in detail how each of these inputs should be specified, and also lists additional options to further fine-tune the fitting and the output.

The following is a complete example showing how to use the *`fit()`* function:

```
from astropy import units as u
from sedfitter import fit
from sedfitter.extinction import Extinction

# Define path to models
model_dir = '/Volumes/Data/models/models_r06'

# Read in extinction law)
extinction = Extinction.from_file('kmh94.par', columns=[0, 3],
                                  wav_unit=u.micron, chi_unit=u.cm**2 / u.g)

# Define filters and apertures
```

(continues on next page)

(continued from previous page)

```
filters = ['2J', '2H', '2K', 'I1', 'I2', 'I3', 'I4']
apertures = [3., 3., 3., 3., 3., 3., 3.] * u.arcsec

# Run the fitting
fit('data_glimpse', filters, apertures, model_dir,
    'output.fitinfo',
    extinction_law=extinction,
    distance_range=[1., 2.] * u.kpc,
    av_range=[0., 40.])
```

Note: in the filter list, you can also specify wavelengths as Astropy `Quantity` instances. If you do this, the SED wavelength closest to that specified will be used in the fitting.

Note: if you do not specify the columns and units when reading in the extinction, the first two columns are read and are assumed to be in c.g.s.. If you have previously used the Fortran version of the SED fitter, you will need to specify `columns=[0, 3]` to choose the first and fourth column.

1.3.4 Plotting SEDs

Once you have fit the data, you will likely want to plot the resulting SED fits. To do this, you will need to make use of the *`plot()`* function, which can be imported with:

```
from sedfitter import plot
```

The *`plot()`* requires the output file from the *`fit()`* function as well as the name of an output directory. For example, continuing the example above, you can do:

```
from sedfitter import plot
plot('output.fitinfo', 'plots_seds')
```

By default, only the best-fit parameter is shown, but this can be changed by using the `select_format` option, which is described in more detail in *Model selection syntax*. For example, to write out all the models with a $\Delta\chi^2$ value per data point (relative to the best fit) of less than 3, you can do:

```
plot('output.fitinfo', 'plots_seds', select_format=('F', 3))
```

In addition, there are many options available to customize the format and appearance of the plots. For more information about these options, see the *`plot()`* page.

1.3.5 Plotting parameters

Functions are available to make 1- and 2-d parameter plots:

```
from sedfitter import plot_params_1d, plot_params_2d
```

As when *Plotting SEDs*, one needs to specify the output file from the *`fit()`* function, the output directory, and the name of the parameters to plot:

```
from sedfitter import plot_params_1d, plot_params_2d

# Make histograms of the disk mass
plot_params_1d('output.fitinfo', 'MDISK', 'plots_mdisk',
               log_x=True)

# Make 2-d plots of the envelope infall rate vs disk mass
plot_params_2d('output.fitinfo', 'MDISK', 'MDOT', 'plots_mdot_mdisk',
               log_x=True, log_y=True)
```

By default, only the best-fit parameter is shown, but this can be changed by using the `select_format` option, which is described in more detail in *Model selection syntax*. In addition, there are many options available to customize the format and appearance of the plots. For more information about these options, see the *plot_params_1d()* and *plot_params_2d()* pages.

1.3.6 Splitting well- and badly-fit sources

After computing the fits with *fit()*, it is possible to split the output file on the basis of the χ^2 value for the best-fit. This is done using the *filter_output()* function which is imported with:

```
from sedfitter import filter_output
```

For example, to split the above file into well- and badly-fit sources based on the absolute χ^2 of the best-fit, you can do:

```
filter_output('output.fitinfo', chi=3.)
```

This will produce files named `output_good.fitinfo` and `output_bad.fitinfo` by default (although you can also specify custom names for the output files). It is also possible to split the fits based on the χ^2 value per datapoint using the `cpd` option. More information about the available options is available in *filter_output()*.

1.3.7 Extracting the fit and model parameters

The output files produced above are in binary format and are not human-readable. To produce ASCII files of the output, you can use the *write_parameters()* and *write_parameter_ranges()* functions. The former is used to write out all the parameters of all the models requested, while the latter will only write out the minimum and maximum for each parameter. The functions are imported with:

```
from sedfitter import write_parameters, write_parameter_ranges
```

To use these functions, you will need to specify the input binary file, and the output ASCII file name:

```
from sedfitter import write_parameters, write_parameter_ranges

# Write out all models with a delta chi^2-chi_best^2 per datapoint < 3
write_parameters('output.fitinfo', 'parameters.txt',
                 select_format=('F', 3.))

# Write out the min/max ranges corresponding to the above file
write_parameter_ranges('output.fitinfo', 'parameter_ranges.txt',
                       select_format=('F', 3.))
```

More information about the available options is given in *write_parameters()* and *write_parameter_ranges()*.

1.4 Convolving models with new filters

1.4.1 Preparing filters

This section describes how to convolve model packages with new filters. As described in the appendix Robitaille et al. (2007), there are a number of ways of defining filters/responses - for example one can define them as the fraction of photons that pass through the system, or the fraction of the energy that passes through the system. One also needs to understand what assumptions are being made to be able to quote a monochromatic flux for a broadband flux (for example, MIPS 24 micron fluxes typically assume that the underlying spectrum is a 10,000K blackbody).

The details of working out these assumptions is left to the user, and the *Filter* class used in the SED fitter requires that the user has already transformed and normalized the filter, such that the convolution can be done using the simple expression:

$$F_\nu[\text{quoted}] = \int \mathrm{F}_\nu[\text{actual}]\,\mathrm{R}_\nu\,\mathrm{d}\nu$$

The filter class is imported using:

```
from sedfitter.filter import Filter
```

and should be instantiated as follows:

```
f = Filter()
f.name = ...
f.central_wavelength = ...
f.nu = ...
f.response = ...
```

for example:

```
f = Filter()
f.name = "NEW_FILT"
f.central_wavelength = 130. * u.micron
f.nu = np.logspace(12., 13., 100) * u.Hz
f.response = np.exp(-(np.log10(f.nu.value) - 12.5)**2 / (2. * 0.1)**2)
```

The filter can then be normalized (so that the integral over ν is 1) using:

```
f.normalize()
```

Once you have set up and normalized the filters, you are now ready to use the convolution functions.

1.4.2 Broadband convolution

Assuming that you have a model directory from *Available Model Packages* or *Creating model packages*, and filters prepared as described above, you can now use the *convolve_model_dir()* function to create the required convolved flux files. This function is used as:

```
from sedfitter.convolve import convolve_model_dir
convolve_model_dir(model_dir, filters)
```

where `model_dir` is a string containing the path to the models directory, and `filters` is a list of *Filter* instances for which you want to compute the convolved fluxes. For example, using the filter object from above:

```
convolve_model_dir('models_r06', [f])
```

This will then create new files in `models_r06/convolved/`, and you will then be able to use the filters for any further fitting.

1.4.3 Monochromatic 'convolution'

In addition to conlving models with the SEDs, it is also possible to pretend that each wavelength for which the SEDs are defined can be associated with a delta function filter, and create a convolved flux file for each of these wavelengths. This can be done using:

```
from sedfitter.convolve import convolve_model_dir_monochromatic
filters = convolve_model_dir_monochromatic(model_dir)
```

This will return a table containing details about the 'filters' that have been created.

1.5 Object-oriented fitting interface

In addition to fitting via the *fit()* function, it is possible to make use of the *Fitter* class to load in all the models and set the fitting parameters, and then fit sources on-demand (this can be useful for GUI tools for example). To use the *Fitter* class, first instantiate the class by setting up the fit parameters, similarly to the *fit()* function, except that no data file or output file is passed to the *Fitter* class:

```
from astropy import units as u
from sedfitter import Fitter
from sedfitter.extinction import Extinction

# Define path to models
model_dir = '/Volumes/Data/models/models_r06'

# Read in extinction law)
extinction = Extinction.from_file('kmh94.par', columns=[0, 3],
                                  wav_unit=u.micron, chi_unit=u.cm**2 / u.g)

# Define filters and apertures
filters = ['2J', '2H', '2K', 'I1', 'I2', 'I3', 'I4']
apertures = [3., 3., 3., 3., 3., 3., 3.] * u.arcsec

# Run the fitting
fitter = Fitter(filters, apertures, model_dir,
                extinction_law=extinction,
                distance_range=[1., 2.] * u.kpc,
                av_range=[0., 40.])
```

To fit sources, you will then need to set up *Source* instances. You can either set up a *Source* instance with a single data file line:

```
from sedfitter.source import Source
s = Source.from_ascii('source_2 0.0 0.0 1 1 1 0.2 0.05 1.2 0.1 1.8 0.3')
```

or by instantiating the class and setting the parameters manually:

```
s = Source()
s.name = ...
s.x = ...
```

Once you have a *Source* instance, you can pass it to the *fit()* method:

```
info = fitter.fit(s)
```

The returned object is a *FitInfo* instance which can be passed instead of a filename to all the post-processing functions (such as *plot()*).

CHAPTER 2

Appendices

2.1 Data format

The data file you will need to feed to the fitter should contain one source per line. Each line should contain `3*(n+1)` columns, where `n` is the number of wavelengths/filters for which fluxes are given:

- Column `1` should contain the source name, with no spaces
- Columns `2` and `3` should contain the source coordinates
- Columns `4` to `3+n` should contain a flag for each flux, where the flag can take the following values:
  - `0` for data points which are not valid or used
  - `1` for valid data points
  - `2` for lower limits
  - `3` for upper limits
  - `4` for valid data points (as for `1`) but where the fluxes and errors are expressed as $\log_{10}[\text{flux}]$ and $\log_{10}[\text{error}]$. This is useful for sources which have a magnitude with a large error, which can then be converted to $\log_{10}[\text{flux}]$ and $\log_{10}[\text{error}]$ without worrying about the conversion from magnitude and magnitude error to linear fluxes and errors.
- Columns `4+n` to `3*(n+1)` should contain the fluxes and flux uncertainties, alternated, in mJy. For lower and upper limits, the error should be set to the confidence placed on the limit, with 0 corresponding to 0% and 1 corresponding to 100%. For example, in the case of an upper limit, setting the confidence to 1 effectively forbids any models to be larger than the upper limit, while setting the confidence between 0 and 1 allows such models, but introduces a χ^2 penalty dependent on the confidence. Setting the confidence to 0 effectively disables the upper limit.

We refer to this format as the **fitter data format**. Note that in a single data file, all lines should have the same number of columns. If one or more fluxes are not available for a specific source, the flag for these fluxes can be set to `0`.

An example data file with four fluxes per source is shown here:

```
SSTGLMC_G009.8925-00.3420 9.89250 -0.34200 1 1 1 1   6.869e+01   6.869e+00   1.
↪512e+02   1.512e+01   1.626e+02   1.626e+01   9.015e+01   9.015e+00
SSTGLMC_G009.8940-00.3367 9.89408 -0.33675 0 0 1 1  -9.999e+02  -9.999e+02  -9.
↪999e+02  -9.999e+02   7.412e+00   7.412e-01   1.199e+01   1.199e+00
SSTGLMC_G009.8943-00.3378 9.89430 -0.33783 0 0 1 1  -9.999e+02  -9.999e+02  -9.
↪999e+02  -9.999e+02   4.462e+00   4.462e-01   6.927e+00   6.927e-01
```

2.2 Model selection syntax

Many of the functions in the SED fitter accept a tuple containing a single character string and a value. This tuple is used to indicate how many models to plot, output, keep, etc.

The character should indicate what kind of selection is being done, and the number quantifies the selection. The options are:

- `('A', value)`: select all models (`value` is ignored).
- `('N', value)`: select a fixed number of fits given by `value`.
- `('C', value)`: select all models with a χ^2 below a threshold given by `value`.
- `('D', value)`: select all models with a $\chi^2 - \chi^2_{\rm best}$ value below a threshold given by `value`.
- `('E', value)`: select all models with a χ^2 per datapoint below a threshold given by `value`.
- `('F', value)`: select all models with a $\chi^2 - \chi^2_{\rm best}$ value per datapoint below a threshold given by `value`.

2.3 Available Model Packages

Below you will find links to existing model packages that can be downloaded.

2.3.1 Robitaille et al. (2006) YSO SED Models

These are models that were published in Robitaille et al. (2006).

Important: Please read this disclaimer before downloading the models!

You are responsible for reading about the models and ensuring that you understand all the limitations inherent to YSO SED models in general, and this specific set of models. A number of caveats related to the models are listed in Section 2.2.4 of Robitaille et al. (2006) and a number of more general limitations are also discussed in Robitaille (2008).

In particular, it is *very* important to not over-interpret results carried out by fitting these models. The models are simple axisymmetric 2-D models, which are likely only a very rough approximation of reality, and the models should only be used accordingly. The χ^2 value is thus not meaningful in the traditional statistical sense of being linked to a well-defined probability. The models also include strong correlations between parameters due to the way the parameters were sampled (this is described in more detail in the two publications linked above) and can therefore not be reliably used to search for real trends in datasets.

An example of a study using the models in the way they were intended (as rough probes of evolutionary stage), see Forbrich et al. (2010). In addition, an example of a study of the reliability of SED modeling for protostars with these models is presented in Offner et al. (2012).

By downloading the models, you acknowledge that you have read and agree with the above disclaimer

Download: models_r06_17jun08.tgz [5.3Gb]

Note: the default `parameters.fits` file contained in the above model package does not include the envelope mass. If you are interested in this parameter, you will need to download this file, decompress it, then put it inside the `models_r06` directory and make sure it is named `parameters.fits`.

2.3.2 Castelli & Kurucz (2004) stellar photosphere models

These are the Castelli and Kurucz (2004) models downloaded from here and bundled into the model package format as a convenience.

Important: Please read this disclaimer before downloading the models!

These models are provided as a convenience, and no warranty is made on the accuracy of the models or of the conversion into model package format. It is your responsability to ensure that you understand the models and their limitations.

By downloading the models, you acknowledge that you have read and agree with the above disclaimer

Download: models_kurucz_05sep11.tgz [87Mb]

2.4 Creating model packages

This page describes the format required of a *model package* in order to be used by the fitter. The format adopted for all files is the FITS standard. All FITS files presented here should have `BITPIX=-32` and `EXTEND=T` in the primary header. The model package should have a name starting with `model_` and containing the following:

- `models.conf` - a configuration file for the models.
- `seds/` - a directory containing all the model SEDs as `fits` or `fits.gz` files, in the format described below. In order to avoid huge numbers of files in a single directory, sub-directories can be created with a section of the model names. The size of this section can later be specified in the models *Configuration file*.
- `convolved/` - a directory containing all the convolved models, one file per filter. The name of the files should be `filter.fits` where `filter` should be replaced with the filter code (e.g. `2J`, `2H`, etc.).
- `parameters.fits.gz` - the parameters for the models, in the same order as the convolved fluxes

2.4.1 Configuration file

The `models.conf` configuation file should be a plain-ASCII file containing the following keyword/value pairs:

- The name of the model package:

```
name = YSO models
```

- The length of the sub-directories for SED files:

```
length_subdir = 5
```

In cases where hundreds of thousands of SEDs are present, it is recommended to split them into sub-directories inside the `seds` directory. This is done by taking the first few letters of all the models and putting all the models that have the same first few letters in the same sub-directory, with the name of these first letters. The above option is the length of the sub-string to use for this. For example, if `length_subdir` is set to 5, all models starting with `30001` should be in a directory called `30001` inside `seds/`.

- Whether the models are aperture/scale dependent:

```
aperture_dependent = yes
```

This should only be set to `no` if the models are not absolutely scaled to 1kpc, and if they are only specified in one aperture. In this case, the `distance_range` option to *fit()* is ignored.

- The step in the distance, in log space. Model SEDs are computed for a range of distances between `mind` and `maxd`, uniformly spaced in log space, and separated by this value:

```
logd_step = 0.025
```

2.4.2 SED files

HDU 0

The primary header must contain following keywords:

```
VERSION  = 1   (integer) (the current version, in case we make big changes in future)
MODEL    = value (character) (the model name)
IMAGE    = T/F (logical) (whether images are available in HDU 0)
WAVLGHTS = T/F (logical) (whether wavelengths are available in HDU 1)
APERTURS = T/F (logical) (whether apertures are available in HDU 2)
SEDS     = T/F (logical) (whether SEDs are available in HDU 3)
```

If `WAVLGHTS = T`, then the header must contain:

```
NWAV = value (integer) which is the number of wavelengths
```

If `APERTURS = T`, then the header must contain:

```
NAP = value (integer) which is the number of apertures
```

If `IMAGE = T`, then this **HDU** should contain a data cube which is the model image as a function of wavelength. This will be specified in more detail in future.

HDU 1

if `WAVLGTHS = T`, this **HDU** should contain a 2-column binary table with `NWAV` values in each column. The first column should have the title `WAVELENGTH`, format `1E`, and unit `MICRONS`. The second column should have the title `FREQUENCY`, format `1E`, and unit `HZ`. Optionally, a third column can be included with the title `STELLAR_FLUX`, format `1E`, and units specified appropriately. This column can be used to specify the model stellar photosphere used to compute YSO models for example. Only `mJy` and `ergs/cm\^2/s` are supported as units at this time. If specified, the stellar photosphere should have the correct scaling relative to the SEDs in `HDU 3`.

HDU 2

if `APERTURS = T`, this **HDU** should contain a 1-column binary table with `NAP` values. The column should have the title `APERTURE`, format `1E`, and units `AU`. These are the apertures for which the SEDs in **HDU 3** are tabulated.

HDU 3

if `SED = T`, this **HDU** should contain a binary table with at least one column, and **HDU 1** and **HDU 2** should also contain data. Each column should consist of `NAP` rows of real vectors with dimension `NWAV`. Thus, each cell contains an SED. The format of each column should be `nE`, where `n=NWAV`. The title and units of each column should be specified. The columns can contains SEDs such as the total flux, the stellar flux, the disk flux, etc. or related uncertainties. The following column titles are examples of ones that can be used:

```
TOTAL_FLUX
TOTAL_FLUX_ERR
STELLAR_FLUX
STELLAR_FLUX_ERR
DISK_FLUX
DISK_FLUX_ERR
ENVELOPE_FLUX
ENVELOPE_FLUX_ERR
DIRECT_FLUX
DIRECT_FLUX_ERR
SCATTERED_FLUX
SCATTERED_FLUX_ERR
THERMAL_FLUX
THERMAL_FLUX_ERR
etc.
```

The order of the columns is not important as there are `FITS` routines to search for a specific column.

Note: The SED fitter requires a column `TOTAL_FLUX` to be present, and will return an error otherwise. Only `mJy` and `ergs/cm^2/s` are supported as units at this time.

2.4.3 Convolved fluxes file

HDU 0

The primary header must contain following keywords:

```
FILTWAV  = value (real) (the characteristic wavelength of the filter)
NMODELS  = value (integer) (the number of models)
NAP      = value (integer) (the number of apertures)
```

HDU 1

This **HDU** should contain a 5-column binary table. The column titles should be:

```
MODEL_NAME
TOTAL_FLUX
TOTALF_FLUX_ERR
```

(continues on next page)

(continued from previous page)

```
RADIUS_SIGMA_50
RADIUS_CUMUL_99
```

The first column should have format `30A` and should contain the name of each model. No units are required. The second and third columns should have format `nE` where `n=NAP`, with each cell containing a vector with the fluxes in the different apertures. The fourth and fifth column should have format `1E` and contain the outermost radius at which the surface brightness falls to 50% of the maximum surface brightness, and the radius inside which 99% of the flux is contained respectively. These two columns should have units `AU`.

HDU 2

This **HDU** should contain a 1-column binary table with `NAP` values. The column should have the title `APERTURE`, format `1E`, and units `AU`. These are the apertures for which the fluxes in **HDU 1** are tabulated.

2.4.4 Model parameters

HDU 0

The primary header must contain following keywords:

```
NMODELS  = value (integer) (the number of models)
```

HDU 1

This **HDU** should contain a binary table with the model parameters. Any number of columns can be included, in any order. Only parameters with format `1E` will be usable by the programs to plot parameters, but text parameters with format `nA` can also be included (e.g. dust model filenames, etc.). One column is compulsory, with title `MODEL_NAME` and format `30A`. It should contain the same names as the convolved fluxes file, and in the same order.

2.5 Common filter names

The filters for which convolved fluxes are available varies depending on the model package. However, The following list gives common filter names:

Short name	Filter
BU	Bessel U-Band
BB	Bessel B-Band
BV	Bessel V-Band
BR	Bessel R-Band
BI	Bessel I-Band
2J	2MASS J-Band
2H	2MASS H-Band
2K	2MASS K-Band
UL	UKIRT L-Band
LP	UKIRT L'-Band
UM	UKIRT M-Band
I1	IRAC Band [1]

Continued on next page

Table 1 – continued from previous page

Short name	Filter
I2	IRAC Band [2]
I3	IRAC Band [3]
I4	IRAC Band [4]
XA	MSX Band A
XC	MSX Band C
XD	MSX Band D
XE	MSX Band E
M1	MIPS 24
M2	MIPS 70
M3	MIPS 160
S1	IRAS 12 microns
S2	IRAS 25 microns
S3	IRAS 60 microns
S4	IRAS 100 microns
W1	Scuba 450WB microns
W2	Scuba 850WB microns
N1	Scuba 350NB microns
N2	Scuba 450NB microns
N3	Scuba 750NB microns
N4	Scuba 850NB microns

2.6 Detailed description of functions and arguments (API)

2.6.1 sedfitter Package

Functions

extract_parameters([input, output_prefix, ...])	
filter_output([input_fits, output_good, ...])	Filter an output file into well and badly fit sources.
fit(data, filter_names, apertures, ...[, ...])	Fit a set of sources with models.
plot(input_fits[, output_dir, ...])	Make SED plots
plot_params_1d(input_fits, parameter[, ...])	Make histogram plots of parameters
plot_params_2d(input_fits, parameter_x, ...)	Make histogram plots of parameters
plot_source_data(ax, source, filters[, ...])	
validate_array(name, value[, domain, ndim, ...])	
write_parameter_ranges(input_fits, output_file)	Write out an ASCII file with ranges of paramters for each source.
write_parameters(input_fits, output_file[, ...])	Write out an ASCII file with the paramters for each source.

extract_parameters

sedfitter.**extract_parameters**(*input=None*, *output_prefix=None*, *output_suffix=None*, *parameters='all'*, *select_format=('N', 1)*, *header=True*)

filter_output

sedfitter.**filter_output**(*input_fits=None*, *output_good='auto'*, *output_bad='auto'*, *chi=None*, *cpd=None*)

Filter an output file into well and badly fit sources.

The sources in the output file are split into ones where the best fit is good, and those where the best fit is bad, where the distinction between good and bad is made based on the chi^2 value or the chi^2 value per datapoint of the best fit.

Parameters

input_fits [str or *sedfitter.fit_info.FitInfo* or iterable] This should be either a file containing the fit information, a *sedfitter.fit_info.FitInfo* instance, or an iterable containing *sedfitter.fit_info.FitInfo* instances.

output_good [str, optional] The name of the file containing information about well-fit sources. If set to 'auto', then the output file will be set to the same as the input file but with a `_good` prefix.

output_bad [str, optional] The name of the file containing information about badly-fit sources. If set to 'auto', then the output file will be set to the same as the input file but with a `_bad` prefix.

chi [float, optional] If set, the separation between well and badly fit sources is done based on the total chi^2 of the best fit, using this value as a threshold.

cpd [float, optional] If set, the separation between well and badly fit sources is done based on the chi^2 per datapoint of the best fit, using this value as a threshold.

fit

sedfitter.**fit**(*data*, *filter_names*, *apertures*, *model_dir*, *output*, *n_data_min=3*, *extinction_law=None*, *av_range=None*, *distance_range=None*, *output_format=('F', 6.0)*, *output_convolved=False*, *remove_resolved=False*)

Fit a set of sources with models.

Parameters

data [str] Filename of the file containing the data, one source per line (see documentation for a description of the required format).

filter_names [tuple or list] List of filter names (given as individual strings) for which the data is defined. The filter names should be the name of the files in the `convolved` directory for the models, without the extensions. This is typically `2J`, `I1`, `M1`, etc. You can also specify the wavelength as a `Quantity` instance instead of a filter name, and this will indicate that the SED fluxes closest to the requested wavelength should be used in the fitting.

apertures [`Quantity` array instance] The aperture radii that the data is specified in (as an angle). The fluxes may not be measured from aperture photometry, but this is meant to give an indication of the sizescale of the emission, and can be used to reject models that would have been clearly resolved at the distance specified.

models_dir [str] Name of the directory containing the models to use.

output [str] Name of the file to output the fit information to (in binary format).

extinction_law [*Extinction* instance] The extinction law to use.

av_range [tuple] Minimum and maximum Av to allow in the fitting.

distance_range [Quantity array instance] Minimum and maximum distance to allow in the fitting in units of length.

n_data_min [int, optional] The minimum number of points a source needs to be fit.

output_format [tuple, optional] Tuple specifying which fits should be output. See the documentation for a description of the tuple syntax.

output_convolved [bool, optional] Whether to output the convolved fluxes (necessary if the convolved model fluxes are needed for the SED plot).

remove_resolved [bool, optional] If set, then models larger than the aperture are removed. See Robitaille et al. (2007) for a discussion of this criterion.

plot

sedfitter.**plot**(*input_fits*, *output_dir=None*, *select_format=('N', 1)*, *plot_max=None*, *plot_mode='A'*, *sed_type='interp'*, *show_sed=True*, *show_convolved=False*, *x_mode='A'*, *y_mode='A'*, *x_range=(1.0, 1.0)*, *y_range=(1.0, 2.0)*, *plot_name=True*, *plot_info=True*, *format='pdf'*, *sources=None*, *memmap=True*, *dpi=None*)

Make SED plots

Parameters

input_fits [str or *sedfitter.fit_info.FitInfo* or iterable] This should be either a file containing the fit information, a *sedfitter.fit_info.FitInfo* instance, or an iterable containing *sedfitter.fit_info.FitInfo* instances.

output_dir [str, optional] If specified, plots are written to that directory

select_format [tuple, optional] Tuple specifying which fits should be plotted. See the documentation for a description of the tuple syntax.

plot_max [int, optional] Maximum number of fits to plot

plot_mode [str, optional] Whether to plot all fits in a single plot ('A') or one fit per plot ('I')

sed_type [str, optional]

Which SED should be shown:

- *largest*: show the SED for the largest aperture specified.
- *smallest*: show the SED for the smallest aperture specified.
- *largest+smallest*: show the SEDs for the largest and smallest apertures specified.
- *all*: show the SEDs for all apertures specified.
- *interp*: interpolate the SEDs to the correct aperture at each wavelength (and interpolated apertures in between), so that a single composite SED is shown.

show_sed [bool, optional] Show the SEDs

show_convolved [bool, optional] Show convolved model fluxes

x_mode [str, optional] Whether to automatically select the wavelength range ('A'), or whether to use manually set values ('M').

x_range [tuple, optional] If x_mode is set to 'M', this is the range of wavelengths to show. If x_mode is set to 'A', this is the marging to add around the wavelength range (in dex).

y_mode [str, optional] Whether to automatically select the flux range ('A'), or whether to use manually set values ('M').

y_range [tuple, optional] If y_mode is set to 'M', this is the range of fluxes to show. If y_mode is set to 'A', this is the marging to add around the flux range (in dex).

plot_name [bool, optional] Whether to show the source name on the plot(s).

plot_info [bool, optional] Whether to show the fit information on the plot(s).

format [str, optional] The file format to use for the plot, if output_dir is specified.

sources [list, optional] If specified, gives the list of sources to plot. If not set, all sources will be plotted

dpi [int, optional] The resolution of the figure to save

plot_params_1d

sedfitter.**plot_params_1d**(*input_fits*, *parameter*, *output_dir=None*, *select_format=('N', 1)*, *log_x=False*, *log_y=True*, *label=None*, *bins=30*, *hist_range=None*, *additional={}*, *plot_name=True*, *format='pdf'*, *dpi=None*)

Make histogram plots of parameters

Parameters

input_fits [str or *sedfitter.fit_info.FitInfo* or iterable] This should be either a file containing the fit information, a *sedfitter.fit_info.FitInfo* instance, or an iterable containing *sedfitter.fit_info.FitInfo* instances.

parameter [str] The parameter to plot a histogram of

output_dir [str, optional] If specified, plots are written to that directory

select_format [tuple, optional] Tuple specifying which fits should be plotted. See the documentation for a description of the tuple syntax.

log_x [bool, optional] Whether to plot the x-axis values in log space

log_y [bool, optional] Whether to plot the y-axis values in log space

label [str, optional] The x-axis label (if not specified, the parameter name is used)

bins [int, optional] The number of bins for the histogram

hist_range [tuple, optional] The range of values to show

additional [dict, optional] A dictionary specifying additional parameters not listed in the parameter list for the models. Each item of the dictionary should itself be a dictionary giving the values for each model (where the key is the model name).

plot_name: bool, optional Whether to show the source name on the plot(s).

format: str, optional The file format to use for the plot, if output_dir is specified.

dpi [int, optional] The resolution of the figure to save

plot_params_2d

sedfitter.**plot_params_2d**(*input_fits*, *parameter_x*, *parameter_y*, *output_dir=None*, *select_format=('N', 1)*, *log_x=False*, *log_y=True*, *bounds=None*, *label_x=None*, *label_y=None*, *additional=None*, *plot_name=True*, *format='pdf'*, *dpi=None*)

Make histogram plots of parameters

Parameters

input_fits [str or *sedfitter.fit_info.FitInfo* or iterable] This should be either a file containing the fit information, a *sedfitter.fit_info.FitInfo* instance, or an iterable containing *sedfitter.fit_info.FitInfo* instances.

parameter_x [str] The parameter to plot on the x-axis

parameter_y [str] The parameter to plot on the y-axis

output_dir [str, optional] If specified, plots are written to that directory

select_format [tuple, optional] Tuple specifying which fits should be plotted. See the documentation for a description of the tuple syntax.

log_x [bool, optional] Whether to plot the x-axis values in log space

log_y [bool, optional] Whether to plot the y-axis values in log space

label_x [str, optional] The x-axis label (if not specified, the parameter name is used)

label_y [str, optional] The y-axis label (if not specified, the parameter name is used)

additional [Table] A dictionary specifying additional parameters not listed in the parameter list for the models. Each item of the dictionary should itself be a dictionary giving the values for each model (where the key is the model name).

plot_name: bool, optional Whether to show the source name on the plot(s).

format: str, optional The file format to use for the plot, if output_dir is specified.

dpi [int, optional] The resolution of the figure to save

plot_source_data

sedfitter.**plot_source_data**(*ax*, *source*, *filters*, *size=20*, *capsize=3*)

validate_array

sedfitter.**validate_array**(*name*, *value*, *domain=None*, *ndim=1*, *shape=None*, *physical_type=None*)

write_parameter_ranges

sedfitter.**write_parameter_ranges**(*input_fits*, *output_file*, *select_format=('N', 1)*, *additional={}*)

Write out an ASCII file with ranges of paramters for each source.

Parameters

input_fits [str or *sedfitter.fit_info.FitInfo* or iterable] This should be either a file containing the fit information, a *sedfitter.fit_info.FitInfo* instance, or an iterable containing *sedfitter.fit_info.FitInfo* instances.

output_file [str, optional] The output ASCII file containing the parameter ranges

select_format [tuple, optional] Tuple specifying which fits should be output. See the documentation for a description of the tuple syntax.

additional [dict, optional] A dictionary giving additional parameters for each model. This should be a dictionary where each key is a parameter, and each value is a dictionary mapping the model names to the parameter values.

write_parameters

sedfitter.**write_parameters**(*input_fits*, *output_file*, *select_format=('N', 1)*, *additional={}*)

Write out an ASCII file with the paramters for each source.

Parameters

input_fits [str or *sedfitter.fit_info.FitInfo* or iterable] This should be either a file containing the fit information, a *sedfitter.fit_info.FitInfo* instance, or an iterable containing *sedfitter.fit_info.FitInfo* instances.

output_file [str, optional] The output ASCII file containing the parameters

select_format [tuple, optional] Tuple specifying which fits should be output. See the documentation for a description of the tuple syntax.

additional [dict, optional] A dictionary giving additional parameters for each model. This should be a dictionary where each key is a parameter, and each value is a dictionary mapping the model names to the parameter values.

Classes

Extinction()	
FitInfoFile(fits[, mode])	
Fitter(filter_names, apertures, model_dir[, ...])	A fitter class that can be used to fit sources.
Models()	
Source()	

Extinction

class sedfitter.**Extinction**

Bases: object

Attributes Summary

chi	
wav	

Methods Summary

from_file(filename[, columns, wav_unit, ...])	Read an extinction law from an ASCII file.
from_table(table)	
get_av(wav)	Interpolate the Av at given wavelengths
to_table()	

Attributes Documentation

chi

wav

Methods Documentation

classmethod from_file(*filename*, *columns=(0, 1)*, *wav_unit=Unit("micron")*, *chi_unit=Unit("cm2 / g")*)
Read an extinction law from an ASCII file.

This reads in two columns: the wavelength, and the opacity (in units of area per unit mass).

Parameters

filename [str, optional] The name of the file to read the extinction law from

columns [tuple or list, optional] The columns to use for the wavelength and opacity respectively

wav_unit [Unit] The units to assume for the wavelength

chi_unit [Unit] The units to assume for the opacity

classmethod from_table(*table*)

get_av(*wav*)
Interpolate the Av at given wavelengths

Parameters

wav [Quantity] The wavelengths at which to interpolate the visual extinction.

to_table()

FitInfoFile

class sedfitter.**FitInfoFile**(*fits*, *mode=None*)
Bases: object

Attributes Summary

meta

Methods Summary

close()
write(info)

Attributes Documentation

meta

Methods Documentation

close()

write(*info*)

Fitter

class sedfitter.**Fitter**(*filter_names*, *apertures*, *model_dir*, *extinction_law=None*, *av_range=None*, *distance_range=None*, *remove_resolved=False*)

Bases: object

A fitter class that can be used to fit sources.

This class is initialized using a particular set of models, and with specific fit parameters. It can then be used to fit data given by Source instances, and returns a FitInfo instance. Once initialized, the fit parameters cannot be changed, because changing most of them would require re-reading the models from disk.

Parameters

filter_names [tuple or list] List of filter names (given as individual strings) for which the data is defined. The filter names should be the name of the files in the convolved directory for the models, without the extensions. This is typically 2J, I1, M1, etc. You can also specify the wavelength as a Quantity instance instead of a filter name, and this will indicate that the SED fluxes closest to the requested wavelength should be used in the fitting.

apertures [Quantity array instance] The aperture radii that the data is specified in (as an angle). The fluxes may not be measured from aperture photometry, but this is meant to give an indication of the sizescale of the emission, and can be used to reject models that would have been clearly resolved at the distance specified.

models_dir [str] Name of the directory containing the models to use.

extinction_law [*Extinction* instance] The extinction law to use.

av_range [tuple] Minimum and maximum Av to allow in the fitting.

distance_range [Quantity array instance] Minimum and maximum distance to allow in the fitting in units of length.

remove_resolved [bool, optional] If set, then models larger than the aperture are removed. See Robitaille et al. (2007) for a discussion of this criterion.

Methods Summary

fit(source)	Fit the specified source.

Methods Documentation

fit(*source*)

Fit the specified source.

Parameters

source [*~sedfitter.source.Source*] The source to fit.

Returns

fit_info [*sedfitter.fit_info.FitInfo*] The results of the fit.

Models

class sedfitter.**Models**
Bases: object

Attributes Summary

apertures	The apertures at which the fluxes are defined
distances	The distances at which the models are defined
fluxes	The model fluxes
log_fluxes_mJy	
n_ap	
n_distances	
n_models	
n_wav	
valid	
wavelengths	The wavelengths at which the models are defined

Methods Summary

fit(source, av_law, sc_law, av_min, av_max)	
read(directory, filters[, distance_range, ...])	

Attributes Documentation

apertures
The apertures at which the fluxes are defined

distances
The distances at which the models are defined

fluxes
The model fluxes

log_fluxes_mJy

n_ap

n_distances

n_models

n_wav

valid

wavelengths
The wavelengths at which the models are defined

Methods Documentation

fit(*source*, *av_law*, *sc_law*, *av_min*, *av_max*, *output_convolved=False*)

classmethod read(*directory*, *filters*, *distance_range=None*, *remove_resolved=False*)

Source

class sedfitter.**Source**

Bases: object

Attributes Summary

error
flux
n_data
n_wav
name
valid
x
y

Methods Summary

from_ascii(line)
from_dict(source_dict)
get_log_fluxes()
to_ascii()
to_dict()

Attributes Documentation

error

flux

n_data

n_wav

name

valid

x

y

Methods Documentation

classmethod from_ascii(*line*)

classmethod from_dict(*source_dict*)

get_log_fluxes()

to_ascii()

to_dict()

2.6.2 sedfitter.sed Package

Classes

BaseCube([valid, names, distance, wav, nu, ...])	A cube to represent a cube of models.
PolarizationCube([valid, names, distance, ...])	
SED()	
SEDCube([valid, names, distance, wav, nu, ...])	

BaseCube

class sedfitter.sed.**BaseCube**(*valid=None*, *names=None*, *distance=None*, *wav=None*, *nu=None*, *apertures=None*, *val=None*, *unc=None*)

Bases: object

A cube to represent a cube of models.

This consists of values and uncertainties as a function of wavelength, aperture, and models.

Parameters

names [1-d iterable, optional] The names of all the models in the cube

distance [~*astropy.units.Quantity*, optional] The distance assumed for the values

wav [1-d ~*astropy.units.Quantity*, optional] The wavelengths at which the SEDs are defined (cannot be used with nu)

nu [1-d ~*astropy.units.Quantity*, optional] The frequencies at which the SEDs are defined (cannot be used with wav)

apertures [1-d ~*astropy.units.Quantity*, optional] The ap for which the SEDs are defined

val [3-d ~*astropy.units.Quantity*, optional] The values of the fluxes or polarization

unc [3-d ~*astropy.units.Quantity*, optional] The uncertainties in the fluxes or polarization

Attributes Summary

apertures	The ap at which the SEDs are defined.
distance	The distance at which the SEDs are defined.
n_ap	
n_models	
n_wav	
names	The names of the models
nu	The frequencies at which the SEDs are defined.
unc	The uncertainties in the fluxes or polarization.

Continued on next page

Table 14 – continued from previous page

val	The fluxes or polarization values.
valid	Which models are valid
wav	The wavelengths at which the SEDs are defined.

Methods Summary

read(filename[, order, memmap])	Read models from a FITS file.
write(filename[, overwrite, meta])	Write the models to a FITS file.

Attributes Documentation

apertures
The ap at which the SEDs are defined.

distance
The distance at which the SEDs are defined.

n_ap

n_models

n_wav

names
The names of the models

nu
The frequencies at which the SEDs are defined.

unc
The uncertainties in the fluxes or polarization.

val
The fluxes or polarization values.

valid
Which models are valid

wav
The wavelengths at which the SEDs are defined.

Methods Documentation

classmethod read(*filename*, *order='nu'*, *memmap=True*)
Read models from a FITS file.

Parameters

filename: str The name of the file to read the cube from.

order: str, optional Whether to sort the SED by increasing wavelength (*wav*) or frequency ('nu').

write(*filename*, *overwrite=False*, *meta={}*)
Write the models to a FITS file.

Parameters

filename: str The name of the file to write the cube to.

PolarizationCube

class sedfitter.sed.**PolarizationCube**(*valid=None*, *names=None*, *distance=None*, *wav=None*, *nu=None*, *apertures=None*, *val=None*, *unc=None*)

Bases: *sedfitter.sed.BaseCube*

SED

class sedfitter.sed.**SED**

Bases: object

Attributes Summary

apertures	The apertures at which the SED is defined
error	The convolved flux errors
flux	The SED fluxes
n_ap	
n_wav	
nu	The frequencies at which the SED is defined
wav	The wavelengths at which the SED is defined

Methods Summary

copy()	
interpolate(apertures)	Interpolate the SED to different apertures
interpolate_variable(wavelengths, apertures)	Interpolate the SED to a variable aperture as a function of wavelength.
read(filename[, unit_wav, unit_freq, ...])	Read an SED from a FITS file.
scale_to_av(av, law)	
scale_to_distance(distance)	Returns the SED scaled to distance *distance*
write(filename[, overwrite])	Write an SED to a FITS file.

Attributes Documentation

apertures
The apertures at which the SED is defined

error
The convolved flux errors

flux
The SED fluxes

n_ap

n_wav

nu

The frequencies at which the SED is defined

wav
The wavelengths at which the SED is defined

Methods Documentation

copy()

interpolate(*apertures*)
Interpolate the SED to different apertures

interpolate_variable(*wavelengths*, *apertures*)
Interpolate the SED to a variable aperture as a function of wavelength. This method should be called with an interpolating function for aperture as a function of wavelength, in log10 space.

classmethod read(*filename*, *unit_wav=Unit("micron")*, *unit_freq=Unit("Hz")*, *unit_flux=Unit("erg / (cm2 s)")*, *order='nu'*)
Read an SED from a FITS file.

Parameters

filename: str The name of the file to read the SED from.

unit_wav: '~astropy.units.Unit', optional The units to convert the wavelengths to.

unit_freq: '~astropy.units.Unit', optional The units to convert the frequency to.

unit_flux: '~astropy.units.Unit', optional The units to convert the flux to.

order: str, optional Whether to sort the SED by increasing wavelength (*wav*) or frequency ('nu').

scale_to_av(*av*, *law*)

scale_to_distance(*distance*)
Returns the SED scaled to distance *distance*

Parameters

distance [float] The distance in cm

Returns

sed [SED] The SED, scaled to the new distance

write(*filename*, *overwrite=False*)
Write an SED to a FITS file.

Parameters

filename: str The name of the file to write the SED to.

SEDCube

class sedfitter.sed.**SEDCube**(*valid=None*, *names=None*, *distance=None*, *wav=None*, *nu=None*, *apertures=None*, *val=None*, *unc=None*)
Bases: *sedfitter.sed.BaseCube*

Methods Summary

get_sed(model_name)	

Methods Documentation

get_sed(*model_name*)

2.6.3 sedfitter.convolved_fluxes Package

Classes

ConvolvedFluxes([wavelength, model_names, ...])	
MonochromaticFluxes([wavelength, ...])	

ConvolvedFluxes

class sedfitter.convolved_fluxes.**ConvolvedFluxes**(*wavelength=None*, *model_names=None*, *apertures=None*, *flux=None*, *error=None*, *initialize_arrays=False*, *initialize_units=Unit("mJy")*)

Bases: object

Attributes Summary

apertures	The apertures at which the SED is defined
central_wavelength	The central or characteristic wavelength of the filter
error	The convolved flux errors
flux	The SED fluxes
model_names	The names of the models
n_ap	
n_models	

Methods Summary

find_radius_cumul(fraction)	Find for each model the radius containing a fraction of the flux.
find_radius_sigma(fraction)	Find for each model a fractional surface brightness radius
interpolate(apertures)	Interpolate the flux to the apertures specified.
read(filename)	Read convolved flux from a FITS file
sort_to_match(requested_model_names)	Sort the models following `requested_model_names`

Continued on next page

Table 21 – continued from previous page

write(filename[, overwrite])	Write convolved flux to a FITS file.

Attributes Documentation

apertures
The apertures at which the SED is defined

central_wavelength
The central or characteristic wavelength of the filter

error
The convolved flux errors

flux
The SED fluxes

model_names
The names of the models

n_ap

n_models

Methods Documentation

find_radius_cumul(*fraction*)
Find for each model the radius containing a fraction of the flux.

Parameters

fraction: float The fraction to use when determining the radius

find_radius_sigma(*fraction*)
Find for each model a fractional surface brightness radius

This is the outermost radius where the surface brightness is larger than a fraction of the peak surface brightness.

Parameters

fraction: float The fraction to use when determining the radius

interpolate(*apertures*)
Interpolate the flux to the apertures specified.

Parameters

apertures [*astropy.units.Quantity* instance] The apertures to interpolate to

classmethod read(*filename*)
Read convolved flux from a FITS file

Parameters

filename [str] The name of the FITS file to read the convolved fluxes from

sort_to_match(*requested_model_names*)
Sort the models following `requested_model_names`

write(*filename*, *overwrite=False*)
Write convolved flux to a FITS file.

Parameters

filename: str The name of the file to output the convolved fluxes to.

overwrite: bool, optional Whether to overwrite the output file

MonochromaticFluxes

class sedfitter.convolved_fluxes.**MonochromaticFluxes**(*wavelength=None*, *model_names=None*, *apertures=None*, *flux=None*, *error=None*, *initialize_arrays=False*, *initialize_units=Unit("mJy")*)

Bases: *sedfitter.convolved_fluxes.ConvolvedFluxes*

Methods Summary

from_sed_cube(cube, wavelength_index)	

Methods Documentation

classmethod from_sed_cube(*cube*, *wavelength_index*)

2.6.4 sedfitter.extinction Package

Classes

Extinction()	

Extinction

class sedfitter.extinction.**Extinction**

Bases: object

Attributes Summary

chi	
wav	

Methods Summary

from_file(filename[, columns, wav_unit, ...])	Read an extinction law from an ASCII file.
from_table(table)	
get_av(wav)	Interpolate the Av at given wavelengths

Continued on next page

Table 25 – continued from previous page

to_table()

Attributes Documentation

chi

wav

Methods Documentation

classmethod from_file(*filename*, *columns=(0, 1)*, *wav_unit=Unit("micron")*, *chi_unit=Unit("cm2 / g")*)
Read an extinction law from an ASCII file.

This reads in two columns: the wavelength, and the opacity (in units of area per unit mass).

Parameters

filename [str, optional] The name of the file to read the extinction law from

columns [tuple or list, optional] The columns to use for the wavelength and opacity respectively

wav_unit [Unit] The units to assume for the wavelength

chi_unit [Unit] The units to assume for the opacity

classmethod from_table(*table*)

get_av(*wav*)
Interpolate the Av at given wavelengths

Parameters

wav [Quantity] The wavelengths at which to interpolate the visual extinction.

to_table()

2.6.5 sedfitter.fit_info Module

Classes

FitInfo([source])	Results from a fit of a set of models to a source.
FitInfoFile(fits[, mode])	
FitInfoMeta	

FitInfo

class sedfitter.fit_info.**FitInfo**(*source=None*)
Bases: object

Results from a fit of a set of models to a source.

Methods Summary

filter_table(input_table[, additional])	Given an input table, return only the rows matching the FitInfo object, and in the same order.
keep(select_format)	Keep only a fraction of fits.
sort()	Sort the fit results from best to worst based on the chi^2 value.

Methods Documentation

filter_table(*input_table*, *additional={}*)
Given an input table, return only the rows matching the FitInfo object, and in the same order.

Parameters

input_table [*~astropy.table.Table*] The input table to filter and sort

additional [dict, optional] Additional columns to include in the table. This can be specified as a dictionary of Numpy arrays, where the key of the dictionary is the name of the column to add to the table.

keep(*select_format*)
Keep only a fraction of fits.

Parameters

select_format [tuple, optional] Tuple specifying which fits should be output. See the documentation for a description of the tuple syntax.

sort()
Sort the fit results from best to worst based on the chi^2 value.

FitInfoFile

class sedfitter.fit_info.**FitInfoFile**(*fits*, *mode=None*)
Bases: object

Attributes Summary

meta

Methods Summary

close()
write(info)

Attributes Documentation

meta

Methods Documentation

close()

write(*info*)

FitInfoMeta

class sedfitter.fit_info.**FitInfoMeta**
Bases: object

2.6.6 sedfitter.source Package

Classes

Source()

Source

class sedfitter.source.**Source**
Bases: object

Attributes Summary

error
flux
n_data
n_wav
name
valid
x
y

Methods Summary

from_ascii(line)
from_dict(source_dict)
get_log_fluxes()
to_ascii()
to_dict()

Attributes Documentation

error

flux

n_data

n_wav

name

valid

x

y

Methods Documentation

classmethod from_ascii(*line*)

classmethod from_dict(*source_dict*)

get_log_fluxes()

to_ascii()

to_dict()

2.6.7 sedfitter.filter Package

Classes

Filter([name, central_wavelength, nu, response])	A filter used to convolve SED models.

Filter

class sedfitter.filter.**Filter**(*name=None*, *central_wavelength=None*, *nu=None*, *response=None*)

Bases: object

A filter used to convolve SED models.

Parameters

name [str] The name of the filter (typically short and with no spaces, such as 2K or 2MASS K-band or I1 for IRAC 3.6 microns).

central_wavelength [Quantity] The central wavelength (in units of length)

nu [Quantity] The frequencies at which the filter is defined.

response [*numpy.ndarray*] The relative response at the different frequencies. Note that this should already take into account the assumptions about whether this is the response per photon or energy, and any other assumptions about converting the SED to a monochromatic flux. See the documentation for more details.

Attributes Summary

central_wavelength	The central or characteristic wavelength of the filter

Continued on next page

Table 34 – continued from previous page

nu	The frequencies at which the filter is defined
response	The filter response

Methods Summary

normalize()	Normalize so the integral over nu is 1
read(filename)	Read a filter from a file.
rebin(nu_new)	Re-bin the filter onto a new frequency grid

Attributes Documentation

central_wavelength
The central or characteristic wavelength of the filter

nu
The frequencies at which the filter is defined

response
The filter response

Methods Documentation

normalize()
Normalize so the integral over nu is 1

classmethod read(*filename*)
Read a filter from a file.

This method assumes that the file contains a column with wavelengths (in micron) and the other column contains the response. It also assumes that the first line contains a header with the central wavelength (in microns), using the following syntax:

```
# wav = 1.25
```

Parameters

filename: str The name of the file containing the filter

rebin(*nu_new*)
Re-bin the filter onto a new frequency grid

Parameters

nu_new: np.ndarray The new frequency grid

Returns

filter: Filter The binned filter

2.6.8 sedfitter.convolve Package

Functions

convolve_model_dir(model_dir, filters[, ...])	Convolve all the model SEDs in a model directory
convolve_model_dir_monochromatic(model_dir[, ...])	Convolve all the model SEDs in a model directory

convolve_model_dir

sedfitter.convolve.**convolve_model_dir**(*model_dir*, *filters*, *overwrite=False*, *memmap=True*)

Convolve all the model SEDs in a model directory

Parameters

model_dir [str] The path to the model directory

filters [list] A list of *Filter* objects to use for the convolution

overwrite [bool, optional] Whether to overwrite the output files

memmap [bool, optional] Whether to use memory mapping when using the SED cubes. If you have enough memory, the convolution will be much faster to set this to False, since the whole cube will need to be read in, so it's faster to do it in one go than many small reads. This option is ignored if not using SED cubes.

convolve_model_dir_monochromatic

sedfitter.convolve.**convolve_model_dir_monochromatic**(*model_dir*, *overwrite=False*, *max_ram=8*, *wav_min=<Quantity -inf micron>*, *wav_max=<Quantity inf micron>*)

Convolve all the model SEDs in a model directory

Parameters

model_dir [str] The path to the model directory

overwrite [bool, optional] Whether to overwrite the output files

max_ram [float, optional] The maximum amount of RAM that can be used (in Gb)

wav_min [float, optional] The minimum wavelength to consider. Only wavelengths above this value will be output.

wav_max [float, optional] The maximum wavelength to consider. Only wavelengths below this value will be output.

Python Module Index

S

Index

W

X

Y