spexxy Documentation

Tim-Oliver Husser

Nov 11, 2022

Contents

spexxy is a spectrum fitting framework.

CHAPTER 1

Quickstart

1.1 Installing *spexxy*

The easiest way for installing *spexxy* is using pip:

```
pip3 install spexxy
```

Add the *–user* switch to install it in your user directory (no root required).

1.2 Running *spexxy*

A basic call of *spexxy* is of the form:

```
spexxy config.yaml *.fits
```

which uses the configuration given in *config.yaml* in order to process the files specified by **.fits*.

By default, results are written into a CSV file named *spexxy.csv*, but this can be changed using the *–output* parameter. The CSV file will contain one line per file, each giving the filename and all results for that file.

A previous fit can be continued by adding the *–resume* switch, which then ignores all files, for which an entry in the output CSV file exists.

If *–mp <N>* is provided, the processing of the files will be parallelized in N processes, of which each runs a part of the given files sequentially.

1.3 Basic configuration

A basic configuration file for *spexxy* might look like this:

```
main:
  class: path.to.module.SomeClass
  variable: 123
```

With this config, *spexxy* tries to instantiante *SomeClass* from package *path.to.module* (which must therefore be in the PYTHONPATH), and forwards all other parameters next to the *class* element to its constructor. Then the object is called, i.e. its *__call__()* method is called.

1.4 Basic fit example

A little more complex example that would actually run a parameter fit on the given spectra looks like this:

```
main:
  class: spexxy.main.ParamsFit
  components:
    star:
      class: spexxy.component.GridComponent
      interpolator:
        phx:
          class: spexxy.interpolator.LinearInterpolator
          grid:
            class: spexxy.grid.FilesGrid
            filename: /path/to/grid.csv
      init:
      - class: spexxy.init.InitFromValues
        values:
          logg: 4.5
          Alpha: 0.
          v: 0
          sig: 0
  fixparams:
    star: [sig, Alpha, logg]
```

As the main routine, in this case the ParamsFit class is provided with one parameter for its constructor, which is a list of "components" for the fit.

One component is provided and will be created from the *spexxy.component.Grid* class, which encapsules a *Grid* or *Interpolator* object, which, in turn, is also given here in the form of a UlyssInterpolator with its one required parameter. All the objects defined by *class* attributes will automatically be created by *spexxy*.

1.5 References in the configuration

Note that for better readability, the config file can also be written in the following form:

```
grids:
  phxgrid:
    class: spexxy.grid.FilesGrid
    filename: /path/to/grid.csv

interpolators:
  phx:
    class: spexxy.interpolator.LinearInterpolator
    grid: phxgrid
```

(continues on next page)

(continued from previous page)

```
components:
  star:
    class: spexxy.component.GridComponent
    interpolator: phx
    init:
    - class: spexxy.init.InitFromValues
      values:
        logg: 4.5
        Alpha: 0.
        v: 0
        sig: 0

main:
  class: spexxy.main.ParamsFit
  components: [star]
  fixparams:
    star: [sig, Alpha, logg]
```

This works, because instead of defining all parameter objects directly in the configuration of a given class, *spexxy* also supports referencing. The GridComponent requires for its *interpolator* parameter either an object of type *Interpolator*, or a definition in form of a dictionary containing a *class* element, or the name of an object that is defined with the *interpolators* of the configuration. Same works for ParamsFit, which accepts a reference to the component named *star*, which is defined in the *components* block.

CHAPTER 2

Parameter fits

2.1 Configuration files

2.1.1 General structure

A configuration for spexxy must be provided as YAML file, e.g. something like this:

```
grids:
  phxgrid:
    class: spexxy.grid.FilesGrid
    filename: /path/to/grid.csv

interpolators:
  phx:
    class: spexxy.interpolator.LinearInterpolator
    grid: phxgrid

components:
  star:
    class: spexxy.component.GridComponent
    interpolator: phx
    init:
    - class: spexxy.init.InitFromValues
      values:
        logg: 4.5
        Alpha: 0.
        v: 0
        sig: 0

main:
  class: spexxy.main.ParamsFit
  components: [star]
  fixparams:
    star: [sig, Alpha, logg]
```

From this example, two observations can be made, which will both be discussed in more detail later:

1. The configuration has several top-level entries (grids, interpolators, ...), of which one – main – has a special meaning.
2. At several places, Python class names are given. Objects from these classes are created automatically during runtime, which allows for changing the behaviour of the fits.

2.1.2 Object creation

In the above example, several Python class names are given, from which objects are automatically created during runtime. Values on the same level as the class definition are passed to the class' constructor, e.g.:

1. The class *spexxy.grid.FilesGrid* has a constructor with a single parameter, *filename*, which is given in the config.
2. The class *spexxy.component.GridComponent* has two parameters (*interpolator*, *name*) in its constructor and gets four more from its parent class *spexxy.component.SpectrumComponent* and three more from its parent (*spexxy.component.Component*) in turn (note that *name* is just passed through). Except for *interpolator* all parameters are optional, so that one must be given in the config. According to the definition in *spexxy.component.Component*, *init* must be a list of `spexxy.init.init.Init` objects, which is provided.

In the last case, the value provided for *interpolator* is a string, and not a *spexxy.interpolator.Interpolator* as required. This works, because *spexxy* usually accepts objects in three forms:

1. An object of the required type directly.
2. A dictionary with a *class* element describing the object.
3. A name of an object described somewhere else. It searches for the name in given groups, which are represented by top-level entries in the config.

2.1.3 Top-level entries

Only one of the top-level entries in the configuration files must always exist: *main*. It describes a sub-class of *spexxy.main.MainRoutine* and is the entry point into any (fitting) routine in *spexxy*.

All the other top-level entries define groups, which are mainly used by the main routine *spexxy.main.ParamsFit*. As described in the section above, objects can be referenced by their name, which is especially interesting, if they are supposed to be used multiple times, e.g. two different components could use the same interpolator.

Currently five different groups are in use:

- grids
- interpolators
- components
- weights
- masks

They can always be used in place of parameters with the same name. In the example at the top of this page this would be:

1. The *grid* parameter for *spexxy.interpolator.LinearInterpolator* refers to an object in the *grids* group with the name *phxgrid*.

2. The *interpolator* parameter for `spexxy.component.GridComponent` points to the interpolator in the 'interpolators' group.
3. The *components* parameter for `spexxy.main.ParamsFit` contains a list with a single name of an element in the *components* group.

CHAPTER 3

spexxy tools

3.1 filters

The methods in *spexxytools filters* can be used to apply filters.

3.1.1 limbdark

With this command one can apply a given filter to either a single specific intensity spectra or to a grid of those:

```
usage: spexxytools filters limbdark [-h] [-g] input output filter

positional arguments:
  input       Spectrum or grid file (in --grid mode)
  output      Output filename or directory (in --grid mode)
  filter      Filter name or filename

optional arguments:
  -h, --help  show this help message and exit
  -g, --grid  Process all files in grid file given as input
```

If *–grid* is given, input must be a grid file and output a directory, otherwise it's input and output filename for a single spectrum.

3.2 grid

The methods in *spexxytools grid* can be used to create and manipulate *FilesGrid* grids.

3.2.1 info

Information about an existing grid can be inspected using *spexxytools grid info* with a grid configuration as paramater:

```
usage: spexxytools grid info [-h] [-v] config

positional arguments:
  config        Config for grid

optional arguments:
  -h, --help    show this help message and exit
  -v, --values  Show all values on axes (default: False)
```

The file specified in the parameters must contain the configuration for a grid (like) object, e.g. something like this:

```
class: spexxy.grid.FilesGrid
filename: /path/to/grid.csv
```

3.2.2 create

Using *spexxytools grid create* a new *FilesGrid* can be created from files in a directory:

```
usage: spexxytools grid create [-h] [-o OUTPUT] [-p PATTERN]
                               [--from-filename FROM_FILENAME]
                               root

positional arguments:
  root                  Path to create grid from

optional arguments:
  -h, --help            show this help message and exit
  -o OUTPUT, --output OUTPUT
                        Output database file (default: grid.csv)
  -p PATTERN, --pattern PATTERN
                        Filename pattern (default: **/*.fits)
  --from-filename FROM_FILENAME
                        Parse parameters from the filenames with the given
                        regular expression (default: None)
```

For example, the name of a spectrum in the PHOENIX library looks something like this:

```
lte05800-4.50-0.0.PHOENIX-ACES-AGSS-COND-2011-HiRes.fits
```

The numbers in the filename are Teff, logg and [M/H], respectively. For alpha element abundances [alpha/Fe]!=0, another term appears like this:

```
lte05800-4.50-0.0.Alpha=+0.50.PHOENIX-ACES-AGSS-COND-2011-HiRes.fits
```

Since all parameters are encoded in the filename, a grid can easily be defining a regular expression that extracts these parameters from the filenames. Using the spexxytools this can be done as:

```
spexxytools grid create --from-filename "lte(?P<Teff>\d{5})-(?P<logg>\d\.\d\d)(?P<FeH>
↪[+-]\d\.\d)(\.Alpha=(?P<Alpha>[+-]\d\.\d\d))?\.PHOENIX" .
```

Note that the groups are named (via ?P<name>) and that those names will be used as names for the parameters.

This spexxytools call will produce an output CSV file that might look like this:

```
Filename,Teff,logg,FeH,Alpha
PHOENIX-ACES-AGSS-COND-2011/Z+0.5.Alpha=+0.50/lte02300-0.00+0.5.Alpha=+0.50.PHOENIX-
↪ACES-AGSS-COND-2011-HiRes.fits,2300.0,0.0,0.5,0.5
PHOENIX-ACES-AGSS-COND-2011/Z+0.5.Alpha=+0.50/lte02300-0.50+0.5.Alpha=+0.50.PHOENIX-
↪ACES-AGSS-COND-2011-HiRes.fits,2300.0,0.5,0.5,0.5
[...]
```

3.2.3 fillholes

The Interpolators in *spexxy* work best on grids with a convex shape without any holes. With *spexxytools grid fillholes* there is a tool for filling holes in a grid using a cubic spline interpolator:

```
usage: spexxytools grid fillholes [-h] grid holes output

positional arguments:
  grid        Name of grid CSV
  holes       Name of holes CSV
  output      Directory to store interpolated spectra in

optional arguments:
  -h, --help  show this help message and exit
```

The CSV file containing a list of holes can be created using *spexxytools grid findholes*.

3.2.4 findholes

Finds holes in a grid that make in non-convex:

```
usage: spexxytools grid findholes [-h] grid output

positional arguments:
  grid        Name of grid CSV
  output      Output file with holes

optional arguments:
  -h, --help  show this help message and exit
```

3.3 isochrone

The methods in *spexxytools isochrone* can be used to manipulate and apply isochrones.

3.3.1 apply

With this command one can apply a given isochrone to all stars in a given photometry file. The tool derives effective temperatures, surface gravities and current masses for all entries:

```
usage: spexxytools isochrone apply [-h] [-o OUTPUT] [--id ID [ID ...]] [--filter1␣
↪FILTER1] [--filter2 FILTER2] [--filter1-iso FILTER1_ISO] [--filter2-iso FILTER2_
↪ISO] [--nearest] [--check-teff CHECK_TEFF CHECK_TEFF] [--check-logg CHECK_LOGG␣
↪CHECK_LOGG] [--check-mact CHECK_MACT CHECK_MACT]
                                  [--quadratic | --cubic]
```

(continues on next page)

(continued from previous page)

```
                                    isochrone photometry

positional arguments:
  isochrone             File containing the isochrone
  photometry            File containing the photometry

optional arguments:
  -h, --help            show this help message and exit
  -o OUTPUT, --output OUTPUT
                        Output file
  --id ID [ID ...]      IDs from photometry file to copy to the output
  --filter1 FILTER1     First filter to use
  --filter2 FILTER2     Second filter to use
  --filter1-iso FILTER1_ISO
                        First filter in isochrone to use (--filter1 if empty)
  --filter2-iso FILTER2_ISO
                        Second filter in isochrone to use (--filter2 if empty)
  --nearest             Use nearest neighbour instead of polynomial
  --check-teff CHECK_TEFF CHECK_TEFF
                        Only apply values if Teff within range
  --check-logg CHECK_LOGG CHECK_LOGG
                        Only apply values if logg within range
  --check-mact CHECK_MACT CHECK_MACT
                        Only apply values if Mact within range
  --quadratic           Use quadratic polynomial instead of linear one
  --cubic               Use cubic polynomial instead of linear one
```

3.3.2 cut

Tries to automatically cut an isochrone to given regions (MS, RGB, . . .):

```
usage: spexxytools isochrone cut [-h] [-p] input output {MS,SGB,RGB,HB,AGB} [{MS,SGB,
↪RGB,HB,AGB} ...]

positional arguments:
  input                 Input file containing the isochrone
  output                Output file
  {MS,SGB,RGB,HB,AGB}   Regions to include

optional arguments:
  -h, --help            show this help message and exit
  -p, --plot            Plot result
```

3.3.3 interpolate

Interpolates an isochrone along a given parameter:

```
usage: spexxytools isochrone interpolate [-h] [--count COUNT] [--column COLUMN] [--
↪space SPACE SPACE] [-p] input output

positional arguments:
  input                 Input file containing the isochrone
  output                Output file
```

(continues on next page)

(continued from previous page)

```
optional arguments:
  -h, --help            show this help message and exit
  --count COUNT         Number of points for new isochrone
  --column COLUMN       Column used for interpolation
  --space SPACE SPACE   Space for distance calculations
  -p, --plot            Plot result
```

3.4 lsf

The methods in *spexxytools lsf* are used for handling line spread functions.

3.4.1 apply

Using *spexxytools lsf apply* one can apply a given LSF to a given spectrum:

```
usage: spexxytools lsf apply [-h] lsf spectrum output

positional arguments:
  lsf         LSF to convolve with
  spectrum    spectrum to convolve
  output      output file

optional arguments:
  -h, --help  show this help message and exit
```

3.5 spectrum

The methods in *spexxytools spectrum* are used for handling spectra.

3.5.1 extract

With this tool a given wavelength range can be extracted from a list of spectra:

```
usage: spexxytools spectrum extract [-h] [-p PREFIX] start end input [input ...]

positional arguments:
  start                 Start of wavelength range
  end                   End of wavelength range
  input                 Input spectrum

optional arguments:
  -h, --help            show this help message and exit
  -p PREFIX, --prefix PREFIX
                        Output file prefix (default: extracted_)
```

3.6 tellurics

The methods in *spexxytools tellurics* are used for handling tellurics.

3.6.1 mean

Calculates a mean tellurics component from TELLURICS extensions in a list of files, e.g. from a ParamsFit run:

```
usage: spexxytools tellurics mean [-h] [-o OUTPUT] [-l SNLIMIT] [-f SNFRAC] [-w] [-r␣
↪RESAMPLE RESAMPLE RESAMPLE] spectra [spectra ...]

positional arguments:
  spectra               List of processed spectra

optional arguments:
  -h, --help            show this help message and exit
  -o OUTPUT, --output OUTPUT
                        File to write tellurics to
  -l SNLIMIT, --snlimit SNLIMIT
                        minimum SNR to use
  -f SNFRAC, --snfrac SNFRAC
                        if snlimit is not given, calculate it from X percent best
  -w, --weight          weight by SNR
  -r RESAMPLE RESAMPLE RESAMPLE, --resample RESAMPLE RESAMPLE RESAMPLE
                        resample to start,step,count
```

CHAPTER 4

API Reference

4.1 Main routines

Main routines are the heart of *spexxy*, describing a specific process that should be run on the data.

There is one pre-defined main routine in *spexxy*:

- *ParamsFit* takes one or more spectrum components and finds the best parameter combination to match a given spectrum.

4.1.1 MainRoutine

class spexxy.main.**MainRoutine**(**args*, ***kwargs*)
MainRoutine is the base class for all main routines.

__call__()
Start the routine.

__init__(**args*, ***kwargs*)
Initialize a new MainRoutine object

4.1.2 FilesRoutine

class spexxy.main.**FilesRoutine**(**args*, ***kwargs*)
FilesRoutine is the base class for all routines fitting files.

__call__(*filename: str*) → List[float]
Start the routine on the given file.

Parameters **filename** – Name of file to process.

Returns List of final values of parameters, ordered in the same way as the return value of parameters()

__init__ (**args*, ***kwargs*)
Initialize a new MainRoutine object

columns () → List[str]
Get list of columns returned by __call__.

The returned list shoud include the list from parameters().

Returns List of columns returned by __call__.

parameters () → List[str]
Get list of parameters fitted by this routine.

Returns List of parameter names (including prefix) fitted by this routine.

4.1.3 ParamsFit

class spexxy.main.**ParamsFit**(*components: List[spexxy.component.component.Component] = None*, *tellurics: spexxy.component.component.Component = None*, *masks: List[spexxy.mask.mask.Mask] = None*, *weights: List[spexxy.weight.weight.Weight] = None*, *fixparams: List[str] = None*, *poly_degree: int = 40*, *maxfev: int = 500*, *ftol: float = 1.49012e-08*, *xtol: float = 1.49012e-08*, *factor: float = 100.0*, *epsfcn: float = 1e-07*, *min_valid_pixels: float = 0.5*, *plot_iterations: bool = False*, **args*, ***kwargs*)
ParamsFit is a fitting routine for spexxy that uses a Levenberg-Marquardt optimization to fit a set of model spectra (components) to a given spectrum.

__call__ (*filename: str*) → List[float]
Start the fitting procedure on the given file.

Parameters **filename** – Name of file to fit.

Returns List of final values of parameters, ordered in the same way as the return value of parameters()

__init__(*components: List[spexxy.component.component.Component] = None*, *tellurics: spexxy.component.component.Component = None*, *masks: List[spexxy.mask.mask.Mask] = None*, *weights: List[spexxy.weight.weight.Weight] = None*, *fixparams: List[str] = None*, *poly_degree: int = 40*, *maxfev: int = 500*, *ftol: float = 1.49012e-08*, *xtol: float = 1.49012e-08*, *factor: float = 100.0*, *epsfcn: float = 1e-07*, *min_valid_pixels: float = 0.5*, *plot_iterations: bool = False*, **args*, ***kwargs*)
Initialize a new ParamsFit object

Parameters

- **components** – List of components (or descriptions to create them) to fit to the spectrum
- **tellurics** – Tellurics component to add to the fit.
- **masks** – List of mask objects to use for masking the spectrum.
- **weights** – List of weight objects to use for creating weights for the spectrum.
- **fixparams** – List of names of parameters to fix during the fit.
- **poly_degree** – Number of coefficients for the multiplicative polynomial.
- **maxfev** – The maximum number of calls to the function (see scipy documentation).
- **ftol** – Relative error desired in the sum of squares (see scipy documentation).
- **xtol** – Relative error desired in the approximate solution (see scipy documentation).

- **factor** – A parameter determining the initial step bound (factor * || diag * x||) (see scipy documentation).
- **epsfcn** – A variable used in determining a suitable step length for the forward- difference approximation of the Jacobian (see scipy documentation)
- **min_valid_pixels** – Fraction of minimum number of required pixels to continue with fit.
- **plot_iterations** – Plot all iterations into a PDF file.

columns() → List[str]
Get list of columns returned by __call__.

The returned list shoud include the list from parameters().

Returns List of columns returned by __call__.

components
Returns all components used in this fit.

Returns List of all components.

fit_parameters() → List[str]
Get list of parameters fitted by this routine. Exclude fixed parameters.

Returns List of parameter names (including prefix) fitted by this routine.

parameters() → List[str]
Get list of parameters fitted by this routine.

Returns List of parameter names (including prefix) fitted by this routine.

4.1.4 MultiMain

class spexxy.main.**MultiMain**(*routines: List[T] = None*, *iterations: int = None*, *max_iterations: int = 8*, *threshold: Dict[str, float] = None*, *poly_degree: int = 40*, *damped: bool = True*, *factors: list = (0.7, 0.3)*, **args*, ***kwargs*)
MultiRun iterates over a given set of main routines and runs them sequentially.

__call__(*filename: str*) → List[float]
Process the given file.

Parameters **filename** – Name of file to process.

Returns List of final values of parameters, ordered in the same way as the return value of parameters()

__init__(*routines: List[T] = None*, *iterations: int = None*, *max_iterations: int = 8*, *threshold: Dict[str, float] = None*, *poly_degree: int = 40*, *damped: bool = True*, *factors: list = (0.7, 0.3)*, **args*, ***kwargs*)
Initialize a new MultiMain object

Parameters

- **routines** – List of main routines to run.
- **iterations** – Number of iterations for the whole cycle.
- **max_iterations** – If set to a value >=2, the fit runs until it converges or until the number of iterations reaches max_iterations.
- **threshold** – Dictionary that contains the absolute values for each fit parameter below which the fit is considered as converged.

- **poly_degree** – Degree of Legendre polynomial used for the continuum fit.
- **damped** – If True the fit is repeated with a damping factor in case the fit does not converge.
- **factors** – List of damping factors used if the fit does not converge.

columns() → List[str]
Get list of columns returned by __call__.

The returned list should include the list from parameters().

Returns List of columns returned by __call__.

convergence(*results*)
Returns true if the fit satisfies the convergence criteria for all fit parameters.

parameters() → List[str]
Get list of parameters fitted by this routine.

Returns List of parameter names (including prefix) fitted by this routine.

4.2 Components

A "Component" in *spexxy* describes a component in a fit, which in general is a collection of parameters and a method to fetch data for a given set of parameter values. Typically, a component build on top of a *Grid* or an *interpolator*.

spexxy comes with a few pre-defined grids:

- `SpectrumComponent` is the base class for all components that serve spectra.
- `StarComponent` contains a single spectrum and adds LOSVD parameters.
- `GridComponent` wraps a grid or an interpolator into a component and adds LOSVD parameters.
- `TelluricsComponent` is just a convenient class derived from *GridComponent* <*spexxy.component.GridComponent*> that changes the default's component name.

4.2.1 Component

class spexxy.component.**Component**(*name: str*, *init: list = None*, *prefix: str = None*, *normalize: bool = False*, **args*, ***kwargs*)
Base class for all Components in spexxy.

__call__(***params*) → Any
Model function, must be implemented.

Parameters **params** – Parameters to retrieve model for

Returns The model from the component

__getitem__(*name: str*) → float
Returns the value for an existing parameter.

Parameters **name** – Name of parameter for which to return the value

Returns Value of given parameter

Raises `KeyError` – If given parameter does not exist

__init__(*name: str*, *init: list = None*, *prefix: str = None*, *normalize: bool = False*, **args*, ***kwargs*)
Initialize a new component.

Parameters

- **name** – Name of component
- **init** – List of Init objects for initializing the component
- **prefix** – Prefix for parameter name when combined in other model. Automatically derived from name if None.
- **normalize** – Whether or not to normalize parameters to 0..1 range

__setitem__(*name: str*, *value: float*)
Returns the value for an existing parameter.

Parameters

- **name** – Name of parameter to set
- **value** – New value for parameter

denorm_param(*name: str*, *value: float*, *stderr: float = None) -> (<class 'float'>*, *<class 'float'>*)
De-Normalize the value of the parameter with the given name to its real value.

Parameters

- **name** – Name of parameter to de-normalize.
- **value** – Value to normalize.
- **stderr** – If given, standard deviation of value.

Returns Normalized value and its standard deviation, if given.

dtype
alias of `numpy.float64`

init(*filename: str*)
Calls all Init objects with the given filename in order to initialize this component.

Parameters **filename** – File to initialize with, may be optional for some Init objects

make_params(***kwargs*) → lmfit.parameter.Parameters
Creates a Parameters object with a Parameter for each parameter of this component.

Parameters **kwargs** – Values to overwrite in parameters

Returns List of Parameters for this component

norm_param(*name: str*, *value: float*) → float
Normalize the value of the parameter with the given name to 0..1 range defined by its min/max.

Parameters

- **name** – Name of parameter to normalize.
- **value** – Value to normalize.

Returns Normalized value.

parse_params(*params: lmfit.parameter.Parameters*)
Loop all Parameters in a Parameters object and set the values of this component accordingly.

Parameters **params** – Parameters objects, usually return from a lmfit optimization

set(*name*, ***kwargs*)
Adds a new parameter with the given name or changes values of an existing one.

Parameters

- **name** (*str*) – Parameter name.

- **value** (*float*) – (Initial) Value of parameter
- **stderr** (*float*) – Standard deviation after fit
- **vary** (*bool*) – Whether or not to vary the parameter in the fit
- **min** (*float*) – Lower bound for fit (default is *-numpy.inf*, no lower bound)
- **max** (*float*) – Upper bound for value (default is *numpy.inf*, no upper bound)
- **kwargs** – Passed to set_param_hint()

write_results_to_file(*fits_file: spexxy.data.fitsspectrum.FitsSpectrum*)
Write results of this component into a given SpectrumFile

Parameters **fits_file** – Opened FitsSpectrum to write results into

4.2.2 SpectrumComponent

class spexxy.component.**SpectrumComponent**(*name: str*, *losvd_hermite: bool = False*, *vac_to_air: bool = False*, *lsf: Union[spexxy.data.lsf.LSF, str, dict] = None*, **args*, ***kwargs*)
SpectrumComponent is the base Component class for all components that deal with spectra.

__call__(***kwargs*)
Model function that creates a spectrum with the given parameters and shift/convolve it using the given LOSVD.

Parameters **kwargs** – Values to overwrite

Returns The model from the component

__init__(*name: str*, *losvd_hermite: bool = False*, *vac_to_air: bool = False*, *lsf: Union[spexxy.data.lsf.LSF, str, dict] = None*, **args*, ***kwargs*)
Initializes a new SpectrumComponent.

Parameters

- **name** – Name of new component
- **losvd_hermite** – Whether or not Hermite polynomials should be used for the LOSVD
- **vac_to_air** – If True, vac_to_air() is called on spectra returned from the model_func
- **lsf** – LSF to apply to spectrum

4.2.3 StarComponent

class spexxy.component.**StarComponent**(*spec: Union[spexxy.data.spectrum.Spectrum, str]*, *name: str = 'STAR'*, **args*, ***kwargs*)

__init__(*spec: Union[spexxy.data.spectrum.Spectrum, str]*, *name: str = 'STAR'*, **args*, ***kwargs*)
Initializes a new Star component that just serves a single given spectrum.

Parameters

- **spec** (*SpectrumComponent | str*) – A Spectrum object or the filename of a spectrum to load
- **name** (*str*) – Name of new component

4.2.4 GridComponent

class spexxy.component.**GridComponent**(*interpolator: spexxy.interpolator.interpolator.Interpolator*, *name: str = 'STAR'*, **args*, ***kwargs*)

A Grid component takes an interpolator and adds LOSVD parameters.

__init__(*interpolator: spexxy.interpolator.interpolator.Interpolator*, *name: str = 'STAR'*, **args*, ***kwargs*)

Initializes a new Grid component.

Parameters

- **interpolator** (*Interpolator*) – The interpolator to use for the component
- **name** (*str*) – Name of the component

4.2.5 TelluricsComponent

class spexxy.component.**TelluricsComponent**(*interpolator: spexxy.interpolator.interpolator.Interpolator*, *name: str = 'TELLURICS'*, **args*, ***kwargs*)

A Component for serving tellurics spectra from a given interpolator.

__init__(*interpolator: spexxy.interpolator.interpolator.Interpolator*, *name: str = 'TELLURICS'*, **args*, ***kwargs*)

Initializes a new Grid component.

Parameters

- **interpolator** (*Interpolator*) – The interpolator to use for the component
- **name** (*str*) – Name of the component

4.3 Grids

A "Grid" in *spexxy* is anything that provides any kind of data in a regularly spaced parameter space. The base class for all Grids is *spexxy.grid.Grid*, which also defines some convenience methods.

The usual way of getting data from a grid is by calling it with the requested parameters:

```
grid = Grid()
data = grid((3.14, 42.))
```

A class inheriting from *Grid* must call Grid's constructor with a list of *GridAxis* objects that describe the axes of the grid, i.e. their names and possbile values. Furthermore it must overwrite all necessary methods, in particular *__call__()*, *__contains__()*, and *all()*. See the implementation of *ValuesGrid* for a simple example.

spexxy comes with two pre-defined grids:

- *ValuesGrid* defines a simple grid, for which the values are defined in its constructor. This grid is mainly used for unit tests.
- *FilesGrid* is a grid, where each "value" is a spectrum in its own file. A CSV file must be provided containing filenames and all parameters.

4.3.1 Grid

class spexxy.grid.**Grid**(*axes: List[spexxy.grid.grid.GridAxis]*, **args*, ***kwargs*)

Base class for all grids in spexxy.

__call__ (*params: Tuple*) → Any
Fetches the value for the given parameter set

Parameters **params** – Parameter set to catch value for.

Returns Grid value at given position.

__contains__ (*params: Tuple*) → bool
Checks, whether the grid contains a given parameter set.

Parameters **params** – Parameter set to check.

Returns Whether or not the given parameter set exists in the grid.

__init__ (*axes: List[spexxy.grid.grid.GridAxis], *args, **kwargs*)
Initialize a new Grid.

Parameters **axes** – List of axes for this grid.

all () → List[Tuple]
Return all possible parameter combinations.

Returns All possible parameter combinations.

axes () → List[spexxy.grid.grid.GridAxis]
Returns information about the axes.

Returns List of GridAxis objects describing the grid's axes.

axis_name (*axis: int*) → str
Returns name of given axis.

Parameters **axis** – Index of axis to return values for.

Returns Name of given axis.

axis_names () → List[str]
Returns names of all axis.

Returns Names of all axis.

axis_values (*axis: int*) → List[float]
Returns all possible values for the given axis.

Parameters **axis** – Index of axis to return values for.

Returns All possible values for the given axis.

create_array () → Any
In case the values provided by this grid are of an array type, this method creates an empty array that can be filled.

Returns Empty array element of same type as values in the grid.

static load (*filename: str*) → spexxy.grid.grid.Grid
Tries to identify type of grid and loads it.

Parameters **filename** – Name of file to load.

nearest (*params*, *scales=None*) → Tuple
Finds the nearest point within the grid.

Calculates distance between given params and all points p in the grid as the sum over all elements in the parameter tuple of ((params - grid_point) * scales)**2.

Parameters

- **params** – Parameters to find nearest point to.
- **scales** – If given, scales dimensions.

Returns Parameters for nearest point in grid.

neighbour(*params: Tuple*, *axis: int*, *distance: int = 1*, *must_exist: bool = False*) → Tuple
Finds a neighbour on the given axis for the given value in the given distance.

Parameters

- **params** – Parameter tuple to search neighbour from.
- **axis** – Axis to search for
- **distance** – Distance in which to find neighbour. >0: Find larger neighbours, i.e. 0 next larger value, 1 the one after that, etc <=0: Find smaller neighbours, i.e. 0 next smaller value (or value itself), -1 the before that, etc
- **must_exist** – Grid point with new parameter set must actually exist.

Returns New parameter tuple with neighbour on the given axis.

num_axes() → int
Returns number of axes.

Returns Number of axes in grid.

4.3.2 GridAxis

class spexxy.grid.**GridAxis**(*name: str*, *values: List[T] = None*, *min: float = None*, *max: float = None*, *initial: float = None*)
Description of a single axis in a grid.

__init__(*name: str*, *values: List[T] = None*, *min: float = None*, *max: float = None*, *initial: float = None*)
Initialize a new grid axis.

Parameters

- **name** – Name of new axis.
- **values** – List of all possible values for this axis.
- **min** – Minimum value for this axis.
- **max** – Maximum value for this axis.
- **initial** – Initial guess for this axis.

__weakref__
list of weak references to the object (if defined)

neighbour(*value: float*, *distance: int = 1*) → float
Finds a neighbour in this axis for the given value in the given distance.

Parameters

- **value** – Value to find neighbour for.
- **distance** – Distance in which to find neighbour >0: Find larger neighbours, i.e. 0 next larger value, 1 the one after that, etc <=0: Find smaller neighbours, i.e. 0 next smaller value (or value itself), -1 the before that, etc

Returns Value on grid in the given distance to the given value. If given value is on grid, distance is counted from that value.

Raises `KeyError` – If no neighbour has been found.

4.3.3 ValuesGrid

class spexxy.grid.**ValuesGrid**(*axes: List[spexxy.grid.grid.GridAxis], values: Dict[Tuple, Any], *args, **kwargs*)

Basic Grid that gets its values from the constructor. Mainly used for testing.

__call__(*params: Tuple*) → Any

Fetches the value for the given parameter set

Parameters **params** – Parameter set to catch value for

Returns Grid value at given position

__contains__(*params: Tuple*) → bool

Checks, whether the grid contains a given parameter set.

Parameters **params** – Parameter set to check.

Returns Whether or not the given parameter set exists in the grid.

__init__(*axes: List[spexxy.grid.grid.GridAxis], values: Dict[Tuple, Any], *args, **kwargs*)

Constructs a new Grid.

Parameters

- **axes** – A list of axes to build the grid from.
- **values** – A dictionary where the keys are the parameters and the values the values at that position in the grid.

all() → List[Tuple]

Return all possible parameter combinations.

Returns All possible parameter combinations.

4.3.4 FilesGrid

class spexxy.grid.**FilesGrid**(*filename: str*, *norm_to_mean: bool = False*, **args*, ***kwargs*)

Grid working on files with a CSV based database.

__call__(*params: Tuple*) → Any

Fetches the value for the given parameter set.

Parameters **params** – Parameter set to catch value for.

Returns Grid value at given position.

__contains__(*params: Tuple*) → bool

Checks, whether the grid contains a given parameter set.

Parameters **params** – Parameter set to check.

Returns Whether or not the given parameter set exists in the grid.

__init__(*filename: str*, *norm_to_mean: bool = False*, **args*, ***kwargs*)

Constructs a new Grid.

Parameters

- **filename** – Filename of CSV file.
- **norm_to_mean** – Normalize spectra to their mean.

all() → List[Tuple]
Return all possible parameter combinations.

Returns All possible parameter combinations.

filename(*params: Tuple*, *absolute: bool = True*) → str
Returns filename for given parameter set.

Parameters

- **params** – Parameter set to catch value for.
- **absolute** – If True, return full absolute path, otherwise relative path within grid.

Returns Filename.

4.3.5 SynspecGrid

class spexxy.grid.**SynspecGrid**(*synspec: str, models: spexxy.grid.grid.Grid, linelist: str, mollist: str, datadir: str, range: Tuple[float, float], vturb: Union[str, float] = 2.0, elements: List[str] = None, input: Union[str, spexxy.grid.grid.Grid] = None, imode: int = 10, idstd: int = 0, iprin: int = 0, inmod: int = 0, intrpl: int = 0, ichang: int = 0, ichemc: int = 1, iophli: int = 0, nunalp: int = 0, nunbet: int = 0, nungam: int = 0, nunbal: int = 0, ifreq: int = 1, inlte: int = 0, icontl: int = 0, inlist: int = 0, ifhe2: int = 0, ihydpr: int = 1, ihe1pr: int = 0, ihe2pr: int = 0, cutof0: int = 40, cutofs: int = 0, relop: float = 1e-05, space: float = 0.03, normalize: bool = False, nstfile: str = 'nstf', nd: int = None, ifmol: int = 1, tmolim: float = None, ippick: int = None, ibfac: int = None, tempdir: str = None, solar_abund: Dict[str, float] = None, *args, **kwargs*)
Synthesizes a new spectrum with Synspec at given grid positions.

__call__(*params: Tuple*) → Any
Fetches the value for the given parameter set.

Parameters **params** – Parameter set to catch value for.

Returns Grid value at given position.

__contains__(*params: Tuple*) → bool
Checks, whether the grid contains a given parameter set.

Parameters **params** – Parameter set to check.

Returns Whether or not the given parameter set exists in the grid.

__init__(*synspec: str, models: spexxy.grid.grid.Grid, linelist: str, mollist: str, datadir: str, range: Tuple[float, float], vturb: Union[str, float] = 2.0, elements: List[str] = None, input: Union[str, spexxy.grid.grid.Grid] = None, imode: int = 10, idstd: int = 0, iprin: int = 0, inmod: int = 0, intrpl: int = 0, ichang: int = 0, ichemc: int = 1, iophli: int = 0, nunalp: int = 0, nunbet: int = 0, nungam: int = 0, nunbal: int = 0, ifreq: int = 1, inlte: int = 0, icontl: int = 0, inlist: int = 0, ifhe2: int = 0, ihydpr: int = 1, ihe1pr: int = 0, ihe2pr: int = 0, cutof0: int = 40, cutofs: int = 0, relop: float = 1e-05, space: float = 0.03, normalize: bool = False, nstfile: str = 'nstf', nd: int = None, ifmol: int = 1, tmolim: float = None, ippick: int = None, ibfac: int = None, tempdir: str = None, solar_abund: Dict[str, float] = None, *args, **kwargs*)
Constructs a new Grid.

Parameters

- **synspec** – Full path to synspec exectuble
- **models** – Grid with model atmospheres
- **linelist** – File with line list
- **mollist** – File with molecular list
- **datadir** – Name of data directory
- **range** – Tuple of start/end wavelenghts
- **vturb** – Either the microturbulence or a CSV file containing a table
- **elements** – List of elements to add as new axis
- **input** – Either the name of a fort.5 file or a Grid or None (in which case an automatic fort.5 will be used)
- **parameters** – Use this fort.55 file instead of the automatically generated one
- **imode**
- **idstd**
- **iprin**
- **inmod**
- **intrpl**
- **ichang**
- **ichemc**
- **iophli**
- **nunalp**
- **nunbet**
- **nungam**
- **nunbal**
- **ifreq**
- **inlte**
- **icontl**
- **inlist**
- **ifhe2**
- **ihydpr**
- **ihe1pr**
- **ihe2pr**
- **cutof0**
- **cutofs**
- **relop**
- **space**

- **normalize** – Normalize spectra
- **nstfile** – Name of file with non-standard flags
- **nd**
- **ifmol**
- **tmolim**
- **ippick**
- **ibfac**
- **tempdir** – Temporary directory. Won't be delete if given.
- **solar_abund** – Dictionary with solar abundances to use.

all() → List[Tuple]
Return all possible parameter combinations.

Returns All possible parameter combinations.

filename(*params: Tuple*) → str
Returns filename for given parameter set.

Parameters **params** – Parameter set to catch value for.

Returns Filename.

4.4 Interpolators

A "Interpolator" in *spexxy* is similar to a *Grid*, but works on a continuous parameter space instead of a discrete one. In fact, many interpolators build on an existing grid and allows for interpolation between grid points.

As with a *Grid*, the usual way of getting data from an interpolator is by calling it with the requested parameters:

```
ip = Interpolator()
data = ip((3.14, 42.))
```

A class inheriting from *Interpolator* must overwrite all necessary methods, in particular *__call__()* and *axes()*.

spexxy comes with three pre-defined interpolators:

- *LinearInterpolator* performs linear interpolation on a given grid.
- *SplineInterpolator* performs a cubic spline interpolation on a given grid.
- *UlyssInterpolator* extracts spectra from interpolator files created for the spectrum fitting package ULySS.

4.4.1 Interpolator

class spexxy.interpolator.**Interpolator**(*cache_level: int = 0*, **args*, ***kwargs*)
Base class for all interpolators in spexxy.

__call__(*params: Tuple*) → Any
Interpolates at the given parameter set

Parameters **params** – Parameter set to interpolate at

Returns Interpolation result at given position

__init__(*cache_level: int = 0*, **args*, ***kwargs*)
Initializes a new interpolator.

Parameters **cache_level** – Level of caching: 0 means off, 1 means only last dimension, 2 is last 2 dimensions and so on. Interpolation might be faster with higher level, but will consume significantly more memory.

axes() → List[spexxy.grid.grid.GridAxis]
Returns information about the axes.

Returns List of GridAxis objects describing the grid's axes

clear_cache()
Clear cache.

4.4.2 LinearInterpolator

class spexxy.interpolator.**LinearInterpolator**(*grid: spexxy.grid.grid.Grid*, **args*, ***kwargs*)
A basic linear interpolator that operates on a given grid.

__call__(*params: Tuple*) → spexxy.data.spectrum.Spectrum
Interpolates at the given parameter set

Parameters **params** – Parameter set to interpolate at

Returns Interpolated spectrum at given position

__init__(*grid: spexxy.grid.grid.Grid*, **args*, ***kwargs*)
Initializes a new linear interpolator.

Parameters **grid** – Grid to interpolate on.

axes() → List[spexxy.grid.grid.GridAxis]
Returns information about the axes.

Returns List of GridAxis objects describing the grid's axes

grid
Returns grid used in this interpolator

Returns Grid used for this interpolator

4.4.3 SplineInterpolator

class spexxy.interpolator.**SplineInterpolator**(*grid: spexxy.grid.grid.Grid*, *derivs: spexxy.grid.grid.Grid = None*, *n: int = 1*, *verbose: bool = False*, **args*, ***kwargs*)
A cubic spline interpolator that operates on a given grid.

__call__(*params: Tuple*) → spexxy.data.spectrum.Spectrum
Interpolates at the given parameter set.

Parameters **params** – Parameter set to interpolate at.

Returns Interpolated spectrum at given position.

__init__(*grid: spexxy.grid.grid.Grid*, *derivs: spexxy.grid.grid.Grid = None*, *n: int = 1*, *verbose: bool = False*, **args*, ***kwargs*)
Initializes a new linear interpolator.

Parameters

- **grid** – Grid to interpolate on.
- **derivs** – If given, contains a second grid at the same parameters as grid, but containg 2nd derivatives for the first axis of the grid.
- **n** – Number of points on each side to use for calculating derivatives.
- **verbose** – If True, output some more logs

axes() → List[spexxy.grid.grid.GridAxis]
Returns information about the axes.

Returns List of GridAxis objects describing the grid's axes.

grid
Returns grid used in this interpolator.

Returns Grid used for this interpolator

4.4.4 UlyssInterpolator

class spexxy.interpolator.**UlyssInterpolator**(*filename: str*, **args*, ***kwargs*)
Interpolator that works with Ulyss input files.

__call__(*params: Tuple*) → spexxy.data.spectrum.Spectrum
Interpolates at the given parameter set.

Parameters **params** – Parameter set to interpolate at.

Returns Interpolated spectrum at given position.

__init__(*filename: str*, **args*, ***kwargs*)
Initializes a new Ulyss interpolator.

Parameters **filename** – Name of file containing ulyss interpolator.

axes() → List[spexxy.grid.grid.GridAxis]
Returns information about the axes.

Returns List of GridAxis objects describing the grid's axes.

static create(*files: List[str]*, *output: str*)
Create a new ulyss interpolator from a set of spectrum files.

Parameters

- **files** – List of spectrum files.
- **output** – Output file name.

4.5 Masks

Classes inheriting form *Mask* create good pixel masks for spectra. They are called by the main routine with each component as parameter. So in order to work, derived classes must implement both the constructor *__init__()* and *__call__()*.

spexxy comes with a few pre-defined mask classes:

- *MaskEnds* masks a given number of pixels at both ends of a spectrum.
- *MaskFromPath* loads a mask from a file with the same name in a given directory.

- *MaskRanges* accepts wavelength ranges in its constructor that are then used to create a mask.

4.5.1 Mask

class spexxy.mask.**Mask**(**args*, ***kwargs*)

Mask is the base class for all objects that can create a good pixel mask for spectra.

__call__(*spectrum: spexxy.data.spectrum.Spectrum*, *filename: str*) → numpy.ndarray

Creates a new mask for a spectrum.

Parameters

- **spectrum** – Spectrum to create mask for.
- **filename** – Name of file containing spectrum to create mask for.

Returns Boolean array containing good pixel mask for given spectrum.

__init__(**args*, ***kwargs*)

Initialize a new mask.

4.5.2 MaskEnds

class spexxy.mask.**MaskEnds**(*npixels=10*, **args*, ***kwargs*)

Masks the ends of a spectrum.

This class, when called, creates a mask that masks the N pixels at each end of the given spectrum.

__call__(*spectrum: spexxy.data.spectrum.Spectrum*, *filename: str = None*) → numpy.ndarray

Creates a new mask for the given spectrum masking only the N pixels at each end.

Parameters

- **spectrum** – Spectrum to create mask for.
- **filename** – Name of file containing spectrum to create mask for (unused).

Returns; Boolean array containing good pixel mask for given spectrum.

__init__(*npixels=10*, **args*, ***kwargs*)

Initializes a new mask masking the ends of a spectrum.

Parameters **npixels** – Number of pixels to mask at each end of the spectrum.

4.5.3 MaskFromPath

class spexxy.mask.**MaskFromPath**(*path: str*, *fits_extension: str = 'GOODPIXELS'*, **args*, ***kwargs*)

Reads a pre-calculated mask from another file of a given name.

This class, when called, searches for a file of the given name in the directory specified in the configuration. If it exists, the extension of the given name (defaults to "GOODPIXELS") is read, converted into a Boolean array, and returned. If it doesn't exist, an empty mask is returned.

__call__(*spectrum: spexxy.data.spectrum.Spectrum*, *filename: str*) → numpy.ndarray

Creates a new mask for a spectrum from a file of the same name in a given directory.

Parameters

- **spectrum** – Spectrum to create mask for.

- **filename** – Name of file containing spectrum to create mask for.

Returns Boolean array containing good pixel mask for given spectrum.

__init__(*path: str*, *fits_extension: str = 'GOODPIXELS'*, **args*, ***kwargs*)
Initializes a new mask from a file in a given path.

Parameters

- **path** – Path to search in for file containing mask.
- **fits_extension** – FITS extension to read mask from.

4.5.4 MaskNegative

class spexxy.mask.**MaskNegative**(**args*, ***kwargs*)
Masks negative fluxes in a spectrum.

This class, when called, creates a mask that masks all negative pixels in the given spectrum.

__call__(*spectrum: spexxy.data.spectrum.Spectrum*, *filename: str = None*) → numpy.ndarray
Creates a new mask for the given spectrum masking all negative pixels.

Parameters

- **spectrum** – Spectrum to create mask for.
- **filename** – Name of file containing spectrum to create mask for (unused).

Returns; Boolean array containing good pixel mask for given spectrum.

__init__(**args*, ***kwargs*)
Initializes a new mask for masking negative pixels in a spectrum.

4.5.5 MaskRanges

class spexxy.mask.**MaskRanges**(*ranges: list*, *vrad: Union[str, float] = None*, *component: str = None*, *vrad_parameter: str = None*, **args*, ***kwargs*)
Masks ranges in spectra.

This class, when called, creates a mask on the given wavelength ranges

__call__(*spectrum: spexxy.data.spectrum.Spectrum*, *filename: str = None*) → numpy.ndarray
Creates a new dynamic mask for a spectrum in the wavelength ranges given in the configuration and shifted by the current radial velocity of the given component.

Parameters

- **spectrum** – Spectrum to create mask for.
- **filename** – Name of file containing spectrum to create mask for.

Returns Boolean array containing good pixel mask for given spectrum.

__init__(*ranges: list*, *vrad: Union[str, float] = None*, *component: str = None*, *vrad_parameter: str = None*, **args*, ***kwargs*)
Initializes a new mask.

Parameters

- **ranges** – List of tuples defining (start, end) of wavelength ranges to mask.

- **vrad** – Radial velocity to shift by, either a number or the name of a FITS header entry, in which case "-<name>" negates the value.
- **component** – Name of component to read the radial velocity from.
- **vrad_parameter** – Name of parameter in given component to use as radial velocity.

4.6 Weights

Classes inheriting form *Weight* create weights arrays for spectra. They work very similar to *Masks*, but instead of returning a boolean mask they return a float array, containing a weight for every pixel in a spectrum.

spexxy comes with a few pre-defined weight classes:

- *WeightFromSigma* creates weights from the spectrum's SIGMA array.
- *WeightRanges* creates weigths from the given ranges.

4.6.1 Weight

class spexxy.weight.**Weight**(**args*, ***kwargs*)

Weight is the base class for all objects that can create a weight array for spectra.

__call__(*spectrum: spexxy.data.spectrum.Spectrum*, *filename: str*) → numpy.ndarray

Creates a new weight for a spectrum.

Parameters

- **spectrum** – Spectrum to create weight for.
- **filename** – Name of file containing spectrum to create weight for.

Returns Array containing weight for given spectrum.

__init__(**args*, ***kwargs*)

Initialize a new weight.

4.6.2 WeightFromSigma

class spexxy.weight.**WeightFromSigma**(*squared: bool = False*, **args*, ***kwargs*)

Reads the SIGMA extension from the given file and creates weights from it as 1/SIGMA.

This class, when called, loads the SIGMA extension from the given file and returns the weights as 1/SIGMA.

__call__(*spectrum: spexxy.data.spectrum.Spectrum*, *filename: str*) → numpy.ndarray

Creates a new weight for a spectrum from its SIGMA extension.

Parameters

- **spectrum** – Spectrum to create weight for.
- **filename** – Name of file containing spectrum to create weight for.

Returns Array containing weight for given spectrum.

__init__(*squared: bool = False*, **args*, ***kwargs*)

Initializes a new weight.

Parameters **squared** – Return 1/SIGMA**2 instead of 1/SIGMA.

4.6.3 WeightRanges

class spexxy.weight.**WeightRanges**(*ranges: List[Tuple[float, float, float]] = None*, *initial: float = 1.0*, **args*, ***kwargs*)

Creates a weights array from given ranges.

This class, when called, creates a weights array from the given wavelength ranges.

__call__(*spectrum: spexxy.data.spectrum.Spectrum*, *filename: str*) → numpy.ndarray

Creates a new weight for a spectrum from the ranges given in the configuration.

Parameters

- **spectrum** – Spectrum to create weight for.
- **filename** – Name of file containing spectrum to create weight for.

Returns Array containing weight for given spectrum.

__init__(*ranges: List[Tuple[float, float, float]] = None*, *initial: float = 1.0*, **args*, ***kwargs*)

Initializes a new weight.

Parameters

- **ranges** – List of tuples of (wave start, wave end, weight).
- **initial** – Initial value for whole array.

4.6.4 WeightFromSNR

class spexxy.weight.**WeightFromSNR**(*keyword: str = 'HIERARCH SPECTRUM SNRATIO'*, **args*, ***kwargs*)

Reads the S/N ratio from the given file and creates weights from it as 1/SQRT(FLUX/SNR).

This class, when called, loads the SNR from the given file and returns weights from it as 1/SQRT(FLUX/SNR).

__call__(*spectrum: spexxy.data.spectrum.Spectrum*, *filename: str*) → numpy.ndarray

Creates a new weight for a spectrum from its S/N.

Parameters

- **spectrum** – Spectrum to create weight for.
- **filename** – Name of file containing spectrum to create weight for.

Returns Array containing weight for given spectrum.

__init__(*keyword: str = 'HIERARCH SPECTRUM SNRATIO'*, **args*, ***kwargs*)

Initializes a new weight.

Parameters **keyword** – FITS header keyword containing S/N.

4.6.5 WeightFromGrid

class spexxy.weight.**WeightFromGrid**(*filename*, *initial: float = 0.0*, *max_line_depth: float = 0.5*, *center_weight: float = 1.0*, *max_step: int = 1*, *mask_lines: Union[bool, str, List[T]] = True*, *max_change=(300, 0.3)*, **args*, ***kwargs*)

This class loads the weights from a grid depending on the initial values of the fit parameters by linear interpolation. It returns an array containing the weights.

__call__(*spectrum: spexxy.data.spectrum.Spectrum*, *filename: str*) → numpy.ndarray
Creates and returns weight array.

Parameters

- **spectrum** – Spectrum to create weight for.
- **filename** – Name of spectrum file.

Returns Array containing the weight for given spectrum.

__init__(*filename*, *initial: float = 0.0*, *max_line_depth: float = 0.5*, *center_weight: float = 1.0*, *max_step: int = 1*, *mask_lines: Union[bool, str, List[T]] = True*, *max_change=(300, 0.3)*, **args*, ***kwargs*)
Initializes a new weight.

Parameters

- **filename** – Name of grid file.
- **initial** – Initial value for the whole weight array.
- **max_line_depth** – Central pixel for lines with larger line depth are masked out.
- **center_weight** – Factor that increases the weight of the central pixel of each line.
- **max_step** – In iteration steps <= max_step new weights are loaded from the grid.
- **mask_lines** – List of absorption lines that are always masked out in their centers.

4.6.6 WeightFromGridNearest

class spexxy.weight.**WeightFromGridNearest**(*filename*, *initial: float = 0.0*, *max_line_depth: float = 0.5*, *center_weight: float = 1.0*, *max_step: int = 1*, *mask_lines: Union[bool, str, List[T]] = True*, *max_change=(300, 0.3)*, **args*, ***kwargs*)

This class loads the weights from a grid depending on the initial values of the fit parameters by choosing the nearest neighbour in the grid. It returns an array containing the weights.

__call__(*spectrum: spexxy.data.spectrum.Spectrum*, *filename: str*) → numpy.ndarray
Creates and returns weight array.

Parameters

- **spectrum** – Spectrum to create weight for.
- **filename** – Name of spectrum file.

Returns Array containing the weight for given spectrum.

__init__(*filename*, *initial: float = 0.0*, *max_line_depth: float = 0.5*, *center_weight: float = 1.0*, *max_step: int = 1*, *mask_lines: Union[bool, str, List[T]] = True*, *max_change=(300, 0.3)*, **args*, ***kwargs*)
Initializes a new weight.

Parameters

- **filename** – Name of grid file.
- **initial** – Initial value for the whole weight array.
- **max_line_depth** – Central pixel for lines with larger line depth are masked out.
- **center_weight** – Factor that increases the weight of the central pixel of each line.

- **max_step** – In iteration steps <= max_step new weights are loaded from the grid.
- **mask_lines** – List of absorption lines that are always masked out in their centers.

4.7 Inits

Classes derived from the *Init* class initialize the parameters of components. After creating an Init, it is applied to a component by calling it with the component and a filename as parameters. Therefore a derived class must implement both the constructor *__init__()* and *__call__()*.

spexxy comes with a few pre-defined init classes:

- *InitFromCsv* reads initial values from a CSV file.
- *InitFromPath* looks for a file of the same name in a given path and reads the initial values from its FITS header.
- *InitFromValues* takes initial values directly from its constructor.
- *InitFromVhelio* takes coordinates and time from the FITS header, and calculates the heliocentric or baryiocentric correction, which then can be set as initial value for a component.

4.7.1 Init

class spexxy.init.**Init**(**args*, ***kwargs*)

Init is the base class for all objects that can initialize values of a component.

__call__(*cmp: spexxy.component.component.Component*, *filename: str*)

Initializes values for the given component.

Parameters

- **cmp** (*Component*) – Component to initialize.
- **filename** (*str*) – Name of file containing spectrum to create mask for.

__init__(**args*, ***kwargs*)

Initialize an Init object.

4.7.2 InitFromCsv

class spexxy.init.**InitFromCsv**(*filename: str = 'initials.csv'*, *filename_col: str = 'Filename'*, *parameters: list = None*, *cmp_sep: str = ' '*, **args*, ***kwargs*)

Initializes a component from a line in a CSV file.

This class, when called, initializes the given parameters (or all if None) of a given component to the values in the given CSV file.

__call__(*cmp: spexxy.component.component.Component*, *filename: str*)

Initializes parameters of the given component with values from the CSV given in the configuration.

Parameters **cmp** – Component to initialize.

filename: Filename of spectrum.

__init__(*filename: str = 'initials.csv'*, *filename_col: str = 'Filename'*, *parameters: list = None*, *cmp_sep: str = ' '*, **args*, ***kwargs*)

Initializes a new Init object.

Parameters

- **filename** – Name of CSV file.
- **filename_col** – Name of column containing filename.
- **parameters** – List of parameter names to set from CSV.
- **cmp_sep** – String separating component and parameter name in CSV.

4.7.3 InitFromPath

class spexxy.init.**InitFromPath**(*path: str*, **args*, ***kwargs*)

Initializes a component from another file in a given directory.

This class, when called, initializes the parameters of a given component to the values in the header of a file with the same name (usually written in a previous fit) in the given directory.

__call__(*cmp: spexxy.component.component.Component*, *filename: str*)

Initializes parameters of the given component with values from another file.

Parameters

- **cmp** – Component to initialize.
- **filename** – Name of file (in given path) to read initial values from.

__init__(*path: str*, **args*, ***kwargs*)

Initializes a new Init object.

Parameters **path** – Path in which to look for the file to read the initial values from.

4.7.4 InitFromValues

class spexxy.init.**InitFromValues**(*values: dict*, **args*, ***kwargs*)

Initializes a component from given values.

This class, when called, initializes the parameters of a given component to the provided values from a dict.

__call__(*cmp: spexxy.component.component.Component*, *filename: str*)

Initializes parameters of the given component with values from the configuration

Parameters

- **cmp** – Component to initialize.
- **filename** – Unused.

__init__(*values: dict*, **args*, ***kwargs*)

Initializes a new Init object.

Parameters **values** – Dictionary of key/value pairs defining initial values for the component's parameters.

4.7.5 InitFromVhelio

class spexxy.init.**InitFromVhelio**(*negative: bool = False*, *parameter: str = 'v'*, *scale: str = 'utc'*, *obs: str = 'paranal'*, *kind: str = 'barycentric'*, **args*, ***kwargs*)

Initializes a component from the heliocentric correction calculated for RA/Dec given in the FITS headers.

This class, when called, initializes a single parameter (the radial velocity, default to "v") of the given component with the heliocentric correction calculated from RA/DEC/DATE-OBS in the given file.

__call__(*cmp: spexxy.component.component.Component*, *filename: str*)

Initializes the radial velocity parameter of a given component to the (negative) heliocentric correction.

Parameters

- **cmp** – Component to initialize.
- **filename** – Name of file containing RA/DEC/DATE-OBS in FITS header to calculate correction from.

__init__(*negative: bool = False*, *parameter: str = 'v'*, *scale: str = 'utc'*, *obs: str = 'paranal'*, *kind: str = 'barycentric'*, **args*, ***kwargs*)

Initializes a new Init object.

Parameters

- **negative** – If True, uses the negative of the calculated correction.
- **parameter** – Name of the parameter in the component to set.
- **scale** – Time scale to use for DATE-OBS.
- **obs** – Observatory to use for calculating correction.
- **kind** – Either barycentric or heliocentric.

Index

Symbols

A

C

D

F

G

I

L

M

N

P

S

T

U

V

W