PyReduce Documentation

Release 0.4.7+234.g7112f52.dirty

Ansgar Wehrhahn

Jan 10, 2022

Contents:

CHAPTER 1

Installation

There are two main ways to install Pyreduce. For the latest stable version simply use `pip install pyreduce-astro`

Alternatively the latest development version can be installed from github using `pip install git+https://github.com/AWehrhahn/PyReduce`

CHAPTER 2

How To use PyReduce

Using PyReduce is easy. Simply specify the instrument and where to find the data, and the rest should take care of itself. Of course you can also set all parameters yourself if you want more control over what is happening.

A good starting point is the examples section. But here is what you need.

```
>>> import pyreduce
>>> from pyreduce import datasets
```

Define parameters

```
>>> # The instrument name as specified in the supported instruments
>>> instrument = "UVES"
>>> # The name of the observation target as specified in the file structure
>>> target = "HD132205"
>>> # The observation night as a string, as specified in the file structure
>>> night = "2010-04-02"
>>> # The instrument mode/setting that is used, depends on the instrument
>>> mode = "middle"
>>> # The data reduction steps to run
>>> steps = (
    "bias",
    "flat",
    "orders",
    "norm_flat",
    "wavecal",
    "freq_comb",
    "shear",
    "science",
    "continuum",
    "finalize",
    )
```

Some basic settings Expected Folder Structure: `base_dir/datasets/HD132205/*.fits.gz` Feel free to change this to your own preference, values in curly brackets will be replaced with the actual values {}

```
>>> input_dir = "{target}/"
>>> output_dir = "reduced/{instrument}/{target}/{night}/{mode}"
```

Load dataset (and save the location)

For the example dataset use

```
>>> base_dir = datasets.UVES_HD132205()
```

For your own observations set base_dir to the path that points to your files. Note that the full path is given by base_dir + input_dir / output_dir. If these are completely independant you can set base_dir = “” instead.

```
>>> base_dir = "your-file-path-here"
```

Start the extraction

```
>>> pyreduce.reduce.main(
    instrument,
    target,
    night,
    mode,
    steps,
    base_dir=base_dir,
    input_dir=input_dir,
    output_dir=output_dir,
    configuration="settings_UVES.json",
    )
```

CHAPTER 3

Examples

The PyReduce distribution includes one example dataset for the UVES spectrograph. If everything is set up correctly, it can be run by simply calling uves_example.py in the examples directory.

```
>>> python examples/uves_example.py
```

This will download the necessary data and run all extraction steps. Inside the script, the individual steps that are executed can be changed by modyfying the "steps" list, e.g. by commenting some entries out.

```
>>> steps = (
    "bias",
    "flat",
    # "orders",
    # "norm_flat",
    # "wavecal",
    # "shear",
    # "science",
    # "continuum",
    # "finalize",
    )
```

The output files will be placed in /examples/dataset/UVES/...

CHAPTER 4

PyReduce Configuration

All free parameters of the PyReduce pipeline are defined in a configuration file, that is passed to the main function. If certain values are not explicitly defined, default values will be used, which may or may not work well. Some configurations for common instruments are provided in the examples directory.

All input is validated using the jsonschema `settings/settings_schema.json`.

The default values are defined as:

Listing 1: settings_pyreduce.json

```
{
    "$schema": "./settings_schema.json",
    "__instrument__": "DEFAULT",
    "reduce": {
        "base_dir": "./",
        "input_dir": "raw",
        "output_dir": "reduced"
    },
    "instrument": {},
    "mask": {},
    "bias": {
        "degree": 0,
        "plot": true,
        "plot_title": "Bias"
    },
    "flat": {
        "bias_scaling": "number_of_files",
        "norm_scaling": "none",
        "plot": true,
        "plot_title": "Flat"
    },
    "orders": {
        "degree": 4,
        "degree_before_merge": 2,
        "bias_scaling": "number_of_files",
```

(continues on next page)

(continued from previous page)


```
        "norm_scaling": "none",
        "regularization": 0,
        "closing_shape": [
            5,
            5
        ],
        "auto_merge_threshold": 0.9,
        "merge_min_threshold": 0.1,
        "split_sigma": 0,
        "filter_size": null,
        "min_cluster": null,
        "min_width": null,
        "noise": null,
        "border_width": null,
        "manual": true,
        "plot": true,
        "plot_title": "Order Tracing"
    },
    "scatter": {
        "scatter_degree": 4,
        "scatter_cutoff": 2,
        "border_width": null,
        "extraction_width": 0.3,
        "bias_scaling": "number_of_files",
        "norm_scaling": "divide",
        "plot": true,
        "plot_title": "Background Scatter"
    },
    "norm_flat": {
        "extraction_method": "normalize",
        "smooth_slitfunction": 4,
        "smooth_spectrum": 1e-7,
        "oversampling": 10,
        "maxiter": 20,
        "swath_width": 200,
        "extraction_width": 0.2,
        "threshold": 0.6,
        "threshold_lower": 0,
        "extraction_cutoff": 20,
        "plot": true,
        "plot_title": "Normalized Flat"
    },
    "wavecal_master": {
        "extraction_method": "arc",
        "collapse_function": "median",
        "extraction_width": 0.5,
        "extraction_cutoff": 20,
        "bias_scaling": "number_of_files",
        "norm_scaling": "divide",
        "plot": true,
        "plot_title": "Wavelength Calibration Spectrum"
    },
    "wavecal_init": {
        "degree": 2,
        "element": "thar",
        "medium": "vac",
        "wave_delta": 20,
```


(continues on next page)

(continued from previous page)

```
        "nwalkers": 100,
        "steps": 50000,
        "resid_delta": 1000,
        "cutoff": 0.01,
        "smoothing": 0,
        "plot": true,
        "plot_title": "Wavelength Calibration Initial"
    },
    "wavecal": {
        "manual": false,
        "threshold": 100,
        "iterations": 3,
        "dimensionality": "2D",
        "degree": [
            6,
            6
        ],
        "nstep": 0,
        "shift_window": 0.01,
        "element": "thar",
        "medium": "vac",
        "plot": true,
        "plot_title": "Wavelength Calibration"
    },
    "freq_comb_master":{
        "extraction_method": "arc",
        "collapse_function": "median",
        "extraction_width": 0.5,
        "extraction_cutoff": 20,
        "bias_scaling": "number_of_files",
        "norm_scaling": "divide",
        "plot": true,
        "plot_title": "Frequency Comb Spectrum"
    },
    "freq_comb": {
        "lfc_peak_width": 3,
        "dimensionality": "2D",
        "nstep": 0,
        "degree": [
            6,
            6
        ],
        "threshold": 100,
        "plot": true,
        "plot_title": "Frequency Comb"
    },
    "curvature": {
        "dimensionality": "1D",
        "degree": 2,
        "curv_degree": 2,
        "extraction_method": "arc",
        "collapse_function": "median",
        "extraction_width": 0.2,
        "curvature_cutoff": 3,
        "extraction_cutoff": 20,
        "peak_threshold": 10,
        "peak_width": 1,
```

(continues on next page)

(continued from previous page)

```
        "window_width": 9,
        "peak_function": "gaussian",
        "bias_scaling": "number_of_files",
        "norm_scaling": "divide",
        "plot": true,
        "plot_title": "Slit Curvature"
    },
    "rectify": {
        "extraction_width": 0.5,
        "input_files": "science",
        "plot": true,
        "plot_title": "Rectified Image"
    },
    "science": {
        "extraction_method": "optimal",
        "extraction_width": 0.25,
        "oversampling": 10,
        "swath_width": 300,
        "maxiter": 20,
        "smooth_slitfunction": 0.1,
        "smooth_spectrum": 1e-7,
        "extraction_cutoff": 20,
        "bias_scaling": "number_of_files",
        "norm_scaling": "divide",
        "plot": true,
        "plot_title": "Science Data"
    },
    "continuum": {
        "plot": true,
        "plot_title": "Continuum Normalization"
    },
    "finalize": {
        "filename": "{instrument}.{night}_{number}.final.ech",
        "plot": true,
        "plot_title": "Final Science Product"
    }
}
```

CHAPTER 5

Supporting Custom instruments

PyReduce supports a number of instruments by default, but it can be extended to support your instrument too. There are two ways to implement your instrument. The first works on a script by script basis. The second is a more complete ompelementation. The second choice is recommended for proper use, although the first can be useful for experimentation.

5.1 Method 1: Create your instrument as part of the script

There is a very simple example in the examples folder for the support of a custom instrument "custom_instrument_example.py". The idea is that we create a custom class on the fly and define all the necessary parameters within the script that is currently being run. This is in general less flexible than the second method, but can be useful since all variables are available within the script itself. Once a good solution has been found it is recommended to convert this to what is described below in method 2, since we can then use the same instrument configuration in other scripts.

5.2 Method 2: Create the instrument class and configuration

In this method, we create a custom class and configuration as seperate files, similar to how instruments are implemented in PyReduce itself. The general instrument is defined by the pyreduce.instruments.common.Instrument class, so the easiest way is to create your own class that inherits from it.

Here are the general steps that need to be followed to support your instrument:

- You need to implement your own version of the instrument class, that inherits from pyreduce.instruments.common.Instrument
- In that class there are a few important functions that may need to be adapted:
  - load_info, this loads a json file with information mostly about the FITS header (e.g. which keyword gives us the time of the observation etc.) Look at other pyreduce.instruments.instrument_schema to get some information about what is what

  - sort_files, finds and sorts the files for the different steps of pyreduce. There is a system here that might work for your instrument as well, depending on the FITS headers. In that case you just need to set the kw_bias etc fields in load_info correctly
  - add_header_info, this modifies the fits header information. Usually to combine fields in the header to get the correct information (e.g. to get the time in the middle of the observation, instead of at the beginning).
  - get_wavecal_filename, should return the filename to the wavelength calibration first guess. See the Wavelength Calibration section on how to create this file.
  - get_wavelength_range, this returns an initial guess for the wavelength of each order, if the initial first guess file from get_wavecal_filename is not provided.
  - (optional) get_mask_filename, should return the filename of the bad pixel map. A fits file with the badpixel map in the main extension. With the same size as the input fits files

- You probably also want to override the settings used by PyReduce (config in the example scripts). You can find examples for settings in pyreduce.settings. (settings_pyreduce.json has all available settings, they all need to be specified)
- When calling PyReduce instead of passing the name of the instrument you pass an instance of your instrument class.

CHAPTER 6

PyReduce

6.1 pyreduce package

6.1.1 Subpackages

pyreduce.clib package

Submodules

pyreduce.clib.build_extract module

Builds the C library that contains the extraction algorithm

This module prepares and builds the C libary containing the curved (and vertical) extraction algorithm using CFFI. It also prepares the ffibuilder objects for setup.py, so that the library is compiled on installation.

The user can also call the Module as a script to compile the C libraries again.

`pyreduce.clib.build_extract.`**`ffi_builder_vertical`**
CFFI Builder for the vertical extraction algorithm

Type FFI

`pyreduce.clib.build_extract.`**`ffi_builder_curved`**
CFFI Builder for the curved extraction algorithm

Type FFI

`pyreduce.clib.build_extract.`**`build`**`()`
Builds the C slitfunc library

Module contents

pyreduce.instruments package

Submodules

pyreduce.instruments.common module

Abstract parent module for all other instruments Contains some general functionality, which may be overridden by the children of course

class pyreduce.instruments.common.**COMMON**
Bases: *pyreduce.instruments.common.Instrument*

class pyreduce.instruments.common.**Instrument**
Bases: object

Abstract parent class for all instruments Handles the handling of instrument specific information

add_header_info(*header*, *mode*, ***kwargs*)
read data from header and add it as REDUCE keyword back to the header

Parameters

- **header** (*fits.header, dict*) – header to read/write info from/to
- **mode** (*str*) – instrument mode

Returns **header** – header with added information

Return type fits.header, dict

apply_filters(*files*, *expected*, *allow_calibration_only=False*)
Determine the relevant files for a given set of expected values.

Parameters

- **files** (*list(files)*) – list if fits files
- **expected** (*dict*) – dictionary with expected header values for each reduction step

Returns **files** – list of files. The first element of each tuple is the used setting, and the second are the files for each step.

Return type list((dict, dict))

find_files(*input_dir*)
Find fits files in the given folder

Parameters **input_dir** (*string*) – directory to look for fits and fits.gz files in, may include bash style wildcards

Returns **files** – absolute path filenames

Return type array(string)

get(*key*, *header*, *mode*, *alt=None*)

get_expected_values(*target*, *night*, **args*, ***kwargs*)

get_extension(*header*, *mode*)

get_mask_filename(*mode*, ***kwargs*)

get_supported_modes()

get_wavecal_filename(*header*, *mode*, ***kwargs*)
Get the filename of the pre-existing wavelength solution for the current setting

Parameters

- **header** (*fits.header, dict*) – header of the wavelength calibration file
- **mode** (*str*) – instrument mode

Returns **filename** – name of the wavelength solution file

Return type str

get_wavelength_range(*header*, *mode*, ***kwargs*)

info = None
Information about the instrument

Type dict

load_fits(*fname*, *mode*, *extension=None*, *mask=None*, *header_only=False*, *dtype=None*)
load fits file, REDUCE style

primary and extension header are combined modeinfo is applied to header data is clipnflipped mask is applied

Parameters

- **fname** (*str*) – filename
- **instrument** (*str*) – name of the instrument
- **mode** (*str*) – instrument mode
- **extension** (*int*) – data extension of the FITS file to load
- **mask** (*array, optional*) – mask to add to the data
- **header_only** (*bool, optional*) – only load the header, not the data
- **dtype** (*str, optional*) – numpy datatype to convert the read data to

Returns

- **data** (*masked_array*) – FITS data, clipped and flipped, and with mask
- **header** (*fits.header*) – FITS header (Primary and Extension + Modeinfo)
- *ONLY the header is returned if header_only is True*

load_info()
Load static instrument information Either as fits header keywords or static values

Returns **info** – dictionary of REDUCE names for properties to Header keywords/static values

Return type dict(str:object)

name = None
Name of the instrument (lowercase)

Type str

populate_filters(*files*)
Extract values from the fits headers and store them in self.filters

Parameters **files** (*list(str)*) – list of fits files

Returns **filters** – list of populated filters (identical to self.filters)

Return type list(Filter)

sort_files(*input_dir*, *target*, *night*, **args*, *allow_calibration_only=False*, ***kwargs*)
Sort a set of fits files into different categories types are: bias, flat, wavecal, orderdef, spec

Parameters

- **input_dir** (*str*) – input directory containing the files to sort
- **target** (*str*) – name of the target as in the fits headers
- **night** (*str*) – observation night, possibly with wildcards
- **mode** (*str*) – instrument mode

Returns

- **files_per_night** (*list[dict{str:dict{str:list[str]}}]*) – a list of file sets, one entry per night, where each night consists of a dictionary with one entry per setting, each fileset has five lists of filenames: "bias", "flat", "order", "wave", "spec", organised in another dict
- **nights_out** (*list[datetime]*) – a list of observation times, same order as files_per_night

class pyreduce.instruments.common.**InstrumentWithModes**
Bases: *pyreduce.instruments.common.Instrument*

get_expected_values(*target*, *night*, *mode*)

pyreduce.instruments.common.**create_custom_instrument**(*name*, *extension=0*, *info=None*, *mask_file=None*, *wavecal_file=None*, *hasModes=False*)

pyreduce.instruments.common.**find_first_index**(*arr*, *value*)
find the first element equal to value in the array arr

class pyreduce.instruments.common.**getter**(*header*, *info*, *mode*)
Bases: object

Get data from a header/dict, based on the given mode, and applies replacements

get(*key*, *alt=None*)
Get data

Parameters

- **key** (*str*) – key of the data in the header
- **alt** (*obj, optional*) – alternative value, if key does not exist (default: None)

Returns **value** – value found in header (or alternatively alt)

Return type obj

pyreduce.instruments.common.**observation_date_to_night**(*observation_date*)
Convert an observation timestamp into the date of the observation night Nights start at 12am and end at 12 am the next day

Parameters **observation_date** (*datetime*) – timestamp of the observation

Returns **night** – night of the observation

Return type datetime.date

pyreduce.instruments.harps module

Handles instrument specific info for the HARPS spectrograph

Mostly reading data from the header

class pyreduce.instruments.harps.**FiberFilter**(*keyword='ESO DPR TYPE'*)
Bases: pyreduce.instruments.filters.Filter

collect(*header*)

class pyreduce.instruments.harps.**HARPS**
Bases: *pyreduce.instruments.common.Instrument*

add_header_info(*header*, *mode*, ***kwargs*)
read data from header and add it as REDUCE keyword back to the header

get_expected_values(*target*, *night*, *mode*, *fiber*, *polarimetry*)
Determine the default expected values in the headers for a given observation configuration

Any parameter may be None, to indicate that all values are allowed

Parameters

- **target** (*str*) – Name of the star / observation target
- **night** (*str*) – Observation night/nights
- **fiber** (*"A", "B", "AB"*) – Which of the fibers should carry observation signal
- **polarimetry** (*"none", "linear", "circular", bool*) – Whether the instrument is used in HARPS or HARPSpol mode and which polarization is observed. Set to true for both kinds of polarisation.

Returns **expectations** – Dictionary of expected header values, with one entry per step. The entries for each step refer to the filters defined in self.filters

Return type dict

Raises ValueError – Invalid combination of parameters

get_extension(*header*, *mode*)

get_wavecal_filename(*header*, *mode*, *polarimetry*, ***kwargs*)
Get the filename of the wavelength calibration config file

get_wavelength_range(*header*, *mode*, ***kwargs*)

class pyreduce.instruments.harps.**PolarizationFilter**(*keyword='ESO INS RET?? POS'*)
Bases: pyreduce.instruments.filters.Filter

collect(*header*)

class pyreduce.instruments.harps.**TypeFilter**(*keyword='ESO DPR TYPE'*)
Bases: pyreduce.instruments.filters.Filter

classify(*value*)

pyreduce.instruments.instrument_info module

Interface for all instrument specific information The actual info is contained in the instruments/{name}.py modules/classes, which are all subclasses of "common"

pyreduce.instruments.instrument_info.**get_instrument_info**(*instrument*)
Load instrument specific information

Parameters **instrument** (*str*) – Name of the instrument

Returns **dict{str** – Dictionary with information

Return type obj}

pyreduce.instruments.instrument_info.**get_supported_modes**(*instrument*)

pyreduce.instruments.instrument_info.**get_wavecal_filename**(*header*, *instrument*, *mode*, ***kwargs*)
Get the filename of the pre-existing wavelength solution for the current settings

Parameters

- **header** (*fits.header, dict*) – header of the wavelength calibration file
- **instrument** (*str*) – instrument name
- **mode** (*str*) – instrument mode (e.g. red/blue for HARPS)

Returns **filename** – wavelength solution file

Return type str

pyreduce.instruments.instrument_info.**load_instrument**(*instrument*) → pyreduce.instruments.common.Instrument
Load an python instrument module

Parameters **instrument** (*str*) – name of the instrument

Returns **instrument** – Instance of the {instrument} class

Return type *Instrument*

pyreduce.instruments.instrument_info.**modeinfo**(*header*, *instrument*, *mode*, ***kwargs*)
Add instrument specific information to a header/dict

Parameters

- **header** (*fits.header, dict*) – header to add information to
- **instrument** (*str*) – instrument name
- **mode** (*str*) – instrument mode (e.g. red/blue for HARPS)

Returns header with added information

Return type header

pyreduce.instruments.instrument_info.**sort_files**(*input_dir*, *target*, *night*, *instrument*, *mode*, ***kwargs*)
Sort a list of files into different categories and discard files that are not used

Parameters

- **input_dir** (*str*) – directory containing all files (with tags for target, night, and instrument)
- **target** (*str*) – observation target name, as found in the files
- **night** (*str*) – observation night of interest, as found in the files
- **instrument** (*str*) – instrument name
- **mode** (*str*) – instrument mode, if applicable (e.g. red/blue for HARPS)

Returns

- **biaslist** (*list(str)*) – list of bias files
- **flatlist** (*list(str)*) – list of flat field files
- **wavelist** (*list(str)*) – list of wavelength calibration files
- **orderlist** (*list(str)*) – list of order definition files (for order tracing)
- **speclist** (*list(str)*) – list of science files, i.e. observations

pyreduce.instruments.uves module

Handles instrument specific info for the UVES spectrograph

Mostly reading data from the header

class pyreduce.instruments.uves.**UVES**
Bases: *pyreduce.instruments.common.Instrument*

add_header_info(*header*, *mode*, ***kwargs*)
read data from header and add it as REDUCE keyword back to the header

get_mask_filename(*mode*, ***kwargs*)

get_wavecal_filename(*header*, *mode*, ***kwargs*)
Get the filename of the wavelength calibration config file

Module contents

6.1.2 Submodules

6.1.3 pyreduce.clipnflip module

Module that

- Clips remove pre- and overscan regions
- Flips orients the image so that orders are roughly horizontal

pyreduce.clipnflip.**clipnflip**(*image*, *header*, *xrange=None*, *yrange=None*, *orientation=None*, *transpose=None*)
Process an image and associated FITS header already in memory as follows: 1. Trim image to desired subregion: newimage = image(xlo:xhi,ylo:yhi) 2. Transform to standard orientation (red at top, orders run left to right)

Parameters

- **image** (*array[nrow, ncol]*) – raw image to be processed.
- **header** (*fits.header, dict*) – FITS header of the image
- **xrange** (*tuple(int, int), optional*) – column - range to keep in the image (default: data from header/instrument)
- **yrange** (*tuple(int, int), optional*) – row - range to keep in the image (default: data from header/instrument)
- **orientation** (*int, optional*) – number of counterclockwise 90 degrees rotation to apply to the image (default: data from header/instrument)
- **transpose** (*bool, optional*) – if True the image will be transposed before rotation (default: data from header/instrument)

Returns **image** – clipped and flipped image

Return type array[yrange, xrange]

Raises `NotImplementedError` – nonlinear images are not supported yet

6.1.4 pyreduce.combine_frames module

Combine several fits files into one master frame

Used to create master bias and master flat

`pyreduce.combine_frames.`**`calculate_probability`**(*buffer*, *window*, *method='sum'*)

Construct a probability function based on buffer data.

Parameters

- **`buffer`** (*`array of shape (nx, ny)`*) – buffer
- **`window`** (*`int`*) – size of the running window
- **`method`** (*`{"sum", "median"}, optional`*) – which method to use to average the probabilities (default: "sum") "sum" is much faster, but "median" is more resistant to outliers

Returns **weights** – probabilities

Return type array of shape (nx, ny - 2 * window)

`pyreduce.combine_frames.`**`combine_bias`**(*files*, *instrument*, *mode*, *extension=None*, *plot=False*, *plot_title=None*, *science_observation_time=None*, ***kwargs*)

Combine bias frames, determine read noise, reject bad pixels. Read noise calculation only valid if both lists yield similar noise.

Parameters

- **`files`** (*`list(str)`*) – bias files to combine
- **`instrument`** (*`str`*) – instrument mode for modinfo
- **`extension`** (*`{int, str}, optional`*) – fits extension to use (default: 1)
- **`xr`** (*`2-tuple(int), optional`*) – x range to use (default: None, i.e. whole image)
- **`yr`** (*`2-tuple(int), optional`*) – y range to use (default: None, i.e. whole image)
- **`dtype`** (*`np.dtype, optional`*) – datatype of the combined bias frame (default: float32)

Returns bias image and header

Return type bias, bhead

`pyreduce.combine_frames.`**`combine_calibrate`**(*files*, *instrument*, *mode*, *mask=None*, *bias=None*, *bhead=None*, *norm=None*, *bias_scaling='exposure_time'*, *norm_scaling='divide'*, *plot=False*, *plot_title=None*, ***kwargs*)

Combine the input files and then calibrate the image with the bias and normalized flat field if provided

Parameters

- **`files`** (*`list`*) – list of file names to load

- **instrument** (Instrument) – PyReduce instrument object with load_fits method
- **mode** (*str*) – descriptor of the instrument mode
- **mask** (*array*) – 2D Bad Pixel Mask to apply to the master image
- **bias** (*tuple(bias, bhead), optional*) – bias correction to apply to the combined image, if bias has 3 dimensions it is used as polynomial coefficients scaling with the exposure time, by default None
- **norm_flat** (*tuple(norm, blaze), optional*) – normalized flat to divide the combined image with after the bias subtraction, by default None
- **bias_scaling** (*str, optional*) – defines how the bias is subtracted, by default "exposure_time"
- **plot** (*bool, optional*) – whether to plot the results, by default False
- **plot_title** (*str, optional*) – Name to put on the plot, by default None

Returns

- **orig** (*array*) – combined image with calibrations applied
- **thead** (*Header*) – header of the combined image

Raises ValueError – Unrecognised bias_scaling option

pyreduce.combine_frames.**combine_frames**(*files*, *instrument*, *mode*, *extension=None*, *threshold=3.5*, *window=50*, *dtype=<class 'numpy.float32'>*, ***kwargs*)

Subroutine to correct cosmic rays blemishes, while adding otherwise similar images.

combine_frames co-adds a group of FITS files with 2D images of identical dimensions. In the process it rejects cosmic ray, detector defects etc. It is capable of handling images that have strip pattern (e.g. echelle spectra) using the REDUCE modinfo conventions to figure out image orientation and useful pixel ranges. It can handle many frames. Special cases: 1 file in the list (the input is returned as output) and 2 files (straight sum is returned).

If the image orientation is not predominantly vertical, the image is rotated 90 degrees (and rotated back afterwards).

Open all FITS files in the list. Loop through the rows. Read next row from each file into a row buffer mBuff[nCol, nFil]. Optionally correct the data for non-linearity.

calc_probability:

```
Go through the row creating "probability" vector. That is for column iCol take␣
↪the median of
the part of the row mBuff[iCol-win:iCol+win,iFil] for each file and divide these␣
↪medians by the
mean of them computed across the stack of files. In other words:
>>> filwt[iFil] = median(mBuff[iCol-win:iCol+win,iFil])
>>> norm_filwt = mean(filwt)
>>> prob[iCol,iFil] = (norm_filtwt>0)?filwt[iCol]/norm_filwt:filwt[iCol]

This is done for all iCol in the range of [win:nCol-win-1]. It is then linearly␣
↪extrapolated to
the win zones of both ends. E.g. for iCol in [0:win-1] range:
>>> prob[iCol,iFil]=2*prob[win,iFil]-prob[2*win-iCol,iFil]

For the other end ([nCol-win:nCol-1]) it is similar:
>>> prob[iCol,iFil]=2*prob[nCol-win-1,iFil]-prob[2*(nCol-win-1)-iCol,iFil]
```

fix_bad_pixels:

```
Once the probailities are constructed we can do the fitting, measure scatter and␣
↪detect outliers.
We ignore negative or zero probabilities as it should not happen. For each iCol␣
↪with (some)
positive probabilities we compute tha ratios of the original data to the␣
↪probabilities and get
the mean amplitude of these ratios after rejecting extreme values:
>>> ratio = mBuff[iCol,iFil]/prob[iCol,iFil]
>>> amp = (total(ratio)-min(ratio)-max(ratio))/(nFil-2)
>>> mFit[iCol,iFil] = amp*prob[iCol,iFil]

Note that for iFil whereprob[iCol,iFil] is zero we simply set mFit to zero. The␣
↪scatter (noise)
consists readout noise and shot noise of the model (fit) co-added in quadratures:
>>> sig=sqrt(rdnoise*rdnoise + abs(mFit[iCol,iFil]/gain))

and the outliers are defined as:
>>> iBad=where(mBuff-mFit gt thresh*sig)

>>> Bad values are replaced from the fit:
>>> mBuff[iBad]=mFit[iBad]

and mBuff is summed across the file dimension to create an output row.
```

Parameters

- **files** (*list(str)*) – list of fits files to combine
- **instrument** (*str*) – instrument id for modinfo
- **mode** (*str*) – instrument mode
- **extension** (*int, optional*) – fits extension to load (default: 1)
- **threshold** (*float, optional*) – threshold for bad pixels (default: 3.5)
- **window** (*int, optional*) – horizontal window size (default: 50)
- **mask** (*array(bool), optional*) – mask for the fits image (default: None)
- **xr** (*int, optional*) – xrange (default: None)
- **yr** (*int, optional*) – yrange (default: None)
- **debug** (*bool, optional*) – show debug plot of noise distribution (default: False)
- **dtype** (*np.dtype, optional*) – datatype of the combined image (default float32)

Returns combined image data, header

Return type combined_data, header

pyreduce.combine_frames.**combine_polynomial**(*files*, *instrument*, *mode*, *mask*, *degree=1*, *plot=False*, *plot_title=None*)

Combine the input files by fitting a polynomial of the pixel value versus the exposure time of each pixel

Parameters

- **files** (*list*) – list of file names
- **instrument** (Instrument) – PyReduce instrument object with load_fits method
- **mode** (*str*) – mode identifier for this instrument

- **mask** (*array*) – bad pixel mask to apply to the coefficients
- **degree** (*int, optional*) – polynomial degree of the fit, by default 1
- **plot** (*bool, optional*) – whether to plot the results, by default False
- **plot_title** (*str, optional*) – Title of the plot, by default None

Returns

- **bias** (*array*) – 3d array with the coefficients for each pixel
- **bhead** (*Header*) – combined FITS header of the coefficients

pyreduce.combine_frames.**fix_bad_pixels**(*probability*, *buffer*, *readnoise*, *gain*, *threshold*)
find and fix bad pixels

Parameters

- **probability** (*array (float)*) – probabilities
- **buffer** (*array (int)*) – image buffer
- **readnoise** (*float*) – readnoise of current amplifier
- **gain** (*float*) – gain of current amplifier
- **threshold** (*float*) – sigma threshold between observation and fit for bad pixels

Returns input buffer, with bad pixels fixed

Return type array(int)

pyreduce.combine_frames.**running_median**(*arr*, *size*)
Calculate the running median of a 2D sequence

Parameters

- **seq** (*2d array [n, l]*) – n datasets of length l
- **size** (*int*) – number of elements to consider for each median

Returns running median

Return type 2d array [n, l-size]

pyreduce.combine_frames.**running_sum**(*arr*, *size*)
Calculate the running sum over the 2D sequence

Parameters

- **arr** (*array[n, l]*) – sequence to calculate running sum over, n datasets of length l
- **size** (*int*) – number of elements to sum

Returns running sum

Return type 2D array

6.1.5 pyreduce.continuum_normalization module

Find the continuum level

Currently only splices orders together First guess of the continuum is provided by the flat field

class pyreduce.continuum_normalization.**Plot_Normalization**(*wsort*, *sB*, *new_wave*, *contB*, *iteration=0*, *title=None*)

Bases: object

close()

plot(*wsort*, *sB*, *new_wave*, *contB*, *iteration*)

pyreduce.continuum_normalization.**continuum_normalize**(*spec*, *wave*, *cont*, *sigm*, *iterations=10*, *smooth_initial=100000.0*, *smooth_final=5000000.0*, *scale_vert=1*, *plot=True*, *plot_title=None*)

Fit a continuum to a spectrum by slowly approaching it from the top. We exploit here that the continuum varies only on large wavelength scales, while individual lines act on much smaller scales

TODO automatically find good parameters for smooth_initial and smooth_final TODO give variables better names

Parameters

- **spec** (*masked array of shape (nord, ncol)*) – Observed input spectrum, masked values describe column ranges
- **wave** (*masked array of shape (nord, ncol)*) – Wavelength solution of the spectrum
- **cont** (*masked array of shape (nord, ncol)*) – Initial continuum guess, for example based on the blaze
- **sigm** (*masked array of shape (nord, ncol)*) – Uncertainties of the spectrum
- **iterations** (*int, optional*) – Number of iterations of the algorithm, note that runtime roughly scales with the number of iterations squared (default: 10)
- **smooth_initial** (*float, optional*) – Smoothing parameter in the initial runs, usually smaller than smooth_final (default: 1e5)
- **smooth_final** (*float, optional*) – Smoothing parameter of the final run (default: 5e6)
- **scale_vert** (*float, optional*) – Vertical scale of the spectrum. Usually 1 if a previous normalization exists (default: 1)
- **plot** (*bool, optional*) – Wether to plot the current status and results or not (default: True)

Returns **cont** – New continuum

Return type masked array of shape (nord, ncol)

pyreduce.continuum_normalization.**splice_orders**(*spec*, *wave*, *cont*, *sigm*, *scaling=True*, *plot=False*, *plot_title=None*)

Splice orders together so that they form a continous spectrum This is achieved by linearly combining the overlaping regions

Parameters

- **spec** (*array[nord, ncol]*) – Spectrum to splice, with seperate orders
- **wave** (*array[nord, ncol]*) – Wavelength solution for each point
- **cont** (*array[nord, ncol]*) – Continuum, blaze function will do fine as well

- **sigm** (*array[nord, ncol]*) – Errors on the spectrum
- **scaling** (*bool, optional*) – If true, the spectrum/continuum will be scaled to 1 (default: False)
- **plot** (*bool, optional*) – If true, will plot the spliced spectrum (default: False)

Raises NotImplementedError – If neighbouring orders dont overlap

Returns **spec, wave, cont, sigm** – spliced spectrum

Return type array[nord, ncol]

6.1.6 pyreduce.cwrappers module

Wrapper for REDUCE C functions

This module provides access to the extraction algorithms in the C libraries and sanitizes the input parameters.

pyreduce.cwrappers.**slitfunc**(*img*, *ycen*, *lambda_sp=0*, *lambda_sf=0.1*, *osample=1*)

Decompose image into spectrum and slitfunction

This is for horizontal straight orders only, for curved orders use slitfunc_curved instead

Parameters

- **img** (*array[n, m]*) – image to decompose, should just contain a small part of the overall image
- **ycen** (*array[n]*) – traces the center of the order along the image, relative to the center of the image?
- **lambda_sp** (*float, optional*) – smoothing parameter of the spectrum (the default is 0, which no smoothing)
- **lambda_sf** (*float, optional*) – smoothing parameter of the slitfunction (the default is 0.1, which)
- **osample** (*int, optional*) – Subpixel ovsersampling factor (the default is 1, which no oversampling)

Returns spectrum, slitfunction, model, spectrum uncertainties

Return type sp, sl, model, unc

pyreduce.cwrappers.**slitfunc_curved**(*img*, *ycen*, *tilt*, *shear*, *lambda_sp*, *lambda_sf*, *osample*, *yrange*, *maxiter=20*, *gain=1*)

Decompose an image into a spectrum and a slitfunction, image may be curved

Parameters

- **img** (*array[n, m]*) – input image
- **ycen** (*array[n]*) – traces the center of the order
- **tilt** (*array[n]*) – tilt (1st order curvature) of the order along the image, set to 0 if order straight
- **shear** (*array[n]*) – shear (2nd order curvature) of the order along the image, set to 0 if order straight
- **osample** (*int*) – Subpixel ovsersampling factor (the default is 1, no oversampling)
- **lambda_sp** (*float*) – smoothing factor spectrum (the default is 0, no smoothing)
- **lambda_sl** (*float*) – smoothing factor slitfunction (the default is 0.1, small smoothing)

- **yrange** (*array[2]*) – number of pixels below and above the central line that have been cut out
- **maxiter** (*int, optional*) – maximumim number of iterations, by default 20
- **gain** (*float, optional*) – gain of the image, by default 1

Returns spectrum, slitfunction, model, spectrum uncertainties

Return type sp, sl, model, unc

6.1.7 pyreduce.datasets module

Provides example datasets for the examples

This requires the server to be up and running, if data needs to be downloaded

pyreduce.datasets.**HARPS**(*local_dir=None*)

Load an example dataset instrument: HARPS target: HD109200

Parameters **local_dir** (*str, optional*) – directory to save data at (default: "./")

Returns **dataset_dir** – directory where the data was saved

Return type str

pyreduce.datasets.**JWST_MIRI**(*local_dir=None*)

Load an example dataset instrument: JWST_MIRI target: ?

Data simulated with MIRIsim

Parameters **local_dir** (*str, optional*) – directory to save data at (default: "./")

Returns **dataset_dir** – directory where the data was saved

Return type str

pyreduce.datasets.**JWST_NIRISS**(*local_dir=None*)

Load an example dataset instrument: JWST_NIRISS target: ?

Data simulated with awesimsoss

Parameters **local_dir** (*str, optional*) – directory to save data at (default: "./")

Returns **dataset_dir** – directory where the data was saved

Return type str

pyreduce.datasets.**KECK_NIRSPEC**(*local_dir=None*)

Load an example dataset instrument: KECK_NIRSPEC target: GJ1214

Parameters **local_dir** (*str, optional*) – directory to save data at (default: "./")

Returns **dataset_dir** – directory where the data was saved

Return type str

pyreduce.datasets.**LICK_APF**(*local_dir=None*)

Load an example dataset instrument: LICK_APF target: KIC05005618

Parameters **local_dir** (*str, optional*) – directory to save data at (default: "./")

Returns **dataset_dir** – directory where the data was saved

Return type str

pyreduce.datasets.**MCDONALD**(*local_dir=None*)
Load an example dataset instrument: JWST_MIRI target: ?

Data simulated with MIRIsim

Parameters **local_dir** (*str, optional*) – directory to save data at (default: "./")

Returns **dataset_dir** – directory where the data was saved

Return type str

pyreduce.datasets.**UVES**(*local_dir=None*)
Load an example dataset instrument: UVES target: HD132205

Parameters **local_dir** (*str, optional*) – directory to save data at (default: "./")

Returns **dataset_dir** – directory where the data was saved

Return type str

pyreduce.datasets.**XSHOOTER**(*local_dir=None*)
Load an example dataset instrument: XSHOOTER target: Ux-Ori

Parameters **local_dir** (*str, optional*) – directory to save data at (default: "./")

Returns **dataset_dir** – directory where the data was saved

Return type str

pyreduce.datasets.**get_dataset**(*name*, *local_dir=None*)
Load a dataset

Note: This method will not override existing files with the same name, even if they have a different content. Therefore if the files were changed for any reason, the user has to manually delete them from the disk before using this method.

Parameters

- **name** (*str*) – Name of the dataset
- **local_dir** (*str, optional*) – directory to save data at (default: "./")

Returns **dataset_dir** – directory where the data was saved

Return type str

pyreduce.datasets.**load_data_from_server**(*filename*, *directory*)

6.1.8 pyreduce.echelle module

Contains functions to read and modify echelle structures, just as in reduce

Mostly for compatibility reasons

class pyreduce.echelle.**Echelle**(*head={}*, *filename=''*, *data={}*)
Bases: object

columns

cont

mask

ncol

nord

static read(*fname*, *extension=1*, *raw=False*, *continuum_normalization=True*, *barycentric_correction=True*, *radial_velociy_correction=True*)

Read data from an echelle file Expand wavelength and continuum polynomials Apply barycentric/radial velocity correction Apply continuum normalization

Will load any fields in the binary table, however special attention is given only to specific names: "SPEC" : Spectrum "SIG" : Sigma, i.e. (absolute) uncertainty "CONT" : Continuum "WAVE" : Wavelength solution "COLUMNS" : Column range

Parameters

- **fname** (*str*) – filename to load
- **extension** (*int, optional*) – fits extension of the data within the file (default: 1)
- **raw** (*bool, optional*) – if true apply no corrections to the data (default: False)
- **continuum_normalization** (*bool, optional*) – apply continuum normalization (default: True)
- **barycentric_correction** (*bool, optional*) – apply barycentric correction (default: True)
- **radial_velociy_correction** (*bool, optional*) – apply radial velocity correction (default: True)

Returns **ech** – Echelle structure, with data contained in attributes

Return type obj

save(*fname*)

Save data in an Echelle fits, i.e. a fits file with a Binary Table in Extension 1

Parameters **fname** (*str*) – filename

sig

spec

wave

pyreduce.echelle.**calc_1dpolynomials**(*ncol*, *poly*)

Expand a set of 1d polynomials (one per order) seperately

Parameters

- **ncol** (*int*) – number of columns
- **poly** (*array[nord, degree]*) – polynomial coefficients

Returns **poly** – expanded polynomials

Return type array[nord, ncol]

pyreduce.echelle.**calc_2dpolynomial**(*solution2d*)

Expand a 2d polynomial, where the data is given in a REDUCE make_wave format Note that the coefficients are for order/100 and column/1000 respectively, where the order is counted from order base up

Parameters **solution2d** (*array*) – data in a REDUCE make_wave format: 0: version 1: number of columns 2: number of orders 3: order base, i.e. 0th order number (should not be 0) 4-6: empty 7: number of cross coefficients 8: number of column only coefficients 9: number of order only coefficients 10: coefficient - constant 11-x: column coefficients x-y : order coefficients z- : cross coefficients (xy, xy2, x2y, x2y2, xy3, x3y), with x = orders, y = columns

Returns **poly** – expanded polynomial

Return type array[nord, ncol]

pyreduce.echelle.**expand_polynomial**(*ncol*, *poly*)
Checks if and how to expand data poly, then expands the data if necessary

Parameters

- **ncol** (*int*) – number of columns in the image
- **poly** (*array[nord, ..]*) – polynomial coefficients to expand, or already expanded data

Returns **poly** – expanded data

Return type array[nord, ncol]

pyreduce.echelle.**read**(*fname*, ***kwargs*)

pyreduce.echelle.**save**(*fname*, *header*, ***kwargs*)
Save data in an Echelle fits, i.e. a fits file with a Binary Table in Extension 1

The data is passed in kwargs, with the name of the binary table column as the key Floating point data is saved as float32 (E), Integer data as int16 (I)

Parameters

- **fname** (*str*) – filename
- **header** (*fits.header*) – FITS header
- ****kwargs** (*array[]*) – data to be saved in the file

6.1.9 pyreduce.estimate_background_scatter module

Module that estimates the background scatter

pyreduce.estimate_background_scatter.**estimate_background_scatter**(*img*, *orders*, *column_range=None*, *extraction_width=0.1*, *scatter_degree=4*, *sigma_cutoff=2*, *border_width=10*, *plot=False*, *plot_title=None*)
Estimate the background by fitting a 2d polynomial to interorder data

Interorder data is all pixels minus the orders +- the extraction width

Parameters

- **img** (*array[nrow, ncol]*) – (flat) image data
- **orders** (*array[nord, degree]*) – order polynomial coefficients
- **column_range** (*array[nord, 2], optional*) – range of columns to use in each order (default: None == all columns)

- **extraction_width** (*float, array[nord, 2], optional*) – extraction width for each order, values below 1.5 are considered fractional, others as number of pixels (default: 0.1)
- **scatter_degree** (*int, optional*) – polynomial degree of the 2d fit for the background scatter (default: 4)
- **plot** (*bool, optional*) – wether to plot the fitted polynomial and the data or not (default: False)

Returns

- *array[nord+1, ncol]* – background scatter between orders
- *array[nord+1, ncol]* – y positions of the interorder lines, the scatter values are taken from

6.1.10 pyreduce.extract module

Module for extracting data from observations

Authors

Version

License

class pyreduce.extract.**ProgressPlot**(*nrow*, *ncol*, *nslitf*, *nbad=1000*, *title=None*)

Bases: object

close()

fix_linear(*data*, *limit*, *fill=0*)
Assures the size of the 1D array data is equal to limit

get_slitf(*img*, *spec*, *slitf*, *ycen*)
get the slit function

get_spec(*img*, *spec*, *slitf*, *ycen*)
get the spectrum corrected by the slit function

plot(*img*, *spec*, *slitf*, *model*, *ycen*, *mask*, *ord_num*, *left*, *right*)

class pyreduce.extract.**Swath**(*nswath*)

Bases: object

pyreduce.extract.**arc_extraction**(*img*, *orders*, *extraction_width*, *column_range*, *gain=1*, *readnoise=0*, *dark=0*, *plot=False*, *plot_title=None*, *tilt=None*, *shear=None*, *collapse_function='median'*, ***kwargs*)

Use "simple" arc extraction to get a spectrum Arc extraction simply takes the sum orthogonal to the order for extraction width pixels

This extraction makes a few rough assumptions and does not provide the most accurate results, but rather a good approximation

Parameters

- **img** (*array[nrow, ncol]*) – image to extract
- **orders** (*array[nord, order]*) – order tracing coefficients
- **extraction_width** (*array[nord, 2]*) – extraction width in pixels

- **column_range** (*array[nord, 2]*) – column range to use
- **gain** (*float, optional*) – adu to electron, amplifier gain (default: 1)
- **readnoise** (*float, optional*) – read out noise (default: 0)
- **dark** (*float, optional*) – dark current noise (default: 0)
- **plot** (*bool, optional*) – wether to plot the results (default: False)

Returns

- **spectrum** (*array[nord, ncol]*) – extracted spectrum
- **uncertainties** (*array[nord, ncol]*) – uncertainties on extracted spectrum

pyreduce.extract.**calc_scatter_correction**(*scatter*, *index*)
Calculate scatter correction by interpolating between values?

Parameters

- **scatter** (*array of shape (degree_x, degree_y)*) – 2D polynomial coefficients of the background scatter
- **index** (*tuple (array, array)*) – indices of the swath within the overall image

Returns **scatter_correction** – correction for scattered light

Return type array of shape (swath_width, swath_height)

pyreduce.extract.**calc_telluric_correction**(*telluric*, *img*)
Calculate telluric correction

If set to specific integer larger than 1 is used as the offset from the order center line. The sky is then estimated by computing median signal between this offset and the upper/lower limit of the extraction window.

Parameters

- **telluric** (*int*) – telluric correction parameter
- **img** (*array*) – image of the swath

Returns **tell** – telluric correction

Return type array

pyreduce.extract.**correct_for_curvature**(*img_order*, *tilt*, *shear*, *xwd*)

pyreduce.extract.**extend_orders**(*orders*, *nrow*)
Extrapolate extra orders above and below the existing ones

Parameters

- **orders** (*array[nord, degree]*) – order tracing coefficients
- **nrow** (*int*) – number of rows in the image

Returns **orders** – extended orders

Return type array[nord + 2, degree]

pyreduce.extract.**extract**(*img*, *orders*, *column_range=None*, *order_range=None*, *extraction_width=0.5*, *extraction_type='optimal'*, *tilt=None*, *shear=None*, *sigma_cutoff=0*, ***kwargs*)
Extract the spectrum from an image

Parameters

- **img** (*array[nrow, ncol](float)*) – observation to extract

- **orders** (*array[nord, degree](float)*) – polynomial coefficients of the order tracing
- **column_range** (*array[nord, 2](int), optional*) – range of pixels to use for each order (default: use all)
- **order_range** (*array[2](int), optional*) – range of orders to extract, orders have to be consecutive (default: use all)
- **extraction_width** (*array[nord, 2]({float, int}), optional*) – extraction width above and below each order, values below 1.5 are considered relative, while values above are absolute (default: 0.5)
- **extraction_type** (*{"optimal", "arc", "normalize"}, optional*) – which extracttion algorithm to use, "optimal" uses optimal extraction, "arc" uses simple arc extraction, and "normalize" also uses optimal extraction, but returns the normalized image (default: "optimal")
- **tilt** (*float or array[nord, ncol], optional*) – The tilt (1st order curvature) of the slit for curved extraction. Will use vertical extraction if no tilt is set. (default: None, i.e. tilt = 0)
- **shear** (*float or array[nord, ncol], optional*) – The shear (2nd order curvature) of the slit for curved extraction (default: None, i.e. shear = 0)
- **polarization** (*bool, optional*) – if true, pairs of orders are considered to belong to the same order, but different polarization. Only affects the scatter (default: False)
- **optional** (***kwargs,*) – parameters for extraction functions

Returns

- **spec** (*array[nord, ncol](float)*) – extracted spectrum for each order
- **uncertainties** (*array[nord, ncol](float)*) – uncertainties on the spectrum
- *if extraction_type == "normalize" instead return*
- **im_norm** (*array[nrow, ncol](float)*) – normalized image
- **im_ordr** (*array[nrow, ncol](float)*) – image with just the orders
- **blaze** (*array[nord, ncol](float)*) – extracted spectrum (equals blaze if img was the flat field)

pyreduce.extract.**extract_spectrum**(*img*, *ycen*, *yrange*, *xrange*, *gain=1*, *readnoise=0*, *lambda_sf=0.1*, *lambda_sp=0*, *osample=1*, *swath_width=None*, *maxiter=20*, *telluric=None*, *scatter=None*, *normalize=False*, *threshold=0*, *tilt=None*, *shear=None*, *plot=False*, *plot_title=None*, *im_norm=None*, *im_ordr=None*, *out_spec=None*, *out_sunc=None*, *out_slitf=None*, *out_mask=None*, *progress=None*, *ord_num=0*, ***kwargs*)

Extract the spectrum of a single order from an image The order is split into several swathes of roughly swath_width length, which overlap half-half For each swath a spectrum and slitfunction are extracted overlapping sections are combined using linear weights (centrum is strongest, falling off to the edges) Here is the layout for the bins:

```
   1st swath    3rd swath    5th swath      ...
/============|============|============|============|============|

          2nd swath    4th swath    6th swath
       |------------|------------|------------|------------|
```

(continues on next page)

(continued from previous page)

```
      |.....|
      overlap

      +     ******* 1
       +   *
        + *
         *             weights (+) previous swath, (*) current swath
        * +
       *   +
      *     +++++++ 0
```

Parameters

- **img** (*array[nrow, ncol]*) – observation (or similar)
- **ycen** (*array[ncol]*) – order trace of the current order
- **yrange** (*tuple(int, int)*) – extraction width in pixles, below and above
- **xrange** (*tuple(int, int)*) – columns range to extract (low, high)
- **gain** (*float, optional*) – adu to electron, amplifier gain (default: 1)
- **readnoise** (*float, optional*) – read out noise factor (default: 0)
- **lambda_sf** (*float, optional*) – slit function smoothing parameter, usually very small (default: 0.1)
- **lambda_sp** (*int, optional*) – spectrum smoothing parameter, usually very small (default: 0)
- **osample** (*int, optional*) – oversampling factor, i.e. how many subpixels to create per pixel (default: 1, i.e. no oversampling)
- **swath_width** (*int, optional*) – swath width suggestion, actual width depends also on ncol, see make_bins (default: None, which will determine the width based on the order tracing)
- **telluric** (*{float, None}, optional*) – telluric correction factor (default: None, i.e. no telluric correction)
- **scatter** (*{array, None}, optional*) – background scatter as 2d polynomial coefficients (default: None, no correction)
- **normalize** (*bool, optional*) – whether to create a normalized image. If true, im_norm and im_ordr are used as output (default: False)
- **threshold** (*int, optional*) – threshold for normalization (default: 0)
- **tilt** (*array[ncol], optional*) – The tilt (1st order curvature) of the slit in this order for the curved extraction (default: None, i.e. tilt = 0)
- **shear** (*array[ncol], optional*) – The shear (2nd order curvature) of the slit in this order for the curved extraction (default: None, i.e. shear = 0)
- **plot** (*bool, optional*) – wether to plot the progress, plotting will slow down the procedure significantly (default: False)
- **ord_num** (*int, optional*) – current order number, just for plotting (default: 0)
- **im_norm** (*array[nrow, ncol], optional*) – normalized image, only output if normalize is True (default: None)

- **im_ordr** (*array[nrow, ncol], optional*) – image of the order blaze, only output if normalize is True (default: None)

Returns

- **spec** (*array[ncol]*) – extracted spectrum
- **slitf** (*array[nslitf]*) – extracted slitfunction
- **mask** (*array[ncol]*) – mask of the column range to use in the spectrum
- **unc** (*array[ncol]*) – uncertainty on the spectrum

pyreduce.extract.**fix_column_range**(*column_range*, *orders*, *extraction_width*, *nrow*, *ncol*)

Fix the column range, so that no pixels outside the image will be accessed (Thus avoiding errors)

Parameters

- **img** (*array[nrow, ncol]*) – image
- **orders** (*array[nord, degree]*) – order tracing coefficients
- **extraction_width** (*array[nord, 2]*) – extraction width in pixels, (below, above)
- **column_range** (*array[nord, 2]*) – current column range
- **no_clip** (*bool, optional*) – if False, new column range will be smaller or equal to current column range, otherwise it can also be larger (default: False)

Returns **column_range** – updated column range

Return type array[nord, 2]

pyreduce.extract.**fix_extraction_width**(*xwd*, *orders*, *cr*, *ncol*)

Convert fractional extraction width to pixel range

Parameters

- **extraction_width** (*array[nord, 2]*) – current extraction width, in pixels or fractions (for values below 1.5)
- **orders** (*array[nord, degree]*) – order tracing coefficients
- **column_range** (*array[nord, 2]*) – column range to use
- **ncol** (*int*) – number of columns in image

Returns **extraction_width** – updated extraction width in pixels

Return type array[nord, 2]

pyreduce.extract.**fix_parameters**(*xwd*, *cr*, *orders*, *nrow*, *ncol*, *nord*, *ignore_column_range=False*)

Fix extraction width and column range, so that all pixels used are within the image. I.e. the column range is cut so that the everything is within the image

Parameters

- **xwd** (*float, array*) – Extraction width, either one value for all orders, or the whole array
- **cr** (*2-tuple(int), array*) – Column range, either one value for all orders, or the whole array
- **orders** (*array*) – polynomial coefficients that describe each order
- **nrow** (*int*) – Number of rows in the image

- **ncol** (*int*) – Number of columns in the image
- **nord** (*int*) – Number of orders in the image
- **ignore_column_range** (*bool, optional*) – if true does not change the column range, however this may lead to problems with the extraction, by default False

Returns

- **xwd** (*array*) – fixed extraction width
- **cr** (*array*) – fixed column range
- **orders** (*array*) – the same orders as before

pyreduce.extract.**get_mask**(*img*, *model*)

pyreduce.extract.**get_y_scale**(*ycen*, *xrange*, *extraction_width*, *nrow*)

Calculate the y limits of the order This is especially important at the edges

Parameters

- **ycen** (*array[ncol]*) – order trace
- **xrange** (*tuple(int, int)*) – column range
- **extraction_width** (*tuple(int, int)*) – extraction width in pixels below and above the order
- **nrow** (*int*) – number of rows in the image, defines upper edge

Returns **y_low, y_high** – lower and upper y bound for extraction

Return type int, int

pyreduce.extract.**make_bins**(*swath_width*, *xlow*, *xhigh*, *ycen*)

Create bins for the swathes Bins are roughly equally sized, have roughly length swath width (if given) and overlap roughly half-half with each other

Parameters

- **swath_width** (*{int, None}*) – initial value for the swath_width, bins will have roughly that size, but exact value may change if swath_width is None, determine a good value, from the data
- **xlow** (*int*) – lower bound for x values
- **xhigh** (*int*) – upper bound for x values
- **ycen** (*array[ncol]*) – center of the order trace

Returns

- **nbin** (*int*) – number of bins
- **bins_start** (*array[nbin]*) – left(beginning) side of the bins
- **bins_end** (*array[nbin]*) – right(ending) side of the bins

pyreduce.extract.**model**(*spec*, *slitf*)

pyreduce.extract.**model_image**(*img*, *xwd*, *tilt*, *shear*)

pyreduce.extract.**optimal_extraction**(*img*, *orders*, *extraction_width*, *column_range*, *tilt*, *shear*, *plot=False*, *plot_title=None*, ***kwargs*)

Use optimal extraction to get spectra

This functions just loops over the orders, the actual work is done in extract_spectrum

Parameters

- **img** (*array[nrow, ncol]*) – image to extract
- **orders** (*array[nord, degree]*) – order tracing coefficients
- **extraction_width** (*array[nord, 2]*) – extraction width in pixels
- **column_range** (*array[nord, 2]*) – column range to use
- **scatter** (*array[nord, 4, ncol]*) – background scatter (or None)
- ****kwargs** – other parameters for the extraction (see extract_spectrum)

Returns

- **spectrum** (*array[nord, ncol]*) – extracted spectrum
- **slitfunction** (*array[nord, nslitf]*) – recovered slitfunction
- **uncertainties** (*array[nord, ncol]*) – uncertainties on the spectrum

pyreduce.extract.**plot_comparison**(*original*, *orders*, *spectrum*, *slitf*, *extraction_width*, *column_range*, *title=None*)

6.1.11 pyreduce.make_shear module

Calculate the tilt based on a reference spectrum with high SNR, e.g. Wavelength calibration image

Authors

Nikolai Piskunov Ansgar Wehrhahn

Version

0.9 - NP - IDL Version 1.0 - AW - Python Version

License

---

class pyreduce.make_shear.**Curvature**(*orders*, *extraction_width=0.5*, *column_range=None*, *order_range=None*, *window_width=9*, *peak_threshold=10*, *peak_width=1*, *fit_degree=2*, *sigma_cutoff=3*, *mode='1D'*, *plot=False*, *plot_title=None*, *peak_function='gaussian'*, *curv_degree=2*)

Bases: object

eval(*peaks*, *order*, *coef_tilt*, *coef_shear*)

execute(*extracted*, *original*)

fit(*peaks*, *tilt*, *shear*)

mode

n

nord

plot_comparison(*original*, *tilt*, *shear*, *peaks*)

plot_results(*ncol*, *plot_peaks*, *plot_vec*, *plot_tilt*, *plot_shear*, *tilt_x*, *shear_x*)

class pyreduce.make_shear.**ProgressPlot**(*ncol*, *width*, *title=None*)
Bases: object

close()

update_plot1(*vector*, *peak*, *offset=0*)

update_plot2(*img*, *model*, *tilt*, *shear*, *peak*)

pyreduce.make_shear.**gaussian**(*x*, *A*, *mu*, *sig*)
A: height mu: offset from central line sig: standard deviation

pyreduce.make_shear.**lorentzian**(*x*, *A*, *x0*, *mu*)
A: height x0: offset from central line mu: width of lorentzian

6.1.12 pyreduce.reduce module

REDUCE script for spectrograph data

Authors

Ansgar Wehrhahn (ansgar.wehrhahn@physics.uu.se) Thomas Marquart (thomas.marquart@physics.uu.se) Alexis Lavail (alexis.lavail@physics.uu.se) Nikolai Piskunov (nikolai.piskunov@physics.uu.se)

Version

1.0 - Initial PyReduce

License

...

class pyreduce.reduce.**BackgroundScatter**(**args*, ***config*)
Bases: *pyreduce.reduce.CalibrationStep*

Determine the background scatter

load()
Load scatter results from disk

Returns **scatter** – scatter coefficients

Return type array

run(*files*, *mask*, *bias*, *orders*)
Execute the current step

This should fail if files are missing or anything else goes wrong. If the user does not want to run this step, they should not specify it in steps.

Parameters **files** (*list(str)*) – data files required for this step

Raises NotImplementedError – needs to be implemented for each step

save(*scatter*)
Save scatter results to disk

Parameters **scatter** (*array*) – scatter coefficients

savefile
Name of the scatter file

Type str

scatter_degree = None
Polynomial degrees for the background scatter fit, in row, column direction

Type tuple(int, int)

class pyreduce.reduce.**Bias**(**args*, ***config*)
Bases: *pyreduce.reduce.Step*

Calculates the master bias

degree = None
polynomial degree of the fit between exposure time and pixel values

Type int

load(*mask*)
Load the master bias from a previous run

Parameters **mask** (*array of shape (nrow, ncol)*) – Bad pixel mask

Returns

- **bias** (*masked array of shape (nrow, ncol)*) – master bias data, with the bad pixel mask applied
- **bhead** (*FITS header*) – header of the master bias

run(*files*, *mask*)
Calculate the master bias

Parameters

- **files** (*list(str)*) – bias files
- **mask** (*array of shape (nrow, ncol)*) – bad pixel map

Returns

- **bias** (*masked array of shape (nrow, ncol)*) – master bias data, with the bad pixel mask applied
- **bhead** (*FITS header*) – header of the master bias

save(*bias*, *bhead*)
Save the master bias to a FITS file

Parameters

- **bias** (*array of shape (nrow, ncol)*) – bias data
- **bhead** (*FITS header*) – bias header

savefile
Name of master bias fits file

Type str

class pyreduce.reduce.**CalibrationStep**(**args*, ***config*)
Bases: *pyreduce.reduce.Step*

bias_scaling = None
how to adjust for diferences between the bias and flat field exposure times

Type {'number_of_files', 'exposure_time', 'mean', 'median', 'none'}

calibrate(*files*, *mask*, *bias=None*, *norm_flat=None*)

norm_scaling = None
how to apply the normalized flat field

Type {'divide', 'none'}

class pyreduce.reduce.**ContinuumNormalization**(**args*, ***config*)
Bases: *pyreduce.reduce.Step*

Determine the continuum to each observation

load(*norm_flat*, *science*)
Load the results from the continuum normalization

Returns

- **heads** (*list(FITS header)*) – FITS headers of each observation
- **specs** (*list(array of shape (nord, ncol))*) – extracted spectra
- **sigmas** (*list(array of shape (nord, ncol))*) – uncertainties of the extracted spectra
- **conts** (*list(array of shape (nord, ncol))*) – continuum for each spectrum
- **columns** (*list(array of shape (nord, 2))*) – column ranges for each spectra

run(*science*, *freq_comb*, *norm_flat*)
Determine the continuum to each observation Also splices the orders together

Parameters

- **science** (*tuple*) – results from science step
- **freq_comb_final** (*tuple*) – results from freq_comb_final step (or wavecal if those don't exist)
- **norm_flat** (*tuple*) – results from the normalized flatfield step

Returns

- **heads** (*list(FITS header)*) – FITS headers of each observation
- **specs** (*list(array of shape (nord, ncol))*) – extracted spectra
- **sigmas** (*list(array of shape (nord, ncol))*) – uncertainties of the extracted spectra
- **conts** (*list(array of shape (nord, ncol))*) – continuum for each spectrum
- **columns** (*list(array of shape (nord, 2))*) – column ranges for each spectra

save(*heads*, *specs*, *sigmas*, *conts*, *columns*)
Save the results from the continuum normalization

Parameters

- **heads** (*list(FITS header)*) – FITS headers of each observation
- **specs** (*list(array of shape (nord, ncol))*) – extracted spectra
- **sigmas** (*list(array of shape (nord, ncol))*) – uncertainties of the extracted spectra

- **conts** (*list(array of shape (nord, ncol))*) – continuum for each spectrum
- **columns** (*list(array of shape (nord, 2))*) – column ranges for each spectra

savefile
savefile name

Type str

class pyreduce.reduce.**ExtractionStep**(**args*, ***config*)
Bases: *pyreduce.reduce.Step*

extract(*img*, *head*, *orders*, *curvature*)

extraction_kwargs = None
arguments for the extraction

Type dict

extraction_method = None
Extraction method to use

Type {'arc', 'optimal'}

class pyreduce.reduce.**Finalize**(**args*, ***config*)
Bases: *pyreduce.reduce.Step*

Create the final output files

output_file(*number*, *name*)
str: output file name

run(*continuum*, *freq_comb_final*, *config*)
Create the final output files

this is includes:

- heliocentric corrections
- creating one echelle file

Parameters

- **continuum** (*tuple*) – results from the continuum normalization
- **freq_comb_final** (*tuple*) – results from the frequency comb step (or wavelength calibration)

save(*i*, *head*, *spec*, *sigma*, *cont*, *wave*, *columns*)
Save one output spectrum to disk

Parameters

- **i** (*int*) – individual number of each file
- **head** (*FITS header*) – FITS header
- **spec** (*array of shape (nord, ncol)*) – final spectrum
- **sigma** (*array of shape (nord, ncol)*) – final uncertainties
- **cont** (*array of shape (nord, ncol)*) – final continuum scales
- **wave** (*array of shape (nord, ncol)*) – wavelength solution

- **columns** (*array of shape (nord, 2)*) – columns that carry signal

Returns **out_file** – name of the output file

Return type str

save_config_to_header(*head*, *config*, *prefix='PR'*)

class pyreduce.reduce.**FitsIOStep**(**args*, ***kwargs*)

Bases: *pyreduce.reduce.Step*

load(*mask*)

Load the master bias from a previous run

Parameters **mask** (*array of shape (nrow, ncol)*) – Bad pixel mask

Returns

- **data** (*masked array of shape (nrow, ncol)*) – master bias data, with the bad pixel mask applied
- **head** (*FITS header*) – header of the master bias

save(*data*, *head*, *dtype=None*)

Save the data to a FITS file

Parameters

- **data** (*array of shape (nrow, ncol)*) – bias data
- **head** (*FITS header*) – bias header

class pyreduce.reduce.**Flat**(**args*, ***config*)

Bases: *pyreduce.reduce.CalibrationStep*

Calculates the master flat

load(*mask*)

Load master flat from disk

Parameters **mask** (*array of shape (nrow, ncol)*) – Bad pixel mask

Returns

- **flat** (*masked array of shape (nrow, ncol)*) – Master flat with bad pixel map applied
- **fhead** (*FITS header*) – Master flat FITS header

run(*files*, *bias*, *mask*)

Calculate the master flat, with the bias already subtracted

Parameters

- **files** (*list(str)*) – flat files
- **bias** (*tuple(array of shape (nrow, ncol), FITS header)*) – master bias and header
- **mask** (*array of shape (nrow, ncol)*) – Bad pixel mask

Returns

- **flat** (*masked array of shape (nrow, ncol)*) – Master flat with bad pixel map applied
- **fhead** (*FITS header*) – Master flat FITS header

save(*flat*, *fhead*)

Save the master flat to a FITS file

Parameters

- **flat** (*array of shape (nrow, ncol)*) – master flat data
- **fhead** (*FITS header*) – master flat header

savefile
Name of master bias fits file

Type str

class pyreduce.reduce.**LaserFrequencyCombFinalize**(**args*, ***config*)
Bases: *pyreduce.reduce.Step*

Improve the precision of the wavelength calibration with a laser frequency comb

degree = None
polynomial degree of the wavelength fit

Type tuple(int, int)

dimensionality = None
Whether to use 1D or 2D polynomials

Type {'1D', '2D'}

lfc_peak_width = None
Width of the peaks for finding them in the spectrum

Type int

load(*wavecal*)
Load the results of the frequency comb improvement if possible, otherwise just use the normal wavelength solution

Parameters **wavecal** (*tuple*) – results from the wavelength calibration step

Returns

- **wave** (*array of shape (nord, ncol)*) – improved wavelength solution
- **comb** (*array of shape (nord, ncol)*) – extracted frequency comb image

run(*freq_comb_master*, *wavecal*)
Improve the wavelength calibration with a laser frequency comb (or similar)

Parameters

- **files** (*list(str)*) – observation files
- **wavecal** (*tuple()*) – results from the wavelength calibration step
- **orders** (*tuple*) – results from the order tracing step
- **mask** (*array of shape (nrow, ncol)*) – Bad pixel mask

Returns

- **wave** (*array of shape (nord, ncol)*) – improved wavelength solution
- **comb** (*array of shape (nord, ncol)*) – extracted frequency comb image

save(*wave*)
Save the results of the frequency comb improvement

Parameters **wave** (*array of shape (nord, ncol)*) – improved wavelength solution

savefile
Name of the wavelength echelle file

Type str

threshold = None
residual threshold in m/s above which to remove lines

Type float

class pyreduce.reduce.**LaserFrequencyCombMaster**(**args*, ***config*)
Bases: *pyreduce.reduce.CalibrationStep*, *pyreduce.reduce.ExtractionStep*

Create a laser frequency comb (or similar) master image

load()
Load master comb from disk

Returns

- **comb** (*masked array of shape (nrow, ncol)*) – Master comb with bad pixel map applied
- **chead** (*FITS header*) – Master comb FITS header

run(*files*, *orders*, *mask*, *curvature*, *bias*, *norm_flat*)
Improve the wavelength calibration with a laser frequency comb (or similar)

Parameters

- **files** (*list (str)*) – observation files
- **orders** (*tuple*) – results from the order tracing step
- **mask** (*array of shape (nrow, ncol)*) – Bad pixel mask
- **curvature** (*tuple*) – results from the curvature step
- **bias** (*tuple*) – results from the bias step

Returns

- **comb** (*array of shape (nord, ncol)*) – extracted frequency comb image
- **chead** (*Header*) – FITS header of the combined image

save(*comb*, *chead*)
Save the master comb to a FITS file

Parameters

- **comb** (*array of shape (nrow, ncol)*) – master comb data
- **chead** (*FITS header*) – master comb header

savefile
Name of the wavelength echelle file

Type str

class pyreduce.reduce.**Mask**(**args*, ***config*)
Bases: *pyreduce.reduce.Step*

Load the bad pixel mask for the given instrument/mode

load()
Load the mask file from disk

Returns **mask** – Bad pixel mask for this setting

Return type array of shape (nrow, ncol)

run()
Load the mask file from disk

Returns **mask** – Bad pixel mask for this setting

Return type array of shape (nrow, ncol)

class pyreduce.reduce.**NormalizeFlatField**(**args*, ***config*)
Bases: *pyreduce.reduce.Step*

Calculate the 'normalized' flat field image

extraction_kwargs = None
arguments for the extraction

Type dict

extraction_method = None
Extraction method to use

Type {'normalize'}

load()
Load normalized flat field results from disk

Returns

- **norm** (*array of shape (nrow, ncol)*) – normalized flat field
- **blaze** (*array of shape (nord, ncol)*) – Continuum level as determined from the flat field for each order

run(*flat*, *orders*, *scatter*, *curvature*)
Calculate the 'normalized' flat field

Parameters

- **flat** (*tuple(array, header)*) – Master flat, and its FITS header
- **orders** (*tuple(array, array)*) – Polynomial coefficients for each order, and the first and last(+1) column containing signal

Returns

- **norm** (*array of shape (nrow, ncol)*) – normalized flat field
- **blaze** (*array of shape (nord, ncol)*) – Continuum level as determined from the flat field for each order

save(*norm*, *blaze*)
Save normalized flat field results to disk

Parameters

- **norm** (*array of shape (nrow, ncol)*) – normalized flat field
- **blaze** (*array of shape (nord, ncol)*) – Continuum level as determined from the flat field for each order

savefile
Name of the blaze file

Type str

threshold = None
Threshold of the normalized flat field (values below this are just 1)

Type int

class pyreduce.reduce.**OrderTracing**(**args*, ***config*)
Bases: *pyreduce.reduce.CalibrationStep*

Determine the polynomial fits describing the pixel locations of each order

border_width = None
Number of pixels at the edge of the detector to ignore

Type int

filter_size = None
Size of the gaussian filter for smoothing

Type int

fit_degree = None
Polynomial degree of the fit to each order

Type int

load()
Load order tracing results

Returns

- **orders** (*array of shape (nord, ndegree+1)*) – polynomial coefficients for each order
- **column_range** (*array of shape (nord, 2)*) – first and last(+1) column that carries signal in each order

manual = None
Whether to use manual alignment

Type bool

min_cluster = None
Minimum size of each cluster to be included in further processing

Type int

min_width = None
Minimum width of each cluster after mergin

Type int, float

noise = None
Background noise value threshold

Type int

run(*files*, *mask*, *bias*)
Determine polynomial coefficients describing order locations

Parameters

- **files** (*list(str)*) – Observation used for order tracing (should only have one element)
- **mask** (*array of shape (nrow, ncol)*) – Bad pixel mask

Returns

- **orders** (*array of shape (nord, ndegree+1)*) – polynomial coefficients for each order
- **column_range** (*array of shape (nord, 2)*) – first and last(+1) column that carries signal in each order

save(*orders*, *column_range*)
Save order tracing results to disk

Parameters

- **orders** (*array of shape (nord, ndegree+1)*) – polynomial coefficients
- **column_range** (*array of shape (nord, 2)*) – first and last(+1) column that carry signal in each order

savefile
Name of the order tracing file

Type str

class pyreduce.reduce.**RectifyImage**(**args*, ***config*)
Bases: *pyreduce.reduce.Step*

Create a 2D image of the rectified orders

filename(*name*)

load(*files*)
Load results from a previous execution

If this raises a FileNotFoundError, run() will be used instead For calibration steps it is preferred however to print a warning and return None. Other modules can then use a default value instead.

Raises NotImplementedError – Needs to be implemented for each step

run(*files*, *orders*, *curvature*, *mask*, *freq_comb_final*)
Execute the current step

This should fail if files are missing or anything else goes wrong. If the user does not want to run this step, they should not specify it in steps.

Parameters **files** (*list(str)*) – data files required for this step

Raises NotImplementedError – needs to be implemented for each step

save(*fname*, *image*, *wavelength*, *header=None*)
Save the results of this step

Parameters ***args** (*obj*) – things to save

Raises NotImplementedError – Needs to be implemented for each step

class pyreduce.reduce.**Reducer**(*files*, *output_dir*, *target*, *instrument*, *mode*, *night*, *config*, *order_range=None*, *skip_existing=False*)
Bases: object

files = None
Filenames sorted by usecase

Type dict(str

Type str)

modules = {'bias': <class 'pyreduce.reduce.Bias'>, 'continuum': <class 'pyreduce.red

prepare_output_dir()

run_module(*step*, *load=False*)

run_steps(*steps='all'*)
Execute the steps as required

Parameters **steps** (*{tuple(str), "all"}, optional*) – which steps of the reduction process to perform the possible steps are: "bias", "flat", "orders", "norm_flat", "wavecal", "freq_comb", "curvature", "science", "continuum", "finalize" alternatively set steps to "all", which is equivalent to setting all steps

step_order = {'bias': 10, 'continuum': 90, 'curvature': 40, 'finalize': 100, 'flat

class pyreduce.reduce.**ScienceExtraction**(**args*, ***config*)
Bases: *pyreduce.reduce.CalibrationStep*, *pyreduce.reduce.ExtractionStep*

Extract the science spectra

load(*files*)
Load all science spectra from disk

Returns

- **heads** (*list(FITS header)*) – FITS headers of each observation
- **specs** (*list(array of shape (nord, ncol))*) – extracted spectra
- **sigmas** (*list(array of shape (nord, ncol))*) – uncertainties of the extracted spectra
- **columns** (*list(array of shape (nord, 2))*) – column ranges for each spectra

run(*files*, *bias*, *orders*, *norm_flat*, *curvature*, *mask*)
Extract Science spectra from observation

Parameters

- **files** (*list(str)*) – list of observations
- **bias** (*tuple*) – results from master bias step
- **orders** (*tuple*) – results from order tracing step
- **norm_flat** (*tuple*) – results from flat normalization
- **curvature** (*tuple*) – results from slit curvature step
- **mask** (*array of shape (nrow, ncol)*) – bad pixel map

Returns

- **heads** (*list(FITS header)*) – FITS headers of each observation
- **specs** (*list(array of shape (nord, ncol))*) – extracted spectra
- **sigmas** (*list(array of shape (nord, ncol))*) – uncertainties of the extracted spectra
- **columns** (*list(array of shape (nord, 2))*) – column ranges for each spectra

save(*fname*, *head*, *spec*, *sigma*, *column_range*)
Save the results of one extraction

Parameters

- **fname** (*str*) – filename to save to
- **head** (*FITS header*) – FITS header
- **spec** (*array of shape (nord, ncol)*) – extracted spectrum
- **sigma** (*array of shape (nord, ncol)*) – uncertainties of the extracted spectrum

- **column_range** (*array of shape (nord, 2)*) – range of columns that have spectrum

science_file(*name*)
Name of the science file in disk, based on the input file

Parameters **name** (*str*) – name of the observation file

Returns **name** – science file name

Return type str

class pyreduce.reduce.**SlitCurvatureDetermination**(**args*, ***config*)
Bases: *pyreduce.reduce.CalibrationStep*, *pyreduce.reduce.ExtractionStep*

Determine the curvature of the slit

curv_degree = None
Orders of the curvature to fit, currently supports only 1 and 2

Type int

curvature_mode = None
Whether to use 1d or 2d polynomials

Type {'1D', '2D'}

extraction_width = None
width of the orders in the extraction

Type float

fit_degree = None
Polynomial degree of the overall fit

Type int

load()
Load the curvature if possible, otherwise return None, None, i.e. use vertical extraction

Returns

- **tilt** (*array of shape (nord, ncol)*) – first order slit curvature at each point
- **shear** (*array of shape (nord, ncol)*) – second order slit curvature at each point

peak_function = None
Function shape that is fit to individual peaks

Type str

peak_threshold = None
peak finding noise threshold

Type float

peak_width = None
peak width

Type int

run(*files*, *orders*, *mask*, *bias*)
Determine the curvature of the slit

Parameters

- **files** (*list (str)*) – files to use for this

- **orders** (*tuple*) – results of the order tracing
- **mask** (*array of shape (nrow, ncol)*) – Bad pixel mask

Returns

- **tilt** (*array of shape (nord, ncol)*) – first order slit curvature at each point
- **shear** (*array of shape (nord, ncol)*) – second order slit curvature at each point

save(*tilt*, *shear*)
Save results from the curvature

Parameters

- **tilt** (*array of shape (nord, ncol)*) – first order slit curvature at each point
- **shear** (*array of shape (nord, ncol)*) – second order slit curvature at each point

savefile
Name of the tilt/shear save file

Type str

sigma_cutoff = None
how many sigma of bad lines to cut away

Type float

window_width = None
window width to search for peak in each row

Type float

class pyreduce.reduce.**Step**(*instrument*, *mode*, *target*, *night*, *output_dir*, *order_range*, ***config*)
Bases: object

Parent class for all steps

dependsOn
Steps that are required before running this step

Type list(str)

instrument = None
Name of the instrument

Type str

load()
Load results from a previous execution

If this raises a FileNotFoundError, run() will be used instead For calibration steps it is preferred however to print a warning and return None. Other modules can then use a default value instead.

Raises NotImplementedError – Needs to be implemented for each step

loadDependsOn
Steps that are required before loading data from this step

Type list(str)

mode = None
Name of the instrument mode

Type str

night = None
Date of the observation (as a string)

Type str

order_range = None
First and Last(+1) order to process

Type tuple(int, int)

output_dir
output directory, may contain tags {instrument}, {night}, {target}, {mode}

Type str

plot = None
Whether to plot the results or the progress of this step

Type bool

plot_title = None
Title used in the plots, if any

Type str

prefix
temporary file prefix

Type str

run(*files*, **args*)
Execute the current step

This should fail if files are missing or anything else goes wrong. If the user does not want to run this step, they should not specify it in steps.

Parameters **files** (*list(str)*) – data files required for this step

Raises NotImplementedError – needs to be implemented for each step

save(**args*)
Save the results of this step

Parameters ***args** (*obj*) – things to save

Raises NotImplementedError – Needs to be implemented for each step

target = None
Name of the observation target

Type str

class pyreduce.reduce.**WavelengthCalibrationFinalize**(**args*, ***config*)
Bases: *pyreduce.reduce.Step*

Perform wavelength calibration

degree = None
Polynomial degree of the wavelength calibration in order, column direction

Type tuple(int, int)

dimensionality = None
Whether to use 1d or 2d polynomials

Type {'1D', '2D'}

element = None
elements of the spectral lamp

Type str

iterations = None
Number of iterations in the remove lines, auto id cycle

Type int

load()
Load the results of the wavelength calibration

Returns

- **wave** (*array of shape (nord, ncol)*) – wavelength for each point in the spectrum
- **coef** (*array of shape (*ndegrees,)*) – polynomial coefficients of the wavelength fit
- **linelist** (*record array of shape (nlines,)*) – Updated line information for all lines

manual = None
Whether to use manual alignment instead of cross correlation

Type bool

medium = None
medium of the detector, vac or air

Type str

nstep = None
Number of detector offset steps, due to detector design

Type int

run(*wavecal_master*, *wavecal_init*)
Perform wavelength calibration

This consists of extracting the wavelength image and fitting a polynomial the the known spectral lines

Parameters

- **wavecal_master** (*tuple*) – results of the wavecal_master step, containing the master wavecal image and its header
- **wavecal_init** (LineList) – the initial LineList guess with the positions and wavelengths of lines

Returns

- **wave** (*array of shape (nord, ncol)*) – wavelength for each point in the spectrum
- **coef** (*array of shape (*ndegrees,)*) – polynomial coefficients of the wavelength fit
- **linelist** (*record array of shape (nlines,)*) – Updated line information for all lines

save(*wave*, *coef*, *linelist*)
Save the results of the wavelength calibration

Parameters

- **wave** (*array of shape (nord, ncol)*) – wavelength for each point in the spectrum
- **coef** (*array of shape (ndegrees,)*) – polynomial coefficients of the wavelength fit

- **linelist** (*record array of shape (nlines,)*) – Updated line information for all lines

savefile
Name of the wavelength echelle file

Type str

shift_window = None
fraction of columns, to allow individual orders to shift

Type float

threshold = None
residual threshold in m/s

Type float

class pyreduce.reduce.**WavelengthCalibrationInitialize**(**args*, ***config*)
Bases: *pyreduce.reduce.Step*

Create the initial wavelength solution file

cutoff = None
Minimum height of spectral lines in the normalized spectrum, values of 1 and above are interpreted as percentiles of the spectrum, set to 0 to disable the cutoff

Type float

degree = None
Polynomial degree of the wavelength calibration in order, column direction

Type tuple(int, int)

element = None
element for the atlas to use

Type str

load(*config*, *wavecal_master*)
Load results from a previous execution

If this raises a FileNotFoundError, run() will be used instead For calibration steps it is preferred however to print a warning and return None. Other modules can then use a default value instead.

Raises NotImplementedError – Needs to be implemented for each step

medium = None
medium the medium of the instrument, air or vac

Type str

nwalkers = None
number of walkers in the MCMC

Type int

resid_delta = None
resiudal range to accept as match between peaks and atlas in m/s

Type float

run(*wavecal_master*)
Execute the current step

This should fail if files are missing or anything else goes wrong. If the user does not want to run this step, they should not specify it in steps.

Parameters **files** (*list(str)*) – data files required for this step

Raises NotImplementedError – needs to be implemented for each step

save(*linelist*)
Save the results of this step

Parameters ***args** (*obj*) – things to save

Raises NotImplementedError – Needs to be implemented for each step

savefile
Name of the wavelength echelle file

Type str

smoothing = None
Gaussian smoothing parameter applied to the observed spectrum in pixel scale, set to 0 to disable smoothing

Type float

steps = None
number of steps in the MCMC

Type int

wave_delta = None
wavelength range around the initial guess to explore

Type float

class pyreduce.reduce.**WavelengthCalibrationMaster**(**args*, ***config*)
Bases: *pyreduce.reduce.CalibrationStep*, *pyreduce.reduce.ExtractionStep*

Create wavelength calibration master image

load()
Load master wavelength calibration from disk

Returns

- **thar** (*masked array of shape (nrow, ncol)*) – Master wavecal with bad pixel map applied
- **thead** (*FITS header*) – Master wavecal FITS header

run(*files*, *orders*, *mask*, *curvature*, *bias*, *norm_flat*)
Perform wavelength calibration

This consists of extracting the wavelength image and fitting a polynomial the the known spectral lines

Parameters

- **files** (*list(str)*) – wavelength calibration files
- **orders** (*tuple(array, array)*) – Polynomial coefficients of each order, and columns with signal of each order
- **mask** (*array of shape (nrow, ncol)*) – Bad pixel mask

Returns

- **wave** (*array of shape (nord, ncol)*) – wavelength for each point in the spectrum
- **thar** (*array of shape (nrow, ncol)*) – extracted wavelength calibration image

- **coef** (*array of shape (*ndegrees,)*) – polynomial coefficients of the wavelength fit
- **linelist** (*record array of shape (nlines,)*) – Updated line information for all lines

save (*thar*, *thead*)
Save the master wavelength calibration to a FITS file

Parameters

- **thar** (*array of shape (nrow, ncol)*) – master flat data
- **thead** (*FITS header*) – master flat header

savefile
Name of the wavelength echelle file

Type str

pyreduce.reduce.**main** (*instrument*, *target*, *night=None*, *modes=None*, *steps='all'*, *base_dir=None*, *input_dir=None*, *output_dir=None*, *configuration=None*, *order_range=None*, *allow_calibration_only=False*, *skip_existing=False*)
Main entry point for REDUCE scripts, default values can be changed as required if reduce is used as a script Finds input directories, and loops over observation nights and instrument modes

Parameters

- **instrument** (*str, list[str]*) – instrument used for the observation (e.g. UVES, HARPS)
- **target** (*str, list[str]*) – the observed star, as named in the folder structure/fits headers
- **night** (*str, list[str]*) – the observation nights to reduce, as named in the folder structure. Accepts bash wildcards (i.e. *, ?), but then relies on the folder structure for restricting the nights
- **modes** (*str, list[str], dict[{instrument}:list], None, optional*) – the instrument modes to use, if None will use all known modes for the current instrument. See instruments for possible options
- **steps** (*tuple(str), "all", optional*) – which steps of the reduction process to perform the possible steps are: "bias", "flat", "orders", "norm_flat", "wavecal", "science" alternatively set steps to "all", which is equivalent to setting all steps Note that the later steps require the previous intermediary products to exist and raise an exception otherwise
- **base_dir** (*str, optional*) – base data directory that Reduce should work in, is prefixxed on input_dir and output_dir (default: use settings_pyreduce.json)
- **input_dir** (*str, optional*) – input directory containing raw files. Can contain placeholders {instrument}, {target}, {night}, {mode} as well as wildcards. If relative will use base_dir as root (default: use settings_pyreduce.json)
- **output_dir** (*str, optional*) – output directory for intermediary and final results. Can contain placeholders {instrument}, {target}, {night}, {mode}, but no wildcards. If relative will use base_dir as root (default: use settings_pyreduce.json)
- **configuration** (*dict[str:obj], str, list[str], dict[{instrument}:dict,str], optional*) – configuration file for the current run, contains parameters for different parts of reduce. Can be a path to a json file, or a dict with configurations for the different instruments. When a list, the order must be the same as instruments (default: settings_{instrument.upper()}.json)

6.1.13 pyreduce.trace_orders module

Find clusters of pixels with signal And combine them into continous orders

pyreduce.trace_orders.**best_fit**(*x*, *y*)

pyreduce.trace_orders.**calculate_mean_cluster_thickness**(*x*, *y*)

pyreduce.trace_orders.**combine**(*i*, *j*, *x*, *y*, *merge*, *mct*, *nrow*, *ncol*, *deg*, *threshold*)

pyreduce.trace_orders.**create_merge_array**(*x*, *y*, *mean_cluster_thickness*, *nrow*, *ncol*, *deg*, *threshold*)

pyreduce.trace_orders.**delete**(*i*, *x*, *y*, *merge*)

pyreduce.trace_orders.**determine_overlap_rating**(*xi*, *yi*, *xj*, *yj*, *mean_cluster_thickness*, *nrow*, *ncol*, *deg=2*)

pyreduce.trace_orders.**fit**(*x*, *y*, *deg*, *regularization=0*)

pyreduce.trace_orders.**fit_polynomials_to_clusters**(*x*, *y*, *clusters*, *degree*, *regularization=0*)

Fits a polynomial of degree opower to points x, y in cluster clusters

Parameters

- **x** (*dict(int: array)*) – x coordinates seperated by cluster
- **y** (*dict(int: array)*) – y coordinates seperated by cluster
- **clusters** (*list(int)*) – cluster labels, equivalent to x.keys() or y.keys()
- **degree** (*int*) – degree of polynomial fit

Returns **orders** – coefficients of polynomial fit for each cluster

Return type dict(int, array[degree+1])

pyreduce.trace_orders.**mark_orders**(*im*, *min_cluster=None*, *min_width=None*, *filter_size=None*, *noise=None*, *opower=4*, *border_width=None*, *degree_before_merge=2*, *regularization=0*, *closing_shape=(5, 5)*, *opening_shape=(2, 2)*, *plot=False*, *plot_title=None*, *manual=True*, *auto_merge_threshold=0.9*, *merge_min_threshold=0.1*, *sigma=0*)

Identify and trace orders

Parameters

- **im** (*array[nrow, ncol]*) – order definition image
- **min_cluster** (*int, optional*) – minimum cluster size in pixels (default: 500)
- **filter_size** (*int, optional*) – size of the running filter (default: 120)
- **noise** (*float, optional*) – noise to filter out (default: 8)
- **opower** (*int, optional*) – polynomial degree of the order fit (default: 4)
- **border_width** (*int, optional*) – number of pixels at the bottom and top borders of the image to ignore for order tracing (default: 5)
- **plot** (*bool, optional*) – wether to plot the final order fits (default: False)
- **manual** (*bool, optional*) – wether to manually select clusters to merge (strongly recommended) (default: True)

Returns **orders** – order tracing coefficients (in numpy order, i.e. largest exponent first)

Return type array[nord, opower+1]

pyreduce.trace_orders.**merge_clusters**(*img*, *x*, *y*, *n_clusters*, *manual=True*, *deg=2*, *auto_merge_threshold=0.9*, *merge_min_threshold=0.1*, *plot_title=None*)

Merge clusters that belong together

Parameters

- **img** (*array[nrow, ncol]*) – the image the order trace is based on
- **orders** (*dict(int, array(float))*) – coefficients of polynomial fits to clusters
- **x** (*dict(int, array(int))*) – x coordinates of cluster points
- **y** (*dict(int, array(int))*) – y coordinates of cluster points
- **n_clusters** (*array(int)*) – cluster numbers
- **threshold** (*int, optional*) – overlap threshold for merging clusters (the default is 100)
- **manual** (*bool, optional*) – if True ask before merging orders

Returns

- **x** (*dict(int: array)*) – x coordinates of clusters, key=cluster id
- **y** (*dict(int: array)*) – y coordinates of clusters, key=cluster id
- **n_clusters** (*int*) – number of identified clusters

pyreduce.trace_orders.**plot_order**(*i*, *j*, *x*, *y*, *img*, *deg*, *title=''*)

Plot a single order

pyreduce.trace_orders.**plot_orders**(*im*, *x*, *y*, *clusters*, *orders*, *order_range*, *title=None*)

Plot orders and image

pyreduce.trace_orders.**update_merge_array**(*merge*, *x*, *y*, *j*, *mean_cluster_thickness*, *nrow*, *ncol*, *deg*, *threshold*)

6.1.14 pyreduce.util module

Collection of various useful and/or reoccuring functions across PyReduce

pyreduce.util.**air2vac**(*wl_air*)

Convert wavelengths in air to vacuum wavelength Author: Nikolai Piskunov

pyreduce.util.**bezier_interp**(*x_old*, *y_old*, *x_new*)

Bezier interpolation, based on the scipy methods

This mostly sanitizes the input by removing masked values and duplicate entries Note that in case of duplicate entries (in x_old) the results are not well defined as only one of the entries is used and the other is discarded

Parameters

- **x_old** (*array[n]*) – old x values
- **y_old** (*array[n]*) – old y values
- **x_new** (*array[m]*) – new x values

Returns **y_new** – new y values

Return type array[m]

pyreduce.util.**bottom**(*f*, *order=1*, *iterations=40*, *eps=0.001*, *poly=False*, *weight=1*, ***kwargs*)
bottom tries to fit a smooth curve to the lower envelope of 1D data array f. Filter size "filter" together with the total number of iterations determine the smoothness and the quality of the fit. The total number of iterations can be controlled by limiting the maximum number of iterations (iter) and/or by setting the convergence criterion for the fit (eps) 04-Nov-2000 N.Piskunov wrote. 09-Nov-2011 NP added weights and 2nd derivative constraint as LAM2

Parameters

- **f** (*Callable*) – Function to fit
- **filter** (*int*) – Smoothing parameter of the optimal filter (or polynomial degree of poly is True)
- **iter** (*int*) – maximum number of iterations [def: 40]
- **eps** (*float*) – convergence level [def: 0.001]
- **mn** (*float*) – minimum function values to be considered [def: min(f)]
- **mx** (*float*) – maximum function values to be considered [def: max(f)]
- **lam2** (*float*) – constraint on 2nd derivative
- **weight** (*array (float)*) – vector of weights.

pyreduce.util.**cutout_image**(*img*, *ymin*, *ymax*, *xmin*, *xmax*)
Cut a section of an image out

Parameters

- **img** (*array*) – image
- **ymin** (*array[ncol] (int)*) – lower y value
- **ymax** (*array[ncol] (int)*) – upper y value
- **xmin** (*int*) – lower x value
- **xmax** (*int*) – upper x value

Returns **cutout** – selection of the image

Return type array[height, ncol]

pyreduce.util.**find_first_index**(*arr*, *value*)
find the first element equal to value in the array arr

pyreduce.util.**gaussbroad**(*x*, *y*, *hwhm*)
Apply gaussian broadening to x, y data with half width half maximum hwhm

Parameters

- **x** (*array (float)*) – x values
- **y** (*array (float)*) – y values
- **hwhm** (*float > 0*) – half width half maximum

Returns broadened y values

Return type array(float)

pyreduce.util.**gaussfit**(*x*, *y*)
Fit a simple gaussian to data

gauss(x, a, mu, sigma) = a * exp(-z**2/2) with z = (x - mu) / sigma

Parameters

- **x** (*array (float)*) – x values
- **y** (*array (float)*) – y values

Returns fitted values for x, fit paramters (a, mu, sigma)

Return type gauss(x), parameters

pyreduce.util.**gaussfit2**(*x*, *y*)

Fit a gaussian(normal) curve to data x, y

gauss = A * exp(-(x-mu)**2/(2*sig**2)) + offset

Parameters

- **x** (*array [n]*) – x values
- **y** (*array [n]*) – y values

Returns **popt** – coefficients of the gaussian: A, mu, sigma**2, offset

Return type array[4]

pyreduce.util.**gaussfit3**(*x*, *y*)

A very simple (and relatively fast) gaussian fit gauss = A * exp(-(x-mu)**2/(2*sig**2)) + offset

Parameters

- **x** (*array of shape (n,)*) – x data
- **y** (*array of shape (n,)*) – y data

Returns **popt** – Parameters A, mu, sigma**2, offset

Return type list of shape (4,)

pyreduce.util.**gaussfit4**(*x*, *y*)

A very simple (and relatively fast) gaussian fit gauss = A * exp(-(x-mu)**2/(2*sig**2)) + offset

Assumes x is sorted

Parameters

- **x** (*array of shape (n,)*) – x data
- **y** (*array of shape (n,)*) – y data

Returns **popt** – Parameters A, mu, sigma**2, offset

Return type list of shape (4,)

pyreduce.util.**gaussfit_linear**(*x*, *y*)

Transform the gaussian fit into a linear least squares problem, and solve that instead of the non-linear curve fit For efficiency reasons. (roughly 10 times faster than the curve fit)

Parameters

- **x** (*array of shape (n,)*) – x data
- **y** (*array of shape (n,)*) – y data

Returns **coef** – a, mu, sig, 0

Return type tuple

pyreduce.util.**gaussval2**(*x*, *a*, *mu*, *sig*, *const*)

pyreduce.util.**gridsearch**(*func*, *grid*, *args=()*, *kwargs={}*)

pyreduce.util.**helcorr**(*obs_long*, *obs_lat*, *obs_alt*, *ra2000*, *dec2000*, *jd*, *system='barycentric'*)
calculates heliocentric Julian date, barycentric and heliocentric radial velocity corrections, using astropy functions

Parameters

- **obs_long** (*float*) – Longitude of observatory (degrees, western direction is positive)
- **obs_lat** (*float*) – Latitude of observatory (degrees)
- **obs_alt** (*float*) – Altitude of observatory (meters)
- **ra2000** (*float*) – Right ascension of object for epoch 2000.0 (hours)
- **dec2000** (*float*) – Declination of object for epoch 2000.0 (degrees)
- **jd** (*float*) – Julian date for the middle of exposure in MJD
- **system** (*{"barycentric", "heliocentric"}, optional*) – reference system of the result, barycentric: around earth-sun gravity center, heliocentric: around sun, usually barycentric is preferred (default: "barycentric)

Returns

- **correction** (*float*) – radial velocity correction due to barycentre offset
- **hjd** (*float*) – Heliocentric Julian date for middle of exposure

pyreduce.util.**in_ipynb**()

pyreduce.util.**interpolate_masked**(*masked*)
Interpolate masked values, from non masked values

Parameters **masked** (*masked_array*) – masked array to interpolate on

Returns **interpolated** – interpolated non masked array

Return type array

pyreduce.util.**log_version**()
For Debug purposes

pyreduce.util.**make_index**(*ymin*, *ymax*, *xmin*, *xmax*, *zero=0*)
Create an index (numpy style) that will select part of an image with changing position but fixed height

The user is responsible for making sure the height is constant, otherwise it will still work, but the subsection will not have the desired format

Parameters

- **ymin** (*array[ncol](int)*) – lower y border
- **ymax** (*array[ncol](int)*) – upper y border
- **xmin** (*int*) – leftmost column
- **xmax** (*int*) – rightmost colum
- **zero** (*bool, optional*) – if True count y array from 0 instead of xmin (default: False)

Returns **index** – numpy index for the selection of a subsection of an image

Return type tuple(array[height, width], array[height, width])

pyreduce.util.**middle**(*f*, *param*, *x=None*, *iterations=40*, *eps=0.001*, *poly=False*, *weight=1*, *lambda2=-1*, *mn=None*, *mx=None*)
middle tries to fit a smooth curve that is located along the "middle" of 1D data array f. Filter size "filter" together with the total number of iterations determine the smoothness and the quality of the fit. The total

number of iterations can be controlled by limiting the maximum number of iterations (iter) and/or by setting the convergence criterion for the fit (eps) 04-Nov-2000 N.Piskunov wrote. 09-Nov-2011 NP added weights and 2nd derivative constraint as LAM2

Parameters

- **f** (*Callable*) – Function to fit
- **filter** (*int*) – Smoothing parameter of the optimal filter (or polynomial degree of poly is True)
- **iter** (*int*) – maximum number of iterations [def: 40]
- **eps** (*float*) – convergence level [def: 0.001]
- **mn** (*float*) – minimum function values to be considered [def: min(f)]
- **mx** (*float*) – maximum function values to be considered [def: max(f)]
- **lam2** (*float*) – constraint on 2nd derivative
- **weight** (*array (float)*) – vector of weights.

pyreduce.util.**opt_filter**(*y*, *par*, *par1=None*, *weight=None*, *lambda2=-1*, *maxiter=100*)

Optimal filtering of 1D and 2D arrays. Uses tridiag in 1D case and sprsin and linbcg in 2D case. Written by N.Piskunov 8-May-2000

Parameters

- **f** (*array*) – 1d or 2d array
- **xwidth** (*int*) – filter width (for 2d array width in x direction (1st index)
- **ywidth** (*int*) – (for 2d array only) filter width in y direction (2nd index) if ywidth is missing for 2d array, it set equal to xwidth
- **weight** (*array (float)*) – an array of the same size(s) as f containing values between 0 and 1
- **lambda1** (*float*) – regularization parameter
- **maxiter** (*int*) – maximum number of iteration for filtering of 2d array

pyreduce.util.**plot2d**(*x*, *y*, *z*, *coeff*, *title=None*)

pyreduce.util.**polyfit1d**(*x*, *y*, *degree=1*, *regularization=0*)

pyreduce.util.**polyfit2d**(*x*, *y*, *z*, *degree=1*, *max_degree=None*, *scale=True*, *plot=False*, *plot_title=None*)

A simple 2D plynomial fit to data x, y, z The polynomial can be evaluated with numpy.polynomial.polynomial.polyval2d

Parameters

- **x** (*array[n]*) – x coordinates
- **y** (*array[n]*) – y coordinates
- **z** (*array[n]*) – data values
- **degree** (*int, optional*) – degree of the polynomial fit (default: 1)
- **max_degree** (*{int, None}, optional*) – if given the maximum combined degree of the coefficients is limited to this value
- **scale** (*bool, optional*) – Wether to scale the input arrays x and y to mean 0 and variance 1, to avoid numerical overflows. Especially useful at higher degrees. (default: True)

- **plot** (*bool, optional*) – wether to plot the fitted surface and data (slow) (default: False)

Returns **coeff** – the polynomial coefficients in numpy 2d format, i.e. coeff[i, j] for x**i * y**j

Return type array[degree+1, degree+1]

pyreduce.util.**polyfit2d_2**(*x*, *y*, *z*, *degree=1*, *x0=None*, *loss='arctan'*, *method='trf'*, *plot=False*)

pyreduce.util.**polyscale2d**(*coeff*, *scale_x*, *scale_y*, *copy=True*)

pyreduce.util.**polyshift2d**(*coeff*, *offset_x*, *offset_y*, *copy=True*)

pyreduce.util.**polyvander2d**(*x*, *y*, *degree*)

pyreduce.util.**remove_bias**(*img*, *ihead*, *bias*, *bhead*, *nfiles=1*)

pyreduce.util.**resample**(*array*, *new_size*)

pyreduce.util.**safe_interpolation**(*x_old*, *y_old*, *x_new=None*, *fill_value=0*)

'Safe' interpolation method that should avoid the common pitfalls of spline interpolation

masked arrays are compressed, i.e. only non masked entries are used remove NaN input in x_old and y_old only unique x values are used, corresponding y values are 'random' if all else fails, revert to linear interpolation

Parameters

- **x_old** (*array of size (n,)*) – x values of the data
- **y_old** (*array of size (n,)*) – y values of the data
- **x_new** (*array of size (m,) or None, optional*) – x values of the interpolated values if None will return the interpolator object (default: None)

Returns **y_new** – if x_new was given, return the interpolated values otherwise return the interpolator object

Return type array of size (m,) or interpolator

pyreduce.util.**start_logging**(*log_file='log.log'*)

Start logging to log file and command line

Parameters **log_file** (*str, optional*) – name of the logging file (default: "log.log")

pyreduce.util.**swap_extension**(*fname*, *ext*, *path=None*)

exchange the extension of the given file with a new one

pyreduce.util.**top**(*f*, *order=1*, *iterations=40*, *eps=0.001*, *poly=False*, *weight=1*, *lambda2=-1*, *mn=None*, *mx=None*)

top tries to fit a smooth curve to the upper envelope of 1D data array f. Filter size "filter" together with the total number of iterations determine the smoothness and the quality of the fit. The total number of iterations can be controlled by limiting the maximum number of iterations (iter) and/or by setting the convergence criterion for the fit (eps) 04-Nov-2000 N.Piskunov wrote. 09-Nov-2011 NP added weights and 2nd derivative constraint as LAM2

Parameters

- **f** (*Callable*) – Function to fit
- **filter** (*int*) – Smoothing parameter of the optimal filter (or polynomial degree of poly is True)
- **iter** (*int*) – maximum number of iterations [def: 40]
- **eps** (*float*) – convergence level [def: 0.001]
- **mn** (*float*) – minimum function values to be considered [def: min(f)]

- **mx** (*float*) – maximum function values to be considered [def: max(f)]
- **lam2** (*float*) – constraint on 2nd derivative
- **weight** (*array (float)*) – vector of weights.

pyreduce.util.**vac2air**(*wl_vac*)
Convert vacuum wavelengths to wavelengths in air Author: Nikolai Piskunov

6.1.15 pyreduce.wavelength_calibration module

Wavelength Calibration by comparison to a reference spectrum Loosely bases on the IDL wavecal function

class pyreduce.wavelength_calibration.**AlignmentPlot**(*ax*, *obs*, *lines*, *offset=(0, 0)*, *plot_title=None*)
Bases: object

Makes a plot which can be clicked to align the two spectra, reference and observed

connect()
connect the click event with the appropiate function

make_ref_image()
create and show the reference plot, with the two spectra

on_click(*event*)
On click offset the reference by the distance between click positions

class pyreduce.wavelength_calibration.**LineAtlas**(*element*, *medium='vac'*)
Bases: object

load_fits(*fname*)

class pyreduce.wavelength_calibration.**LineList**(*lines=None*)
Bases: object

add_line(*wave*, *order*, *pos*, *width*, *height*, *flag*)

append(*linelist*)

dtype = dtype((numpy.record, [(('wlc', 'WLC'), '>f8'), (('wll', 'WLL'), '>f8'), (('pos

classmethod from_list(*wave*, *order*, *pos*, *width*, *height*, *flag*)

classmethod load(*filename*)

save(*filename*)

class pyreduce.wavelength_calibration.**WavelengthCalibration**(*threshold=100*, *degree=(6, 6)*, *iterations=3*, *dimensionality='2D'*, *nstep=0*, *shift_window=0.01*, *manual=False*, *polarim=False*, *lfc_peak_width=3*, *closing=5*, *element=None*, *medium='vac'*, *plot=True*, *plot_title=None*)
Bases: object

Wavelength Calibration Module

Takes an observed wavelength image and the reference linelist and returns the wavelength at each pixel

align(*obs*, *lines*)
Align the observation with the reference spectrum Either automatically using cross correlation or manually (visually)

Parameters

- **obs** (*array[nrow, ncol]*) – observed wavelength calibration spectrum (e.g. obs=ThoriumArgon)
- **lines** (*struct_array*) – reference line data
- **manual** (*bool, optional*) – wether to manually align the spectra (default: False)
- **plot** (*bool, optional*) – wether to plot the alignment (default: False)

Returns **offset** – offset in order and column

Return type tuple(int, int)

align_manual(*obs*, *lines*)
Open an AlignmentPlot window for manual selection of the alignment

Parameters

- **obs** (*array of shape (nord, ncol)*) – observed image
- **lines** (*recarray of shape (nlines,)*) – reference linelist

Returns **offset** – offset in order and column to be applied to each line in the linelist

Return type tuple(int, int)

apply_alignment_offset(*lines*, *offset*, *select=None*)
Apply an offset to the linelist

Parameters

- **lines** (*recarray of shape (nlines,)*) – reference linelist
- **offset** (*tuple(int, int)*) – offset in (order, column)
- **select** (*array of shape(nlines,), optional*) – Mask that defines which lines the offset applies to

Returns **lines** – linelist with offset applied

Return type recarray of shape (nlines,)

auto_id(*obs*, *wave_img*, *lines*)
Automatically identify peaks that are close to known lines

Parameters

- **obs** (*array of shape (nord, ncol)*) – observed spectrum
- **wave_img** (*array of shape (nord, ncol)*) – wavelength solution image
- **lines** (*struc_array*) – line data
- **threshold** (*int, optional*) – difference threshold between line positions in m/s, until which a line is considered identified (default: 1)
- **plot** (*bool, optional*) – wether to plot the new lines

Returns **lines** – line data with new flags

Return type struct_array

build_2d_solution(*lines*, *plot=False*)

Create a 2D polynomial fit to flagged lines degree : tuple(int, int), optional

polynomial degree of the fit in (column, order) dimension (default: (6, 6))

Parameters

- **lines** (*struc_array*) – line data
- **plot** (*bool, optional*) – wether to plot the solution (default: False)

Returns **coef** – 2d polynomial coefficients

Return type array[degree_x, degree_y]

build_step_solution(*lines*, *plot=False*)

Fit the least squares fit to the wavelength points, with additional free parameters for detector gaps, e.g. due to stitching.

The exact method of the fit depends on the dimensionality. Either way we are using the usual polynomial fit for the wavelength, but the x points are modified beforehand by shifting them some amount, at specific indices. We assume that the stitching effects are distributed evenly and we know how many steps we expect (this is set as "nstep").

Parameters

- **lines** (*np.recarray*) – linedata
- **plot** (*bool, optional*) – whether to plot results or not, by default False

Returns coefficients of the best fit

Return type coef

calculate_AIC(*lines*, *wave_solution*)

calculate_residual(*wave_solution*, *lines*)

Calculate all residuals of all given lines

Residual = (Wavelength Solution - Expected Wavelength) / Expected Wavelength * speed of light

Parameters

- **wave_solution** (*array of shape (degree_x, degree_y)*) – polynomial coefficients of the wavelength solution (in numpy format)
- **lines** (*recarray of shape (nlines,)*) – contains the position of the line on the detector (posm), the order (order), and the expected wavelength (wll)

Returns **residual** – Residual of each line in m/s

Return type array of shape (nlines,)

closing = None

grey closing range for the input image

Type int

create_image_from_lines(*lines*)

Create a reference image based on a line list Each line will be approximated by a Gaussian Space inbetween lines is 0 The number of orders is from 0 to the maximum order

Parameters **lines** (*recarray of shape (nlines,)*) – line data

Returns **img** – New reference image

Return type array of shape (nord, ncol)

degree = None
polynomial degree of the wavelength fit in (pixel, order) direction

Type tuple(int, int)

dimensionality
Whether to use 1d or 2d fit

Type {"1D", "2D"}

element = None
Elements used in the wavelength calibration. Used in AutoId to find more lines from the Atlas

Type str

evaluate_solution(*pos*, *order*, *solution*)
Evaluate the 1d or 2d wavelength solution at the given pixel positions and orders

Parameters

- **pos** (*array*) – pixel position on the detector (i.e. x axis)
- **order** (*array*) – order of each point
- **solution** (*array of shape (nord, ndegree) or (degree_x, degree_y)*) – polynomial coefficients. For mode=1D, one set of coefficients per order. For mode=2D, the first dimension is for the positions and the second for the orders
- **mode** (*str, optional*) – Wether to interpret the solution as 1D or 2D polynomials, by default "1D"

Returns **result** – Evaluated polynomial

Return type array

Raises `ValueError` – If pos and order have different shapes, or mode is of the wrong value

evaluate_step_solution(*pos*, *order*, *solution*)

execute(*obs*, *lines*)
Perform the whole wavelength calibration procedure with the current settings

Parameters

- **obs** (*array of shape (nord, ncol)*) – observed image
- **lines** (*recarray of shape (nlines,)*) – reference linelist

Returns **wave_img** – Wavelength solution for each pixel

Return type array of shape (nord, ncol)

Raises `NotImplementedError` – If polarimitry flag is set

f(*x*, *poly_coef*, *step_coef_pos*, *step_coef_diff*)

fit_lines(*obs*, *lines*)
Determine exact position of each line on the detector based on initial guess

This fits a Gaussian to each line, and uses the peak position as a new solution

Parameters

- **obs** (*array of shape (nord, ncol)*) – observed wavelength calibration image

- **lines** (*recarray of shape (nlines,)*) – reference line data

Returns **lines** – Updated line information (posm is changed)

Return type recarray of shape (nlines,)

g(*x*, *step_coef_pos*, *step_coef_diff*)

iterations = None
Number of iterations in the remove residuals, auto id, loop

Type int

lfc_peak_width = None
Laser Frequency Peak width (for scipy.signal.find_peaks)

Type int

make_wave(*wave_solution*, *plot=False*)
Expand polynomial wavelength solution into full image

Parameters

- **wave_solution** (*array of shape(degree,)*) – polynomial coefficients of wavelength solution
- **plot** (*bool, optional*) – wether to plot the solution (default: False)

Returns **wave_img** – wavelength solution for each point in the spectrum

Return type array of shape (nord, ncol)

manual = None
Whether to manually align the reference instead of using cross correlation

Type bool

medium = None
Medium of the detector, vac or air

Type str

ncol = None
Number of columns in the observation

Type int

nord = None
Number of orders in the observation

Type int

normalize(*obs*, *lines*)
Normalize the observation and reference list in each order individually Copies the data if the image, but not of the linelist

Parameters

- **obs** (*array of shape (nord, ncol)*) – observed image
- **lines** (*recarray of shape (nlines,)*) – reference linelist

Returns

- **obs** (*array of shape (nord, ncol)*) – normalized image
- **lines** (*recarray of shape (nlines,)*) – normalized reference linelist

nstep = None
Whether to fit for pixel steps (offsets) in the detector

Type bool

plot = None
Whether to plot the results. Set to 2 to plot during all steps.

Type int

plot_residuals(*lines*, *coef*, *title='Residuals'*)

plot_results(*wave_img*, *obs*)

polarim = None
Whether to use polarimetric orders instead of the usual ones. I.e. Each pair of two orders represents the same data. Not Supported yet

Type bool

reject_lines(*lines*, *plot=False*)
Reject the largest outlier one by one until all residuals are lower than the threshold

Parameters

- **lines** (*recarray of shape (nlines,)*) – Line data with pixel position, and expected wavelength
- **threshold** (*float, optional*) – upper limit for the residual, by default 100
- **degree** (*tuple, optional*) – polynomial degree of the wavelength solution (pixel, column) (default: (6, 6))
- **plot** (*bool, optional*) – Wether to plot the results (default: False)

Returns **lines** – Line data with updated flags

Return type recarray of shape (nlines,)

reject_outlier(*residual*, *lines*)
Reject the strongest outlier

Parameters

- **residual** (*array of shape (nlines,)*) – residuals of all lines
- **lines** (*recarray of shape (nlines,)*) – line data

Returns

- **lines** (*struct_array*) – line data with one more flagged line
- **residual** (*array of shape (nlines,)*) – residuals of each line, with outliers masked (including the new one)

shift_window = None
Fraction if the number of columns to use in the alignment of individual orders. Set to 0 to disable

Type float

step_mode

threshold = None
Residual threshold in m/s above which to remove lines

Type float

class pyreduce.wavelength_calibration.**WavelengthCalibrationComb**(*threshold=100*, *degree=(6, 6)*, *iterations=3*, *dimensionality='2D'*, *nstep=0*, *shift_window=0.01*, *manual=False*, *polarim=False*, *lfc_peak_width=3*, *closing=5*, *element=None*, *medium='vac'*, *plot=True*, *plot_title=None*)

Bases: *pyreduce.wavelength_calibration.WavelengthCalibration*

execute(*comb*, *wave*, *lines=None*)

Perform the whole wavelength calibration procedure with the current settings

Parameters

- **obs** (*array of shape (nord, ncol)*) – observed image
- **lines** (*recarray of shape (nlines,)*) – reference linelist

Returns **wave_img** – Wavelength solution for each pixel

Return type array of shape (nord, ncol)

Raises `NotImplementedError` – If polarimitry flag is set

class pyreduce.wavelength_calibration.**WavelengthCalibrationInitialize**(*degree=2*, *plot=False*, *plot_title='Wavecal Initial'*, *wave_delta=20*, *nwalkers=100*, *steps=50000*, *resid_delta=1000*, *cutoff=5*, *smoothing=0*, *element='thar'*, *medium='vac'*)

Bases: *pyreduce.wavelength_calibration.WavelengthCalibration*

create_new_linelist_from_solution(*spectrum*, *wavelength*, *atlas*, *order*) → pyreduce.wavelength_calibration.LineList

Create a new linelist based on an existing wavelength solution for a spectrum, and a line atlas with known lines. The linelist is the one used by the rest of PyReduce wavelength calibration.

Observed lines are matched with the lines in the atlas to improve the wavelength solution.

Parameters

- **spectrum** (*array*) – Observed spectrum at each pixel
- **wavelength** (*array*) – Wavelength of spectrum at each pixel
- **atlas** (LineAtlas) – Atlas with wavelength of known lines
- **order** (*int*) – Order of the spectrum within the detector
- **resid_delta** (*float, optional*) – Maximum residual allowed between a peak and the closest line in the atlas, to still match them, in m/s, by default 1000.

Returns **linelist** – new linelist with lines from this order

Return type *LineList*

cutoff = None
minimum value in the spectrum to be considered a spectral line, if the value is above (or equal 1) it defines the percentile of the spectrum

Type float

determine_wavelength_coefficients(*spectrum*, *atlas*, *wave_range*) → numpy.ndarray
Determines the wavelength polynomial coefficients of a spectrum, based on an line atlas with known spectral lines, and an initial guess for the wavelength range. The calculation uses an MCMC approach to sample the probability space and find the best cross correlation value, between observation and atlas.

Parameters

- **spectrum** (*array*) – observed spectrum at each pixel
- **atlas** (LineAtlas) – atlas containing a known spectrum with wavelength and flux
- **wave_range** (*2-tuple*) – initial wavelength guess (begin, end)
- **degrees** (*int, optional*) – number of degrees of the wavelength polynomial, lower numbers yield better results, by default 2
- **w_range** (*float, optional*) – uncertainty on the initial wavelength guess in Angstrom, by default 20
- **nwalkers** (*int, optional*) – number of walkers for the MCMC, more is better but increases the time, by default 100
- **steps** (*int, optional*) – number of steps in the MCMC per walker, more is better but increases the time, by default 20_000
- **plot** (*bool, optional*) – whether to plot the results or not, by default False

Returns **coef** – polynomial coefficients in numpy order

Return type array

execute(*spectrum*, *wave_range*) → pyreduce.wavelength_calibration.LineList
Perform the whole wavelength calibration procedure with the current settings

Parameters

- **obs** (*array of shape (nord, ncol)*) – observed image
- **lines** (*recarray of shape (nlines,)*) – reference linelist

Returns **wave_img** – Wavelength solution for each pixel

Return type array of shape (nord, ncol)

Raises NotImplementedError – If polarimitry flag is set

get_cutoff(*spectrum*)

normalize(*spectrum*)
Normalize the observation and reference list in each order individually Copies the data if the image, but not of the linelist

Parameters

- **obs** (*array of shape (nord, ncol)*) – observed image
- **lines** (*recarray of shape (nlines,)*) – reference linelist

Returns

- **obs** (*array of shape (nord, ncol)*) – normalized image
- **lines** (*recarray of shape (nlines,)*) – normalized reference linelist

nwalkers = None
number of walkers in the MCMC

Type int

resid_delta = None
residual uncertainty allowed when matching observation with known lines

Type float

smoothing = None
gaussian smoothing applied to the wavecal spectrum before the MCMC in pixel scale, disable it by setting it to 0

Type float

steps = None
number of steps in the MCMC

Type int

wave_delta = None
wavelength uncertainty on the initial guess in Angstrom

Type float

pyreduce.wavelength_calibration.**polyfit**(*x*, *y*, *deg*)

6.1.16 Module contents

class pyreduce.**TqdmLoggingHandler**(*level=0*)
Bases: logging.Handler

emit(*record*)
Do whatever it takes to actually log the specified logging record.

This version is intended to be implemented by subclasses and so raises a NotImplementedError.

CHAPTER 7

Wavelength Calibration

Wavelength calibration in PyReduce happens in multiple steps.

7.1 Initial Linelist

To start the wavelength calibration we need an initial guess. PyReduce provides a number of initial guess files in the wavecal directory for the supported instruments and modes. These files are numpy ".npz" archives that contain a recarray with the key "cs_lines", which has the following datatype:

- (("wlc", "WLC"), ">f8"), # Wavelength (before fit)
- (("wll", "WLL"), ">f8"), # Wavelength (after fit)
- (("posc", "POSC"), ">f8"), # Pixel Position (before fit)
- (("posm", "POSM"), ">f8"), # Pixel Position (after fit)
- (("xfirst", "XFIRST"), ">i2"), # first pixel of the line
- (("xlast", "XLAST"), ">i2"), # last pixel of the line
- (("approx", "APPROX"), "O"), # ???
- (("width", "WIDTH"), ">f8"), # width of the line in pixels
- (("height", "HEIGHT"), ">f8"), # relative strength of the line
- (("order", "ORDER"), ">i2"), # echelle order the line is found in
- ("flag", "?"), # flag that tells us if we should use the line or not

If such a file is not available, or you want to create a new one, it is possible to do so just based on a rough initial guess of the wavelength ranges of each order¨ as well as using a reference atlas of known spectral lines, for the gas lamp used in the wavelength calibration.

First create the master wavelength calibration spectrum by running the "wavecal_master" step. Then you can use the "wavecal_creator.py" script in tools to create the linelist, by providing rough guesses for the wavelength (by default uncertainties of up to 20 Angstrom are allowed). Alternatively you can run the "wavecal_init" step in PyReduce, assum-

ing that the current instrument provides the correct initial wavelength guess when calling the "get_wavelength_range" function.

Either way, this will start a search for the best fit between the observation and the known atlas using MCMC. This approach is unfortunately quite slow but should be stable. It is therefore recommended that the created linelist is placed in the location provided by the "get_wavecal_filename" function of the instrument, and the "wavecal_init" step is only run once.

The MCMC determines the best fit based on a combination of the cross correlation and the least squares match between the two spectra.

References for the included spectra:

ThAr Palmer, B.A. and Engleman, R., Jr., 1983, Atlas of the Thorium Spectrum, Los Alamos National Laboratory, Los Alamos, New Mexico Norlén, G., 1973, Physica Scripta, 8, 249.

UNe Redman S.L. et al. A High-Resolution Atlas of Uranium-Neon in the H Band

7.2 "Traditional" Gas Lamp

For an absolute wavelength reference most spectrometers rely on a gas lamp, e.g. based on ThAr. Such lamps have a limited number of well known spectral lines, well spaced over the entire range of the detector. Different gas lamps may be used to cover different settings of the detector. The initial linelist described above will tell us at which pixels and in which orders to expect which lines on the detector. This allows us to assign a mapping between the pixel position and the wavelength. In PyReduce we use a polynomial to interpolate between the known spectral lines. First we match the observation to the linelist using cross correlation in both order and pixel directions. Afterwards we match the observed peaks to their closest partners in the linelist to get the wavelength. Here we discard peaks that are further than a set cutoff away (usually around 100 m/s). Based on these we can fit a polynomial between the peaks. In PyReduce we recommend using a 2D polynomial, where the order number is used as the second coordinate. This works since the wavelength derivatives between orders is similar. Using this polynomial we can derive the wavelengths of all pixels in the spectrum. We also use this to identify additional peaks that exist both in the observation and the linelist, but weren't matched before. After repeating this method a few times, we arrive at a stable solution.

7.3 Frequency Comb / Fabry Perot Interferometer

Recent developments in the detector design incorporate Frequency Combs or Fabry Perot Interferometers to achieve better wavelength calibrations. On problem with the gas lamp, is that there are only a limited number of spectral lines, and they are not evenly spaced. Using a FC/FPI however solves these problems, by providing dense, evenly spaced peaks over all wavelengths. An additional feature of these is that the Frequency between peaks is constant. PyReduce can use such calibrations just using the assumption that the steps are constant in frequency space. This is done in the "freq_comb" step.

f(n) = f0 + n * fr, where f0 is the anchor frequency, and fr is the frequency step between two peaks, and n is the number of the peak.

First we identify the peaks in each order as usual. The wavelength of each peak is estimated by using the gas lamp solution calculated above. The individual peaks of the FC/FPI are great for relative differences but lack an absolute calibration. Special care is taken to number the peaks correctly, even between orders, so that we only have one anchor frequency and one frequency step, for all peaks. One correction that we particularly want to point out is that the grating equation provides us with the assumption that n * w = const for neighbouring peaks, where w is the wavelength of the peak. This can be used to correct a misidentified order, and correct their numbering, usually by 1 or 2 peak numbers.

The wavelength solution is then given by determining the best fit f0 and fr using all peaks in all orders. Those can then be used to determine the wavelength of each peak. Once we have a wavelength for each peak we can fit a polynomial

just as we did for the gas lamp, to get the wavelength for each pixel. Of course now there are a lot more peaks, so the solution is much better.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X