pymusepipe Documentation

Release 2.25.4

Eric Emsellem

Jul 24, 2023

CONTENTS

This is the documentation of **pymusepipe**.

CHAPTER

ONE

CONTENTS

1.1 Welcome

Pymusepipe is a Python package which serves as a wrapper around the main processing steps of the MUSE data reduction pipeline (Weilbacher et al. 2019). Pymusepipe includes a simple data organiser and prescriptions for the structure of the data files (but no database per se), a wrapper around the main functionalities of MUSE data reduction pipeline, accessed via EsoRex command-line recipes, to remove the instrumental signatures. Pymusepipe additionally provides a set of modules supporting the alignment, mosaicking, two-dimensional and three-dimensional convolution.

pymusepipe is also made for multi-pointing mosaics and multi-targets surveys as it will process targets automatically when provided a specific dictionary of which target and which pointings to consider.

A description of the pipeline and its usage to reduce data from the PHANGS-MUSE survey are presented in Emsellem et al. (2022)

Contact

The pymusepipe module is maintained by Eric Emsellem. contact via

---

> **Attention:** Please do not forget to cite Emsellem et al. (2022) and the MUSE data reduction pipeline paper (Weilbacher et al. 2019) if you make use of pymusepipe in your work. In particular, we suggest you add the following text (or equivalent) to the data reduction section of your work.
>
> *The dataset was reduced using recipes the MUSE data processing pipeline software (Weilbacher et al. 2019). All recipes were executed with ESOREX commands, wrapped around using the dedicated python package pymusepipe (Emsellem et al. 2022).*

1.1.1 GitHub Repository

You can access the source code of pymusepipe and its previous releases directly in its official GitHub repository https://github.com/emsellem/pymusepipe.

1.2 Install

1.2.1 Prerequisites

Pymusepipe assumes you have a working installation of ESOREX (i.e. that you have a working MUSE data reduction pipeline installation) and likwid-pin. The installation of these components is not covered here but can be found in the MUSE Pipeline User Manual.

Pymusepipe uses Python 3 and is not compatible with Python 2. It requires a number of standard python packages including:

- **numpy**
- **scipy**
- **matplotlib**
- **astropy**
- **mpdaf** a utility package to process and analyse datacubes, and more specifically MUSE cubes, images and spectra developed by the MUSE GTO-CRAL Team.

In addition some packages are needed to access specific functionality:

- **pypher** to use the convolution package of pymusepipe
- **spacepylot** to use the automatic alignment module

1.2.2 Installation

You can install this package via pypi via a simple:

```
pip install pymusepipe
```

You can obviously also install it by cloning it from github, or downloading the source (from github) and do something like:

```
python setup.py develop
```

The "develop" option is recommended as it actually does not copy the files in your system but just creates a link. In that way you can easily update the source software without reinstalling it. The link will directly use the source which has been udpated.

The other option is to use the standard "install" option:

```
python setup.py install
```

1.3 Getting Started

1.3.1 Basic Usage - Dealing with OBs, individually

The pymusepipe wrapper is meant to provide the user with a simple way to run the MUSE pipeline.

Only three steps are needed:

1. preparing the data (download),
2. preparing the configuration files (templates are provided),
3. using the code (a few lines at most).

I recommend to use `Ipython` as an environment, possibly via a `screen` or `tmux` which would allow you to disconnect from the server that actually runs the commands.

1.3.2 Step 1: Preparing your data

The first thing to do is to prepare the folder structure to welcome your MUSE datasets.

Imagine you have:

- a target or field named e.g. `NGC1000`.
- several **datasets**. In the simplest cases this corresponds to data from one MUSE Observing Block (OB), including all the calibration and science object raw data files, as downloaded from the ESO archive. In practice pymusepipe will also reduce several OBs, provided all the necessary calibrations are avaiable. Some functionality requires a distinction between **dataset** and **pointing**. This distinction is described in *Targets and Mosaicking*.

Then under your data root folder <my_data_folder> create the following folder structure:

```
<my_data_folder>/NGC1000
   ./OB001
      ./Raw
   ./OB002
     ./Raw
   ./OB003
     ./Raw
```

Each dataset, or OB for short, has a `Raw` folder.

The next step is to download your MUSE data (including raw calibrations) from the ESO web site, and put all the raw files (in fitsor fits.gz format) into each individual `Raw` folders, associated with the right dataset.

1.3.3 Step 2: Preparing your configuration files

pymusepipe only needs two configurations ascii files:

1. `calib_tables.dic`, which contains a series of file names associated with muse static calibration files and other configuration files (e.g. fitler lists). Most names are self-explantory. These include:
   - *geo_table* and *astro_table*: static files, time dependent geometry files can be specified (see `rc.dic`).
   - *badpix_table*, *vignetting_mask*, *std_flux_table*, *extinct_table*, *line_catalog*, statical calibrations provided with the MUSE pipeline, no need to change these.
   - *filter_list* : used in case you wish to provide your own. Note that the file it needs to follow the MUSE standard for such a table.

2. `rc.dic`, which provides the root folders for the static calibration files and for your datasets.

   - *root* provides the root folder for your data. For Target NGC1000, and OB 1, the Raw data will be looked for in *root*/NGC1000/OB001/Raw.
   - *musecalib* should contain the standard MUSE calibration files. These are distributed in the MUSE pipeline installation in a “muse-calib-x.x.x/cal” folder.
   - *musecalib_time*: time dependent geometry and astrometry files (the correspondence between observing run dates and specific files are hard-coded into pymusepipe).

Examples of such files are provided in the `config_templates` folder of the pymusepipe package.

1.3.4 Step 3: Running the pipeline

Here is an example of how to run the pipeline to reduce a single OB (dataset):

```
# Import the modules
import pymusepipe as pmp
from pymusepipe import musepipe

# define the paths to the two configuration files
rcfile = "my_data/MUSE/Config/rc.dic"
calfile = "my_data_MUSE/Config/calib_tables.dic"

# Initialisation of the python - MusePipe Class - structure
mypipe = musepipe.MusePipe(targetname="NGC1000", dataset=1, rc_filename=rcfile,
                        cal_filename=calfile, log_filename="NGC1000_version01.log",
                        fakemode=False, overwrite_astropy_table=True,
                        filter_list="white,Cousins_R",
                        filter_for_alignment="Cousins_R")

# Launching the pipeline
mypipe.run_recipes()
```

That’s it! Your data has now been reduced!

Some explanation may be needed to understand what is happening:

- `targetname`: is just the name of the target, used to decided where the data will be
- `dataset`: the number of the OB that will be used, namely “OB001” etc.
- `logfile`: name of the logging file.
- `fakemode`: you can set this to True if you just wish to initialise things without actually running any recipes. The pipeline will only set things up but if you run any recipes will only “fake” them (not launch any esorex command, only spitting the log out)
- `filter_list`: list of filter names to use to reconstruct images when building up cubes. This should be part of the filter_list fits table provided (see `calib_tables` config file).
- `filter_for_alignment`: specific filter name used for alignment between exposures.

Other options can be useful:

- `musemode`: this is by default `WFM_NOAO_N` which is the most often used MUSE mode. This will filter out exposures not compatible with the given mode.
- `reset_log`: will reset the log file. By default it is False, hence new runs will be appended.

- `overwrite_astropy_table`: by default this is False. If True, new runs will rewrite the Astropy output tables.
- `time_astrometry``: by default it is False, meaning the pipeline will try to detect a GEOMETRY and ASTROMETRY Files delivered with the Rawfiles by ESO. If set to True, it will use the time dependent astro/geo files provided by the GTO Team but you would need to make these available on your system.Hence I would recommend to keep the default (False).

1.3.5 Under the hood of run_recipes

`run_recipes()` launches a default set of functions listed below:

```
# generate the master bias using the muse_bias esorex recipe
mypipe.run_bias()
# generate the master flat using the muse_flat esorex recipe
mypipe.run_flat()
# generate the wavelength calibration using the muse_wavecal esorex recipe
mypipe.run_wave()
# generate the lsf using the muse_lsf esorex recipe
mypipe.run_lsf()
# generate the illumination correction using the muse_lsf esorex recipe
mypipe.run_twilight(illum=True)
# process individual exposures to remove the instrumental signature usign the muse_
↪scibasic
# esorex recipes. It runs on both the object, standard star and sky exposures
mypipe.run_scibasic_all(illum=True)
# generates the response function using the standard star observations and the muse_
↪standard
# esorex recipe
mypipe.run_standard()
# uses the sky exposures to generate a sky spectrum
mypipe.run_sky(fraction=0.8)
# runs the esosex muse_scipost recipe individually on each object exposures generating
# a datacubes and image in the requested filter for each exposure.
# These images are then used for aligment.
mypipe.run_prep_align()
# runs the muse_exp_align recipe to generate an OFFSET_TABLE files containing the␣
↪astrometric
# shifts between individual exposures. Pymusepipe provides more refined options for this
mypipe.run_align_bydataset()
# ??
mypipe.run_align_bygroup()
# generates the final aligned datacubes for individual exposures using muse_scipost
mypipe.run_scipost_perexpo()
# generates the sky datacube
mypipe.run_scipost_sky()
# merge exposures in the final datacube
mypipe.combine_dataset()
```

Individual pipeline stages can be (re)run by calling any of the individual functions above. The order is important, as in any data reduction process

Attention: This pipeline flow closely mirrors the standard data reduction for MUSE data implemented by the e.g. EsoReflex workflow. Pymusepipe offers alternative recipes to perform alignment (*Alignment*) and mosaicking (*Targets and Mosaicking*). For best results, therefore, we do not recommend running the above workflow. Examples workflows are presented in *PHANGS pipeline example*.

1.3.6 Structure of the output

Folders

The structure of the output is driven by a set of folder names described in `pymusepipe.init_musepipe()` in a few dictionaries (`dic_input_folders()`, `dic_folders()`, `dic_folders_target()`). You can in principle change the names of the folders themselves, although it is not advisable.

The pipeline will create the folder structure automatically, checking whether the folders exist or not.

Log files

Two basic log files are produced: one is the Esorex output which will be stored in the "Esorex_log" folder. The other one will be in the "Log" folder with the name provided at start: that one is like a shell script which can be used to rerun things directly via the command line. In the "Log" folder, there will also be, for each log file, a file ".out" and one with ".err" extensions, respectively including all the stdout and stderr messages. This may be useful to trace details in the data reduction and problems.

Astropy Tables

Each recipe will trigger the creation of a astropy Table. These are stored under "Astro_Tables". You can use these to monitor which files have been processed or used.

Sof files

Sof files are saved under the "Sof" directory for each esorex recipes used in the pipeline. These are useful to see exactly which files are processed by each esorex recipe.

Python structure

Most of the information you may need is actually stored in the python `pymusepipe.musepipe.MusePipe` class structure. More details to come.

1.4 Alignment

1.4.1 Limitations of run_align recipes

The esorex implementation of the alignment procedure (implemented in the `pymusepipe.prep_recipes_pipe.PipePrep.run_align_bydataset()`) for multiple object exposures suffers from some severe limitations:

- It does not perform absolute astrometry, but merely fixes the astrometry of subsequent exposures to the WCS of the first one. This is problematic for comparison of MUSE data with external datasets.

- It works by finding and matching point sources across white light images from multiple exposures. This requires the images to contain a sufficient number of point sources. Moreover, in case of mosaics, is requires the *overlap region* between different MUSE pointings to contain a sufficient number of point sources. In practice, this requirement is very limiting.

Pymusepipe provides the `pymusepipe.align_pipe()` module to overcome both these limitations.

1.5 Targets and Mosaicking

tbw

1.6 Convolution

tbw

1.7 PHANGS pipeline example

tbw

1.8 License

1.9 Contributors

- Eric Emsellem <eric.emsellem@eso.org>

1.10 Changelog

1.10.1 Version 2.9.9

- Cleaning the full version including Target Sample

1.10.2 Version 1.0

- First structure with MusePointing

1.11 pymusepipe

1.11.1 pymusepipe package

Submodules

pymusepipe.align_pipe module

MUSE-PHANGS alignement module. This module can be used to align MUSE reconstructed images either with each others or using a reference background image. It spits the results out in a Fits table which can then be used to process and mosaic Muse PIXTABLES via the MUSE ESO pipeline. It includes a normalisation factor, an estimate of the background, as well as any potential rotation. Fine tuning can be done by hand by the user, using a set of reference plots.

class pymusepipe.align_pipe.**AlignMuseDataset**(*name_reference*, *folder_reference=None*, *folder_muse_images=None*, *name_muse_images=None*, *sel_indices_images=None*, *median_window=10*, *subim_window=10*, *dynamic_range=10*, *border=10*, *hdu_ext=(0, 1)*, *chunk_size=15*, *firstguess='crosscorr'*, ***kwargs*)

Bases: `object`

Class to align MUSE images onto a reference image.

apply_extra_offset_ima(*nima=0*, *extra_pixel=None*, *extra_arcsec=None*, *extra_rotation=None*, ***kwargs*)

Shift image with index nima with the total offset after adding any extra given offset This does not return anything but could in principle if using the output of the self.shift

Input

nima: int
Index of image to consider

extra_pixel: list of 2 floats
Extra offsets (x,y) in pixels. If None, nothing is applied

extra_arcsec: list of 2 floats
Extra offsets (x,y) in arcsec if extra_pixel is not provided If None, nothing is applied

extra_rotation: float
Rotation in degrees. If None, no new extra offset is applied

`apply_optical_flow_offset_ima`(*nima=0*)

Transfer the value of the optical flow into the extra pixel

`apply_optical_flow_offset_listima`(*list_nima=None*)

Apply the optical flow offset as extra pixels offsets and rotation

Input

list_nima: list
If None, will be initiliased to the default list of indices

`compare`(*data1*, *data2*, *header=None*, *start_nfig=1*, *nlevels=10*, *levels=None*, *convolve_data1=0.0*, *convolve_data2=0.0*, *showcontours=True*, *showcuts=True*, *shownormalise=True*, *showdiff=True*, *normalise=True*, *median_filter=True*, *ncuts=5*, *percentage=5.0*, *suffix_fig=''*, ***kwargs*)

Compare the projected reference and MUSE image by plotting the contours, the difference and vertical/horizontal cuts.

Parameters

- **data1** –
- **data2** (*2d np.arrays*) – Array to compare
- **header** (*Header*) – If provided, will be use to get the WCS in the plots. Default is None (ignored).
- **polypar** (*ODR result*) – If None, it will be recalculated
- **showcontours** (*bool [True]*) –
- **showcuts** (*bool [True]*) –
- **shownormalise** (*bool [True]*) –
- **showdiff** (*bool [True]*) – All options corresponding to 1 specific plot. By default show them all (all True)
- **ncuts** (*int [5]*) – Number of vertical / horizontal cuts along the ratio between the 2 maps to be shown ("cuts")
- **percentage** (*float [5]*) – Used to compute which percentile to show
- **start_nfig** (*int [1]*) – Number of the matplotlib Figure to start with
- **nlevels** (*int [10]*) – Number of levels for the contour plots
- **levels** (*list of float [None]*) – Specific list of levels if any (default is None)

- **convolve_data1** (*float [0]*) – If not 0, will convolve with a gaussian of that sigma
- **convolve_data2** (*float [0]*) – If not 0, will convolve the reference image with a gaussian of that sigma
- **(bool)** (*savefig*) – If True, will save the figure into a png
- **suffix_fig** (*str*) – Suffix name to add to the figure filenames
- **figures** (*Makes a maximum of 4*) –

compare_ima(*nima=0*, *nima_museref=None*, *convolve_muse=0*, *convolve_reference=0.0*, ***kwargs*)

Input

nima: int
Index of input image

nima_museref: int
Index of second input image for the reference. Default is None, hence ignored and the default reference image will be used.

convolve_muse: float
Sigma of the gaussian to convolve the input images. Default is 0 (no convolution)

convolve_reference: float
Sigma of the gaussian to convolve the reference. Default is 0 (no convolution)

threshold_muse: float
Minimum value to consider in the input images

Creates

Plots which compare the two input datasets as defined by the indices

find_cross_peak(*muse_hdu*, *rotation=0.0*, *threshold=None*, ***kwargs*)
Aligns the MUSE HDU to a reference HDU

Input

muse_hdu: astropy.io.fits hdu
MUSE hdu file

name_musehdr: str
name of the muse hdr to save

rotation: float
Angle in degrees (0).

threshold: minimum flux to be used in the cross-correlation
Flux below that value will be set to 0. Default is 0.

returns

- *xpix_cross*
- **ypix_cross** (*x and y pixel coordinates of the cross-correlation peak*)

`find_cross_peak_ima`(*nima=0*, *threshold=None*)

Find the cross correlation peak and get the x and y shifts for a given image, given its index nima

Input

nima: int
Index of the image

threshold: float
Minimum flux for the cross-correlation

`find_cross_peak_listima`(*list_nima=None*, *threshold=None*)

Run the cross correlation peaks on all MUSE images Derive the self.cross_off_pixel/arcsec parameters

Input

list_nima: list
list of indices for images to process Should be a list. Default is None and all images are processed

threshold: float [None]
minimum flux to be used in the cross-correlation Flux below that value will be set to 0.

`get_imaref_muse`(*muse_hdu*, *rotation=0.0*, ***kwargs*)

Returns the ref and input images on the same grid as the given input hdu assuming a given rotation

Input

muse_hdu: HDU
MUSE hdu file

name_musehdr: str
name of the muse hdr to save

rotation: float
Angle in degrees (0).

threshold: float
Minimum flux to prepare the image (0).

returns

- **ima_ref, ima_muse** (*arrays*) – Reprojected images
- *Note that the original images are saved in self._temp_input_origmuse and*
- *self._temp_input_origref when debug mode is on (self._debug)*

`get_normfactor_ima`(*nima=0*, *median_filter=True*, *border=0*, *convolve_muse=0.0*, *convolve_reference=0.0*, *chunk_size=10*, ***kwargs*)

Get the normalisation factor for shifted and projected images. This function only consider the input image given by index nima and the reference image (after projection).

Input

nima: int
Index of image to consider

median_filter: bool
If True, will median filter

convolve_muse: float [0]
Will convolve the image with index nima with a gaussian with that sigma. 0 means no convolution

convolve_reference: float [0]
Will convolve the reference image with a gaussian with that sigma. 0 means no convolution

border: int
Number of pixels to crop

threshold: float [None]
Threshold for the input image flux to consider

chunk_size: int
Size of chunks to consider for chunk statistics (polynomial normalisation)

returns

- **data** (*2d array*)
- **refdata** (*2d array*) – The 2 arrays (input, reference) after processing

`get_shift_from_pcc`(*muse_hdu*, *rotation=0.0*, *threshold=0.0*, *verbose=False*, ***kwargs*)

Find a guess translation using PCC

Input

muse_hdu: HDU
MUSE hdu file

rotation: float
Angle in degrees (0).

threshold: float
Minimum flux to prepare the image (0).

name_musehdr: str
Name of the muse hdr to save. Optional. Only operational if self.save_hdr is True

returns

- *xpix_pcc*
- *ypix_pcc x and y pixel coordinates of the cross-correlation peak*

`get_shift_from_pcc_ima`(*nima=None*, *threshold=None*, *rotation=None*, *verbose=False*)

Run the PCC shift guess for image nima

Input

nima: int
Index of image

threshold: float [None]
minimum value to be used in the phase cross-correlation Flux below that value will be set to 0.

rotation: float
If None, will take the init_rotangle. Otherwise it will take the input value

get_shift_from_pcc_listima(*list_nima=None*, *threshold=None*, *verbose=False*)

Run the PCC shift guess on a list of images given by a list of indices

Input

list_nima: list of indices for images to process
Should be a list. Default is None and all images are processed

thhreshold: float [None]
minimum value to be used in the phase cross-correlation Flux below that value will be set to 0.

init_guess_offset(***kwargs*)

Initialise first guess, either from cross-correlation (default) or from an Offset FITS Table

Input

firstguess: str
If "crosscorr" uses cross-correlation to get the first guess of the offsets. If "fits" uses the input fits table.

init_optical_flow_hdu(*muse_hdu*, *rotation=0.0*, *threshold=None*, *guess_translation=(0.0, 0.0)*, *header=None*, *verbose=False*, ***kwargs*)

Get the optical flow for this hdu

Input

muse_hdu: HDU
Muse HDU input

rotation: float
Input rotation

threshold: float
Minimum flux to consider in the image

guess_translation: tuple of 2 floats
Guess offset in X and Y, e.g., (0., 0.)

name_musehdr: str
Name of hdr in case those are saved (self.save_hdr is True)

`init_optical_flow_ima`(*nima=0*, *threshold=None*, *guess_offset_pixel=None*, *guess_offset_arcsec=None*, *guess_rotation=None*, *force_pcc_guess=False*, *verbose=False*, *provide_header=True*, ***kwargs*)

Initialise the optical flow using the current image with index nima

Input

nima: int
Index of image

threshold: float
Minimum flux to consider

`init_optical_flow_listima`(*list_nima=None*, ***kwargs*)

Initialise the optical flow on a list of images given by a list of indices

Input

list_nima: list of indices for images to process
Should be a list. Default is None and all images are processed

`iterate_on_optical_flow_ima`(*nima=0*, *niter=5*, *verbose=False*, *use_rotation=True*, ***kwargs*)

Iterate solution using the optical flow guess

Input

nima: int
Index of image to consider

niter: int
Number of iterations

`iterate_on_optical_flow_listima`(*list_nima=None*, *use_rotation=True*, ***kwargs*)

Run the iteration for the optical flow on a list of images given by a list of indices

Input

list_nima: list of indices for images to process
Should be a list. Default is None and all images are processed

niter: int
Number of iterations. Optional. If not provided, will use the default in self.iterate_on_optical_flow_ima

`list_states`(*nstate_max=None*)

Input

`offset_and_compare_ima`(*nima=0*, *extra_pixel=None*, *extra_arcsec=None*, *extra_rotation=None*, ***kwargs*)

Run the offset and comparison for a given image number

Input

nima: int
Index of the image to consider

extra_pixel: list of 2 floats
Offsets in X and Y in pixels to add to the existing guessed offsets IMPORTANT NOTE: extra_pixel will be considered first (before extra_arcsec).

extra_arcsec: list of 2 floats
Offsets in X and Y in arcsec to add to the existing guessed offsets. Ignored if extra_pixel is given or None

extra_rotation: rotation in degrees
Angle to rotate the image (in degrees). Ignore if None

Additional arguments

threshold: float [0]
Threshold to consider when plotting the comparison

plot (bool): if True, will plot the comparison

If not used, will use the default self.plot

- flux comparison (1 to 1)
- Map of the flux ratio
- Contours of the two scaled maps
- Cuts of the division between the 2 maps

See also all arguments from self.compare

`open_hdu`()

Open the HDU of the MUSE and reference images

`open_offset_table`(*name_table=None*)

Read offset table from fits file

Input

name_table: str
Name of the input OFFSET table

returns

- **status** (*None if no table name is given, False if file does not*) – exist, True if it does
- **Table** (*the result of a astropy.Table.read of the fits table*)

`print_images_names()`
Print out the names of the images being considered for alignment

`print_offsets_and_norms`(*filename='_temp.txt'*, *folder_output_file=None*, *overwrite=True*)
Save all offsets and norms into filename. By default, file will be overwritten.

Input

filename: str
Name of file where the output will be written

folder_output_file: str
Name of output folder where the file will be written

overwrite: bool
Default is True

Creates

Ascii file named via the filename input argument

`retrieve_state`(*nstate=1*)
Retrieve the state with offset and background and norm factors

Input

nstate = int default=1

`run_optical_flow`(*list_nima=None*, *save_plot=True*, *use_rotation=True*, *verbose=False*, ***kwargs*)
Run Optical flow, first with a guess offset and then iterating. The solution is saved as extra offset in the class, and a op_plot instance is created. If save_plot is True, it will save a set of default plots

Input

list_nima: list
List of indices. If None, will use the default list of all images

save_plot
[bool] Whether to save the optical flow diagnostic plots or not.

use_rotation: bool
True if you wish to have rotation. False otherwise

verbose: bool

`run_optical_flow_ima`(*nima=0*, *save_plot=True*, *use_rotation=True*, *verbose=False*, ***kwargs*)

Run Optical flow on image with index nima, first with a guess offset and then iterating. The solution is saved as extra offset in the class, and a op_plot instance is created. If save_plot is True, it will save a set of default plots

Input

nima: int
Image index.

save_plot
[bool] Whether to save the optical flow diagnostic plots or not.

`save_fits_offset_table`(*name_output_table=None*, *folder_output_table=None*, *overwrite=False*, *suffix=''*, *save_flux_scale=True*, *save_other_params=True*)

Save the Offsets into a fits Table

Input

folder_output_table: str [None]
Folder of the output table. If None (default) the folder for the input offset table will be used or alternatively the folder of the MUSE images.

name_output_table: str [None]
Name of the output fits table. If None (default) it will use the one given in self.name_output_table

overwrite: bool [False]
If True, overwrite if the file exists

suffix: str [“”]
Suffix to be used to add to the input name. This is handy to just modify the given fits name with a suffix (e.g., version number).

save_flux_scale: bool [True]
If True, saving the flux in FLUX_SCALE If False, do not save the flux conversion

save_other_params: bool [True]
If True, saving the background + rotation If False, do not save these 2 parameters.

Creates

A fits table with the given name (using the suffix if any)

save_image(*newfits_name=None*, *nima=0*)

Save the newly determined hdu

Input

newfits_name: str
Name of the fits file to be used

nima: int [0]
Index of the image to save

Creates

A new fits file

save_polypar_ima(*nima=0*, *beta=None*)

Saving the input values into the fixed arrays for the polynomial

Input

beta: list/array of 2 floats

save_state(*force_nstate=None*)

Save the offset and background and normalisation to a given attribute defined by self._nstate

Input

force_nstate: int default=None

show_linearfit_values()

Print some information about the linearly fitted parameters pertaining to the normalisation.

show_norm_factors()

Print some information about the normalisation factors.

show_offset_fromfits(*name_table=None*)

Print offset table from fits file

Input

name_table: str
Name of the input OFFSET table

`show_offsets()`

Print out the offset from the Alignment class

`transfer_extra_to_guess`(*transfer_rotation=True*)

Transfer the values of the extra offset as a guess

`class pymusepipe.align_pipe.OffsetState`(*nstate=1*, *info=None*)

Bases: `object`

A very simple class used to store the offsets

`pymusepipe.align_pipe.align_hdu`(*hdu_target=None*, *hdu_to_align=None*, *target_rotation=0.0*, *to_align_rotation=0.0*, *conversion_factor=1.0*, *use_mpdaf=False*)

Project the reference image onto the MUSE dataset Hidden function, as only used internally

Input

hdu_target: HDU [None]
Target hdu (on to which to project)

hdu_to_align: HDU [None]
Hdu to be aligned

target_rotation: float [0]
Rotation angle in degrees of the target hdu

to_align_rotation: float [0]
Rotation angle in degrees of the to be aligned hdu

conversion_factor: float
Factor to be applied to the to_align hdu

use_mpdaf: bool
If True, use mpdaf to project. This is not recommended. If False, use reproject. This is the recommended option (default)

returns
hdu_repr – Reprojected HDU. None if nothing is provided

rtype
HDU

`pymusepipe.align_pipe.arcsec_to_pixel`(*hdu*, *xy_arcsec=(0.0, 0.0)*)

Transform from arcsec to pixel for the muse image using the hdu to extract the WCS, hence the scaling.

Input

hdu: astropy hdu (fits)
Input hdu which includes a WCS

xy_arcsec: list of 2 floats ([0,0])
Coordinates to transform from arcsec to pixel.

returns

- **xpix, ypix** (*tuple or list of 2 floats*) – Pixel coordinates
- **See also** (*pixel_to_arcsec (align_pipe.py)*)

pymusepipe.align_pipe.**get_conversion_factor**(*input_unit, output_unit, filter_name='WFI', dict_equiv={'DUPONT_R': [(Unit("erg / (Angstrom cm2 s)"), Unit("erg / (cm2 Hz s)"), <function spectral_density.<locals>.f_la_to_f_nu>, <function spectral_density.<locals>.f_la_from_f_nu>), (Unit("erg / (cm2 Hz s)"), Unit("erg / (cm2 s)"), <function spectral_density.<locals>.f_nu_to_nu_f_nu>, <function spectral_density.<locals>.f_nu_from_nu_f_nu>), (Unit("erg / (Angstrom cm2 s)"), Unit("erg / (cm2 s)"), <function spectral_density.<locals>.f_la_to_la_f_la>, <function spectral_density.<locals>.f_la_from_la_f_la>), (Unit("ph / (Angstrom cm2 s)"), Unit("erg / (Angstrom cm2 s)"), <function spectral_density.<locals>.phot_f_la_to_f_la>, <function spectral_density.<locals>.phot_f_la_from_f_la>), (Unit("ph / (Angstrom cm2 s)"), Unit("erg / (cm2 Hz s)"), <function spectral_density.<locals>.phot_f_la_to_f_nu>, <function spectral_density.<locals>.phot_f_la_from_f_nu>), (Unit("ph / (Angstrom cm2 s)"), Unit("ph / (cm2 Hz s)"), <function spectral_density.<locals>.phot_f_la_to_phot_f_nu>, <function spectral_density.<locals>.phot_f_la_from_phot_f_nu>), (Unit("ph / (cm2 Hz s)"), Unit("erg / (cm2 Hz s)"), <function spectral_density.<locals>.phot_f_la_to_f_la>, <function spectral_density.<locals>.phot_f_la_from_f_la>), (Unit("ph / (cm2 Hz s)"), Unit("erg / (Angstrom cm2 s)"), <function spectral_density.<locals>.phot_f_nu_to_f_la>, <function spectral_density.<locals>.phot_f_nu_from_f_la>), (Unit("ph / (cm2 s)"), Unit("erg / (cm2 s)"), <function spectral_density.<locals>.phot_f_la_to_f_la>, <function spectral_density.<locals>.phot_f_la_from_f_la>), (Unit("erg / (Angstrom s)"), Unit("erg / (Hz s)"), <function spectral_density.<locals>.f_la_to_f_nu>, <function spectral_density.<locals>.f_la_from_f_nu>), (Unit("erg / (Hz s)"), Unit("erg / s"), <function spectral_density.<locals>.f_nu_to_nu_f_nu>, <function spectral_density.<locals>.f_nu_from_nu_f_nu>), (Unit("erg / (Angstrom s)"), Unit("erg / s"), <function spectral_density.<locals>.f_la_to_la_f_la>, <function spectral_density.<locals>.f_la_from_la_f_la>), (Unit("ph / (Angstrom s)"), Unit("erg / (Angstrom s)"), <function spectral_density.<locals>.phot_f_la_to_f_la>, <function spectral_density.<locals>.phot_f_la_from_f_la>), (Unit("ph / (Angstrom s)"), Unit("erg / (Hz s)"), <function spectral_density.<locals>.phot_f_la_to_f_nu>, <function spectral_density.<locals>.phot_f_la_from_f_nu>), (Unit("ph / (Angstrom s)"), Unit("ph"), <function spectral_density.<locals>.phot_f_la_to_phot_f_nu>, <function spectral_density.<locals>.phot_f_la_from_phot_f_nu>), (Unit("ph"), Unit("erg / (Hz s)"), <function spectral_density.<locals>.phot_f_la_to_f_la>, <function spectral_density.<locals>.phot_f_la_from_f_la>), (Unit("ph"), Unit("erg / (Angstrom s)"), <function spectral_density.<locals>.phot_f_nu_to_f_la>, <function spectral_density.<locals>.phot_f_nu_from_f_la>),* Conversion of units from an input one to an output one

Input

input_unit: astropy unit
Input astropy unit to analyse

output_unit: astropy unit
Astropy unit to compare to input unit.

filter_name: str, optional
Name of the filter

dict_equiv: dict, optional
Dictionary listing the equivalencies for various filters. Will be using it if the filter name is the key list to help with the conversion.

returns
conversion

rtype
astropy unit conversion

`pymusepipe.align_pipe.`**`init_plot_optical_flow`**`(`*opflow*`)`

Initialise the optical flow plot using the AlignmentPlotting

Input

opflow: optical flow instance (see spacepylot)

rtype
An optical flow plot instance

`pymusepipe.align_pipe.`**`is_sequence`**`(`*arg*`)`

Test if sequence and return the boolean result

Parameters
arg (*input argument*) –

Returns
result

Return type
boolean

`pymusepipe.align_pipe.`**`pixel_to_arcsec`**`(`*hdu, xy_pixel=(0.0, 0.0)*`)`

Transform from arcsec to pixel for the muse image using the hdu to extract the WCS, hence the scaling.

Input

hdu: astropy hdu (fits)
Input hdu which includes a WCS

xy_pixel: tuple or list of 2 floats ((0,0))
Coordinates to transform from pixel to arcsec

returns

- **xarc, yarc** (*2 floats*) – Arcseconds coordinates
- **See also** (*arcsec_to_pixel (align_pipe.py)*)

`pymusepipe.align_pipe.rotate_pixtable(folder='', name_suffix='', nifu=1, angle=0.0, **kwargs)`

Rotate a single IFU PIXTABLE_OBJECT Will thus update the HIERARCH ESO INS DROT POSANG keyword.

Input

folder: str
name of the folder where the PIXTABLE are

name_suffix: str
name of the suffix to be used on top of PIXTABLE_OBJECT

nifu: int
Pixtable number. Default is 1

angle: float
Angle to rotate (in degrees)

`pymusepipe.align_pipe.rotate_pixtables(folder='', name_suffix='', list_ifu=None, angle=0.0, **kwargs)`

Will update the derotator angle in each of the 24 pixtables Using a loop on rotate_pixtable

Will thus update the HIERARCH ESO INS DROT POSANG keyword.

Input

folder: str
name of the folder where the PIXTABLE are

name_suffix: str
name of the suffix to be used on top of PIXTABLE_OBJECT

list_ifu: list[int]
List of Pixtable numbers. If None, will do all 24

angle: float
Angle to rotate (in degrees)

pymusepipe.check_pipe module

MUSE-PHANGS check pipeline module

class pymusepipe.check_pipe.**CheckPipe**(*mycube='DATACUBE_FINAL.fits'*, *pdf_name='check_pipe.pdf'*, *pipe=None*, *standard_set=True*, ***kwargs*)

Bases: *MusePipe*

Checking the outcome of the data reduction

check_given_images(*suffix=None*)

Check all images with given suffix

check_master_bias_flat()

Checking the Master bias and Master flat

check_quadrants()

Checking spectra from the 4 quadrants

check_sky_spectra(*suffix*)

Check all sky spectra from the exposures

check_white_line_images(*line='Ha'*, *velocity=0.0*)

Building the White and Ha images and Adding them on the page

pymusepipe.check_pipe.**print_plot**(*text*)

pymusepipe.combine module

MUSE-PHANGS combine module

class pymusepipe.combine.**MusePointings**(*targetname=None*, *list_datasets=None*, *list_pointings=None*, *pointing_table=None*, *pointing_table_format='ascii'*, *pointing_table_folder=''*, *folder_config=''*, *rc_filename=None*, *cal_filename=None*, *suffix=''*, *name_offset_table=None*, *folder_offset_table=None*, *log_filename='MusePipeCombine.log'*, *verbose=True*, *debug=False*, ***kwargs*)

Bases: *SofPipe*, *PipeRecipes*

Class for a set of MUSE Pointings which can be covering several datasets. This provides a set of rules and methods to access the data and process them.

assign_pointing_table(*input_table=None*, *folder=''*, *table_format='ascii'*)

Assign the pointing table as provided. If not provided it will create one from the pixtable list

Input

input_table: str, QTable, Table or PointingTable

Create pointing_table attribute.

create_all_pointings_wcs(*filter_list='white'*, *list_pointings=None*, ***kwargs*)

Create all pointing masks one by one as well as the wcs for each individual pointings. Using the grid from the global WCS of the mosaic but restricting it to the range of non-NaN. Hence this needs a global WCS mosaic as a reference to work.

Input

filter_list = list of str
List of filter names to be used.

`create_combined_wcs`(*refcube_name=None*, *lambdaminmax_wcs=[6800, 6805]*, ***kwargs*)

Create the reference WCS from the full mosaic with a given range of lambda.

Input

refcube_name: str
Name of the cube. Can be None, and then the final datacube from the combine folder will be used.

wave1: float - optional
Wavelength taken for the extraction. Should only be present in all spaxels you wish to get.

prefix_wcs: str - optional
Prefix to be added to the name of the input cube. By default, will use "refwcs".

add_targetname: bool [True]
Add the name of the target to the name of the output WCS reference cube. Default is True.

`create_pointing_wcs`(*pointing*, *lambdaminmax_mosaic=[4700, 9400]*, *filter_list='white'*, ***kwargs*)

Create the mask of a given pointing And also a WCS file which can then be used to compute individual pointings with a fixed WCS.

Input

pointing: int
Number of the pointing

lambdaminmax_mosaic: array of 2 floats
Default is lambdaminmax_for_mosaic, the starting end ending wavelengths needed for a mosaic.

filter_list = list of str
List of filter names to be used.

Creates:
pointing mask WCS cube

returns
Name of the created WCS cube

`create_reference_wcs`(*pointings_wcs=True*, *mosaic_wcs=True*, *wcs_refcube_name=None*, *refcube_name=None*, *folder_refcube=''*, *list_pointings=None*, ***kwargs*)

Create the WCS reference files, for all individual pointings and for the mosaic.

Input

pointings_wcs: bool [True]
Will run the individual pointings WCS

mosaic_wcs: bool [True]
Will run the combined WCS

wcs_refcube_name: str default=None, optional
Name of the input WCS to be used. If None (default), we will look at the refcube_name keyword. If set to 'auto', we will used the default naming conventions to find it on disk. If set to a bona fide name, it will be used as reference WCS.

refcube_name: str default=None, optional
Name of the input cube to guide the WCS building (only used if wcs_refcube_name is None). If None, a run_combine will ensure we have a good reference cube that can be used for the building of a reference WCS. If provided, it will be used as the reference cube to then build the reference WCS.

folder_refcube: str, optional
Folder name for the reference cube or wcs.

list_pointings: list of int default=None, optional
List of pointings to consider

****kwargs: additional keywords including**
lambdaminmax: [float, float]

`property dict_pixtabs_in_datasets`

`property dict_pixtabs_in_pointings`

`property dict_tplexpo_per_dataset`

`property dict_tplexpo_per_pointing`

`extract_combined_narrow_wcs`(*name_cube=None*, ***kwargs*)

Create the reference WCS from the full mosaic with only 2 lambdas

Input

name_cube: str
Name of the cube. Can be None, and then the final datacube from the combine folder will be used.

wave1: float - optional
Wavelength taken for the extraction. Should only be present in all spaxels you wish to get.

prefix_wcs: str - optional
Prefix to be added to the name of the input cube. By default, will use "refwcs".

add_targetname: bool default=True
Add the name of the target to the name of the output WCS reference cube. Default is True.

Creates:
Combined narrow band WCS cube

returns
name of the created cube

filter_pixables_with_list(*list_datasets=None*, *list_pointings=None*)

Filter a list of pixtables

Parameters

- **list_datasets** (*list of int, optional*) –
- **list_pointings** (*list of int, optional*) –
- **pointings** (*Filter out the pointing table using those datasets and*) –

property full_list_datasets

get_all_pixtables()

List all pixtables in the data folder Fill in the dict_allpixtabs_in_datasets dictionary and creates the pointing table

get_pointing_table(***kwargs*)

Create the pointing table from the all pixtables list

Add the pointing_table attribute and do the selection according to list_datasets and list_pointings

get_qtable(***kwargs*)

Create the qtable from the all pixtables list

goto_folder(*newpath*, *addtolog=False*)

Changing directory and keeping memory of the old working one

Input

newpath: str
Nanme of the folder where to go.

addtolog: bool, optional
Add this change of folder to the log file.

goto_origfolder(*addtolog=False*)

Go back to original folder

Input

addtolog: bool, optional
Add this change of folder to the log file.

goto_prevfolder(*addtolog=False*)

Go back to previous folder

Input

addtolog: bool, optional
Add this change of folder to the log file.

run_combine(*sof_filename='pointings_combine'*, *lambdaminmax=(4000.0, 10000.0)*, *list_pointings=None*, *suffix=''*, ***kwargs*)

MUSE Exp_combine treatment of the reduced pixtables Will run the esorex muse_exp_combine routine

Parameters

- **sof_filename** (*string (without the file extension)*) – Name of the SOF file which will contain the Bias frames
- **lambdaminmax** (*list of 2 floats*) – Minimum and maximum lambda values to consider for the combine
- **suffix** (*str*) – Suffix to be used for the output name

run_combine_all_single_pointings(*add_suffix=''*, *sof_filename='pointings_combine'*, *list_pointings=None*, ***kwargs*)

Run for all pointings individually, provided in the list of pointings, by just looping over the pointings.

Input

list_pointings: list of int
By default to None (using the default self.list_pointings). Otherwise a list of pointings you wish to conduct a combine but for each individual pointing.

add_suffix: str
Additional suffix. 'PXX' where XX is the pointing number will be automatically added to that add_suffix for each individual pointing.

sof_filename: str
Name (suffix only) of the sof file for this combine. By default, it is set to 'pointings_combine'.

lambdaminmax: list of 2 floats [in Angstroems]
Minimum and maximum lambda values to consider for the combine. Default is 4000 and 10000 for the lower and upper limits, resp.

run_combine_single_pointing(*pointing*, *add_suffix=''*, *sof_filename='pointing_combine'*, ***kwargs*)

Running the combine routine on just one single pointing

Input

pointing: int
Pointing number. No default: must be provided.

add_suffix: str
Additional suffix. 'PXX' where XX is the pointing number will be automatically added to that add_suffix.

sof_filename: str
Name (suffix only) of the sof file for this combine. By default, it is set to 'pointings_combine'.

lambdaminmax: list of 2 floats [in Angstroems]
Minimum and maximum lambda values to consider for the combine. Default is 4000 and 10000 for the lower and upper limits, resp.

wcs_from_pointing: bool
True by default, meaning that the WCS of the pointings will be used. If not there, will ignore it.

set_fullpath_names()

Create full path names to be used That includes: root, data, target, but also _dict_paths, paths

pymusepipe.config_pipe module

MUSE-PHANGS configuration module

pymusepipe.config_pipe.**get_suffix_product**(*expotype*)

pymusepipe.create_sof module

MUSE-PHANGS creating sof file module

class pymusepipe.create_sof.**SofDict**

Bases: OrderedDict

New Dictionary for the SOF writing Inheriting from ordered Dictionary

class pymusepipe.create_sof.**SofPipe**

Bases: object

SofPipe class containing all the SOF writing modules

write_sof(*sof_filename*, *new=False*, *verbose=None*)

Feeding an sof file with input filenames from a dictionary

pymusepipe.cube_convolve module

MUSE-PHANGS convolve module

pymusepipe.cube_convolve.**convolution_kernel**(*input_psf*, *target_psf*, *scale=0.2*)

Create the 3D convolution kernel starting from a 3D model of the original PSF and a 2D model of the target PSF using pypher.

Parameters

input_psf (np.ndarray): 3D array with the model of the original PSF target_psf (np.ndarray): 2D array with a model of the target PSF scale (float): spatial scale of both PSF in arcsec/pix

Returns

conv_kernel (np.ndarray): 3D array with a convolution kernel
that varies as a function of wavelength.

pymusepipe.cube_convolve.**convolution_kernel_gaussian**(*fwhm_wave*, *target_fwhm*, *target_psf*, *scale=0.2*)

Create the 3D convolution kernel starting from a 3D model of the original PSF and a 2D model of the target PSF using both gaussian functions.

Parameters

- **fwhm_wave** (*array*) – FWHM of the original PSF as a function of wavelength
- **target_fwhm** (*float*) – fwhm of the target PSF
- **target_psf** (*np.ndarray*) – target psf2d
- **scale** (*float*) – spatial scale of both PSF in arcsec/pix

Returns

np.ndarray
3D array with a convolution kernel that varies as a function of wavelength.

Return type
conv_kernel

`pymusepipe.cube_convolve.cube_convolve`(*data*, *kernel*, *variance=None*, *fft=True*, *fill_value=nan*)

Convolve a 3D datacube

Parameters

- **datacube** –
- **kernel** –

Returns
the convolved 3D data and its variance

`pymusepipe.cube_convolve.cube_kernel`(*shape*, *wave*, *input_fwhm*, *target_fwhm*, *input_function*, *target_function*, *lambda0=6483.58*, *input_nmoffat=None*, *target_nmoffat=None*, *b=-3e-05*, *scale=0.2*, *compute_kernel='pypher'*)

Main function to create the convolution kernel for the datacube

Parameters

- **shape** (*array*) – the shape of the datacube that is going to be convolved. It must be in the form (z, y, x).
- **wave** (*float array*) – wavelengths for the datacube
- **target_fwhm** (*float*) – fwhm of the target PSF.
- **input_fwhm** (*float*) – fwhm of the original PSF at the reference wavelength lambda0
- **input_function** (*str*) – function to be used to describe the input PSF
- **target_function** (*str*) – function to be used to describe the target PSF
- **lambda0** – float, optional the wavelength at which the original_fwhm has been measured. Default: 6483.58 (central wavelenght of WFI_BB filter)
- **input_nmoffat** (*float*) – power index of the original PSF if Moffat [None]
- **target_nmoffat** (*float*) – power index for the target PSF if Moffat [None]
- **b** (*float*) – steepness of the fwhm vs wavelength relation. Default: -3e-5
- **step** (*float*) – wavelength dispersion in AA/px
- **scale** (*float*) – spatial pixel scale of the PSFs in arcsec/pix
- **compute_kernel** (*str*) – method to compute the convolution kernel. It can be 'pypher' or 'gaussian'

Returns

np.ndarray
3D array to be used in the convolution

Return type
Kernel

`pymusepipe.cube_convolve.`**`gaussian_kernel`**(*fwhm*, *size*, *scale=0.2*, ***kwargs*)

Gaussian kernel. Input:

fwhm (float): fwhm of the Gaussian kernel, in arcsec. size (int, ndarray): size of the requested kernel along each axis.

If ``size” is a scalar number the final kernel will be a square of side ``size”. If ``size” has two element they must be in (y_size, x_size) order. In each case size must be an integer number of pixels.

scale (float): pixel scale of the image **kwargs: is there to absorb any additional parameter which could be

provided (but won’t be used).

`pymusepipe.cube_convolve.`**`moffat_kernel`**(*fwhm*, *size*, *n=1.0*, *scale=0.2*)

Moffat kernel. Returns a Moffat function array according to given input parameters. Using astropy Moffat2DKernel.

Parameters

- **fwhm** (*float*) – fwhm of the Moffat kernel, in arcsec.
- **n** (*float*) – power index of the Moffat
- **size** (*int numpy array*) – size of the requested kernel along each axis. If ``size” is a scalar number the final kernel will be a square of side ``size”. If ``size” has two element they must be in (y_size, x_size) order. In each case size must be an integer number of pixels.
- **scale** (*float*) – pixel scale of the image [optional]

`pymusepipe.cube_convolve.`**`psf2d`**(*size*, *fwhm*, *function='gaussian'*, *nmoffat=None*, *scale=0.2*)

Create a model of the target PSF of the convolution. The target PSF does not vary as a function of wavelenght, therefore the output is a 2D array.

Parameters

size: int, array-like
the size of the final array. If ``size” is a scalar number the kernel will be a square of side ``size”. If ``size” has two elements they must be in (y_size, x_size) order.

fwhm: float
the FWHM of the psf

function: str, optional
the function to model the target PSF. Only ‘gaussian’ or ‘moffat’ are accepted. Default: ‘gaussian’

nmoffat (float): Moffat power index. It must be defined if function = ‘moffat’.

Default: None

scale: float, optional
the spatial scale of the final kernel

Returns

target: np.ndarray
a 2D array with the model of the target PSF.

`pymusepipe.cube_convolve.psf3d`(*wave*, *size*, *fwhm0*, *lambda0=6483.58*, *b=-3e-05*, *scale=0.2*, *nmoffat=None*, *function='moffat'*)

Function to create the cube with the lambda dependent PSF, following a given slope and nominal wavelength.

Parameters

wave: np.ndarray
array with the wavelength axis of the datacube

size: int, array-like
the size of the 2D PSF. If ``size” is a scalar number the 2D PSF kernel will be a square of side ``size”. If ``size” has two element they must be in (y_size, x_size) order.

fwhm0: float
the fwhm at the reference wavelength in arcseconds.

n: float
Power index of the Moffat profile. It is usually 2.8 for NOAO cubes and 2.3 for AO cubes.

lambda0: float
reference wavelength at which fwhm0 is measured. Default: 6483.58. (It’s the average wavelength for the WFI_BB filter)

b: float, optional
the steepness of the relation between wavelength and FWHM. Default: -3e-5 (arcsec/A) (From MUSE team)

scale: float, optional
spatial scale of the new datacube in arcsec. Default: 0.2 (MUSE spatial resolution).

function (str): ‘moffat’ or ‘gaussian’

Returns

psf_cube: np.array
Datacube containing MUSE PSF as a function of wavelength.

`pymusepipe.cube_convolve.pypher_script`(*psf_source*, *psf_target*, *pixscale_source=0.2*, *pixscale_target=0.2*, *angle_source=0.0*, *angle_target=0.0*, *reg_fact=0.0001*, *verbose=False*)

calculate the convolution kernel to move from one PSF to a target one. This is an adaptation of the main pypher script that it is meant to be used from the terminal.

Parameters

- **psf_source** (*ndarray*) – 2D PSF of the source image.
- **psf_target** (*ndarray*) – target 2D PSF
- **pixscale_source** (*float*) – pixel scale of the source PSF [0.2]
- **pixscale_target** (*float*) – pixel scale of the target PSF [0.2]
- **angle_source** (*float*) – position angle of the source PSF. [0]
- **angle_target** (*float*) – position angle of the target PSF. [0]
- **reg_fact** (*float*) – Regularisation parameter for the Wiener filter [1.e-4]
- **verbose** (*bool*) – If True it prints more info on screen [False]

Returns

a 2D kernel that convolved with the source PSF
returns the target PSF

Return type
kernel

pymusepipe.emission_lines module

Utility files and functions for wavelengths

`pymusepipe.emission_lines.`**`doppler_shift`**(*wavelength*, *velocity=0.0*)

Return the redshifted wavelength

`pymusepipe.emission_lines.`**`get_emissionline_band`**(*line='Ha'*, *velocity=0.0*, *redshift=None*, *medium='air'*, *lambda_window=10.0*)

Get the wavelengths of an emission line, including a correction for the redshift (or velocity) and a lambda_window around that line (in Angstroems)

Parameters

- **line** (*name of the line (string). Default is 'Ha'*) –
- **velocity** (*shift in velocity (km/s)*) –
- **medium** (*'air' or 'vacuum'*) –
- **lambda_window** (*lambda_window in Angstroem*) –

`pymusepipe.emission_lines.`**`get_emissionline_wavelength`**(*line='Ha'*, *velocity=0.0*, *redshift=None*, *medium='air'*)

Get the wavelength of an emission line, including a correction for the redshift (or velocity)

`pymusepipe.emission_lines.`**`print_emission_lines`**()

Printing the names of the various emission lines

pymusepipe.graph_pipe module

MUSE-PHANGS plotting routines

class `pymusepipe.graph_pipe.`**`GraphMuse`**(*pdf_name='drs_check.pdf'*, *figsize=(10, 14)*, *rect_layout=[0, 0.03, 1, 0.95]*, *verbose=True*)

Bases: `object`

Graphic output to check MUSE data reduction products

`close`()

`plot_page`(*list_data*)

Plot a set of blocks, each made of a set of spectra or images. This is for 1 page It first counts the number of lines needed according to the separation for images (default is 2 per line, each image taking 2 lines) and spectra (1 spectrum per line over 2 columns)

`plot_set_images`(*set_of_images=None*)

Plotting a set of images Skipping the ones that are 'None'

`plot_set_spectra`(*set_of_spectra=None*)

Plotting a set of spectra Skipping the ones that are 'None' `savepage()`

`start_page()`

Start the page

`pymusepipe.graph_pipe.open_new_wcs_figure(nfig, mywcs=None)`

Open a new figure (with number nfig) with given wcs. If not WCS is provided, just opens a subplot in that figure.

Input

nfig
[int] Figure number to consider

mywcs
[astropy.wcs.WCS] Input WCS to open a new figure (Default value = None)

returns
Figure itself with the subplots using the wcs projection

rtype
fig, subplot

`pymusepipe.graph_pipe.plot_compare_contours(data1, data2, plotwcs=None, labels=('Data1', 'Data2'), levels=None, nlevels=10, fignum=1, namefig='dummy_contours.png', figfolder='', savefig=False, **kwargs)`

Creates a plot with the contours of two input datasets for comparison

Input

data1 data2: 2d np.arrays

Input arrays to compare

plotwcs: WCS
WCS used to set the plot if provided

labels: tuple/list of 2 str
Labels for the plot

levels: list of floats
Levels to be used for the contours. Calculated if None.

fignum: int
Number for the figure

namefig: str
Name of the figure to be saved (if savefig is True)

figfolder: str
Name of the folder for the figure

savefig: bool
If True, will save the figure as namefig

Creates

Plot with contours of the two input dataset

pymusepipe.graph_pipe.**plot_compare_cuts**(*data1*, *data2*, *labels=('X', 'Y')*, *figfolder=''*, *fignum=1*, *namefig='dummy_polypar.png'*, *ncuts=11*, *savefig=False*, ***kwargs*)

Input

data1 data2 label1 label2 figfolder fignum namefig savefig kwargs

Creates

Plot with a comparison of the two data arrays using regular X and Y cuts

pymusepipe.graph_pipe.**plot_compare_diff**(*data1*, *data2*, *plotwcs=None*, *figfolder=''*, *percentage=5*, *fignum=1*, *namefig='dummy_diff.ong'*, *savefig=False*, ***kwargs*)

Parameters

- **data1** –
- **data2** –
- **figfolder** –
- **fignum** –
- **namefig** –
- **savefig** –
- **kwargs** –

pymusepipe.graph_pipe.**plot_polypar**(*polypar*, *labels=('Data 1', 'Data 2')*, *figfolder=''*, *fignum=1*, *namefig='dummy_polypar.png'*, *savefig=False*, ***kwargs*)

Creating a plot showing the normalisation arising from a polypar object

Parameters

- **polypar** –
- **label1** –
- **label2** –
- **foldfig** –
- **namefig** –

pymusepipe.graph_pipe.**print_fig**(*text*)

pymusepipe.init_musepipe module

MUSE-PHANGS pipeline wrapper initialisation of folders

class pymusepipe.init_musepipe.**InitMuseParameters**(*folder_config='Config/'*, *rc_filename=None*, *cal_filename=None*, *verbose=True*)

Bases: `object`

init_default_param(*dict_param*, *subattr=None*)

Initialise the parameters as defined in the input dictionary Hardcoded in config_pipe.py

Input

dict_param: dict
Input dictionary defining the attributes

subattr: str
Use subattr to add attributes under self.subattr

read_param_file(*filename*, *dict_param*)

Reading an input parameter initialisation file

class pymusepipe.init_musepipe.**PipeObject**(*info=None*)

Bases: `object`

A very simple class used to store astropy tables.

pymusepipe.init_musepipe.**add_suffix_tokeys**(*dic*, *suffix='_folder'*)

pymusepipe.mpdaf_pipe module

MUSE-PHANGS mpdaf-functions module

class pymusepipe.mpdaf_pipe.**BasicFile**(*filename*, ***kwargs*)

Bases: `object`

Basic file with just the name and some properties to attach to that Cube

class pymusepipe.mpdaf_pipe.**BasicPSF**(*function='gaussian'*, *fwhm0=0.0*, *nmoffat=2.8*, *b=0.0*, *l0=6483.58*, *psf_array=None*)

Bases: `object`

Basic PSF function and parameters

property psf_array

class pymusepipe.mpdaf_pipe.**MuseCube**(*source=None*, *verbose=False*, ***kwargs*)

Bases: `Cube`

Wrapper around the mpdaf Cube functionalities

astropy_convolve(*other*, *fft=True*, *inplace=False*)

Convolve a DataArray with an array of the same number of dimensions using a specified convolution function.

Copy of _convolve for a cube, but doing it per slice or not

Masked values in self.data and self.var are replaced with zeros before the convolution is performed. However masked pixels in the input data remain masked in the output.

Any variances in self.var are propagated correctly.

If self.var exists, the variances are propagated using the equation:

```
result.var = self.var (*) other**2
```

where (*) indicates convolution. This equation can be derived by applying the usual rules of error-propagation to the discrete convolution equation.

Uses `astropy.convolution.convolve_fft' or 'astropy.convolution.convolve'

Parameters

- **fft** (*boolean*) – The convolution function to use, chosen from:
  - `astropy.convolution.convolve_fft'
  - `astropy.convolution.convolve'

  In general convolve_fft() is faster than convolve() except when other.data only contains a few pixels. However convolve_fft uses a lot more memory than convolve(), so convolve() is sometimes the only reasonable choice. In particular, convolve_fft allocates two arrays whose dimensions are the sum of self.shape and other.shape, rounded up to a power of two. These arrays can be impractically large for some input data-sets.
- **other** (*DataArray or numpy.ndarray*) – The array with which to convolve the contents of self. This must have the same number of dimensions as self, but it can have fewer elements. When this array contains a symmetric filtering function, the center of the function should be placed at the center of pixel, `(other.shape - 1)//2`.

  Note that passing a DataArray object is equivalent to just passing its DataArray.data member. If it has any variances, these are ignored.
- **inplace** (*bool*) – If False (the default), return a new object containing the convolved array. If True, record the convolved array in self and return self.

Return type

~mpdaf.obj.DataArray

build_filterlist_images(*filter_list*, *prefix='IMAGE_FOV'*, *suffix=''*, *folder=None*, ***kwargs*)

Parameters

- **filter_list** –
- **prefix** –
- **suffix** –
- **folder** –
- ****kwargs** –

Returns:

convolve_cube_to_psf(*target_fwhm*, *target_nmoffat=None*, *target_function='gaussian'*, *outcube_folder=None*, *outcube_name=None*, *factor_fwhm=3*, *fft=True*, *erode_edges=True*, *npixels_erosion=2*)

Convolve the cube for a target function 'gaussian' or 'moffat'

Parameters

- **target_fwhm** (*float*) – target FWHM in arcsecond
- **target_nmoffat** – target n if Moffat function
- **target_function** (*str*) – 'gaussian' or 'moffat' ['gaussian']
- **factor_fwhm** (*float*) – number of FWHM for size of Kernel
- **fft** (*bool*) – use FFT to convolve or not [False]
- **perslice** (*bool*) – doing it per slice, or not [True] If doing it per slice, using a direct astropy fft. If doing it with the cube, it uses much more memory but is more efficient as the convolution is done via mpdaf directly.

Creates:
Folder and convolved cube names

create_reference_cube(*lambdamin=4700*, *lambdamax=9400*, *step=1.25*, *outcube_name=None*, *filter_for_nan=False*, ***kwargs*)

Create a reference cube using an input one, and overriding the lambda part, to get a new WCS

Parameters

- **lambdamin** –
- **lambdamax** –
- **step** –
- **outcube_name** –
- **filter_for_nan** –
- ****kwargs** –

Returns

the name of the folder where
the output cube is, and its name

Return type
cube_folder, outcube_name (str, str)

extract_onespectral_cube(*wave1=6500.0*, *outcube_name=None*, ***kwargs*)

Create a single pixel cube extracted from this one.

Parameters

- **wave1** (*float*) – Value of the wavelength to extract. In Angstroems.
- **outcube_name** (*str*) – Name of the output cube
- **prefix** (*str*) – If outcube_name is None (default), use that prefix to append in front of the input cube name (same folder)

Returns

- *A new cube with only 2 lambda. To be used as a WCS reference for*
- *masks.*

get_emissionline_image(*line=None*, *velocity=0.0*, *redshift=None*, *lambda_window=10.0*, *medium='vacuum'*)

Get a narrow band image around Ha

Input

lambda_window: in Angstroems (10 by default). Width of the window of integration medium: vacuum or air (string, 'vacuum' by default) velocity: default is 0. (km/s) redshift: default is None. Overwrite velocity if provided. line: name of the emission line (see emission_lines dictionary)

`get_filter_image`(*filter_name=None*, *own_filter_file=None*, *filter_folder=''*, *dict_filters=None*)

Get an image given by a filter. If the filter belongs to the filter list, then use that, otherwise use the given file

`get_image_from_cube`(*central_lambda=None*, *lambda_window=0*)

Get image from integrated cube, with spectral pixel centred at central_lambda and with a lambda_window of lambda_window

`get_quadrant_spectra_from_cube`(*pixel_window=0*)

Get quadrant spectra from the Cube

Input

pixel_window : pixel_window of integration

`get_set_spectra`()

Get a set of standard spectra from the Cube

`get_spectrum_from_cube`(*nx=None*, *ny=None*, *pixel_window=0*, *title='Spectrum'*)

Get a spectrum from the cube with centre defined in pixels with nx, ny and a window of 'pixel_window'

`get_whiteimage_from_cube`()

`mask_trail`(*pq1=[0, 0]*, *pq2=[10, 10]*, *width=1.0*, *margins=0.0*, *reset=False*, *save=True*, ***kwargs*)

Build a cube mask from 2 points measured from a trail on an image

Input

pq1: array or tuple (float)
p and q coordinates of point 1 along the trail

pq2: array or tuple (float)
p and q coordinates of point 2 along the trail

width: float
Value (in pixel) of the full slit width to exclude

margins: float
Value (in pixel) to extend the slit beyond the 2 extrema If 0, this means limiting it to the extrema themselves. Default is None, which mean infinitely long slit

reset (bool): if True, reset the mask before masking the slit save (bool): if True, save the masked cube

`rebin_spatial`(*factor*, *mean=False*, *inplace=False*, *full_covariance=False*, ***kwargs*)

Combine neighboring pixels to reduce the size of a cube by integer factors along each axis.

Each output pixel is the mean of n pixels, where n is the product of the reduction factors in the factor argument. Uses mpdaf rebin function, but add a normalisation factor if mean=False (sum). It also updates the unit by just copying the old one.

Input

factor
[(int or (int,int))] Factor by which the spatial dimensions are reduced

mean
[bool] If True, taking the mean, if False (default) summing

inplace
[bool] If False (default) making a copy. Otherwise using the present cube.

full_covariance: bool
If True, will assume that spaxels are fully covariant. This means that the variance will be normalised by sqrt(N) where N is the number of summed spaxels. Default is False

returns
Cube

rtype
rebinned cube

`save_mask`(*mask_name='dummy_mask.fits'*)

Save the mask into a 0-1 image

`class pymusepipe.mpdaf_pipe.MuseCubeMosaic`(*ref_wcs*, *folder_refcube=''*, *folder_cubes=''*, *prefix_cubes='DATACUBE_FINAL_WCS'*, *list_suffix=[]*, *use_fixed_cubes=True*, *excluded_suffix=[]*, *included_suffix=[]*, *prefix_fixed_cubes='tmask'*, *verbose=False*, *pointing_table=None*, *list_pointings=None*, *dict_psf={}*, *list_cubes=None*, ***kwargs*)

Bases: `CubeMosaic`

`build_list`(*folder_cubes=None*, *prefix_cubes=None*, *list_cubes=None*, ***kwargs*)

Building the list of cubes to process

Parameters

- **folder_cubes** (*str*) – folder for the cubes
- **prefix_cubes** (*str*) – prefix to be used

`convolve_cubes`(*target_fwhm*, *target_nmoffat=None*, *target_function='gaussian'*, *suffix='conv'*, ***kwargs*)

Parameters

- **target_fwhm** –
- **target_nmoffat** –
- **input_function** –
- **target_function** –
- **suffix** –
- ****kwargs** –

Returns:

`property cube_names`

property list_cubes

madcombine(*folder_cubes=None*, *outcube_name='dummy.fits'*, *fakemode=False*, *mad=True*)

Combine the CubeMosaic and write it out.

Parameters

- **folder_cubes** (*str*) – name of the folder for the cube [None]
- **outcube_name** (*str*) – name of the outcube
- **mad** (*bool*) – using mad or not [True]

Creates:

A new cube, combination of all input cubes listes in CubeMosaic

property ncubes

print_cube_names()

class pymusepipe.mpdaf_pipe.MuseFilter(*filter_name='Cousins_R'*, *filter_fits_file='filter_list.fits'*, *filter_ascii_file=None*)

Bases: object

read()

Reading the data in the file

class pymusepipe.mpdaf_pipe.MuseImage(*source=None*, ***kwargs*)

Bases: Image

Wrapper around the mpdaf Image functionalities

get_fwhm_startend()

Get range of FWHM

mask_trail(*pq1=[0, 0]*, *pq2=[10, 10]*, *width=0.0*, *reset=False*, *extent=None*)

Build an image mask from 2 points measured from a trail

Input

pq1: array or tuple (float)

p and q coordinates of point 1 along the trail

pq2: array or tuple (float)

p and q coordinates of point 2 along the trail

width: float

Value (in pixel) of the full slit width to exclude

extent: float

Value (in pixel) to extend the slit beyond the 2 extrema If 0, this means limiting it to the extrema themselves. Default is None, which mean infinitely long slit

reset_mask()

Resetting the Image mask

save_mask(*mask_name='dummy_mask.fits'*)

Save the mask into a 0-1 image

class pymusepipe.mpdaf_pipe.**MuseSetImages**(**args*, ***kwargs*)

Bases: `list`

Set of images

update(***kwargs*)

class pymusepipe.mpdaf_pipe.**MuseSetSpectra**(**args*, ***kwargs*)

Bases: `list`

Set of spectra

update(***kwargs*)

class pymusepipe.mpdaf_pipe.**MuseSkyContinuum**(*filename*)

Bases: `object`

integrate(*muse_filter*, *ao_mask=False*)

Integrate a sky continuum spectrum using a certain filter file. If the file is a fits file, use it as the MUSE filter list. Otherwise use it as an ascii file

Input

muse_filter: MuseFilter

read()

Read sky continuum spectrum from MUSE data reduction

save_normalised(*norm_factor=1.0*, *prefix='norm'*, *overwrite=False*)

Normalises a sky continuum spectrum and save it within a new fits file

Input

norm_factor: float
Scale factor to multiply the input continuum

prefix: str
Prefix for the new continuum fits name. Default is 'norm', so that the new file is 'norm_oldname.fits'

overwrite: bool
If True, existing file will be overwritten. Default is False.

set_normfactor(*background*, *filter_name='Cousins_R'*)

Get the normalisation factor given a background value Takes the background value and the sky continuuum spectrum and convert this to the scaling Ks needed for this sky continuum The principle relies on having the background measured as: MUSE_calib = ((MUSE - Sky_cont) + Background) * Norm

as measured from the alignment procedure.

Since we want: MUSE_calib = ((MUSE - Ks * Sky_cont) + 0) * Norm

This means that: Ks * Sky_cont = Sky_cont - Background ==> Ks = 1 - Background / Sky_cont

So we integrate the Sky_cont to get the corresponding S value and then provide Ks as 1-B/S

Input

background: float
Value of the background to consider

filter_name: str
Name of the filter to consider

`class pymusepipe.mpdaf_pipe.MuseSpectrum`(*source=None*, ***kwargs*)

Bases: `Spectrum`

Wrapper around the mpdaf Spectrum functionalities

`class pymusepipe.mpdaf_pipe.PixTableToMask`(*pixtable_name*, *image_name*, *suffix_out='tmask'*)

Bases: `object`

This class is meant to just be a simple tool to mask out some regions from the PixTable using Image masks

`create_mask`(*pq1=[0, 0]*, *pq2=[10, 10]*, *width=0.0*, *reset=False*, *mask_name='dummy_mask.fits'*, *extent=None*, ***kwargs*)

Create the mask and save it in one go

Input

pq1: array or tuple (float)
p and q coordinates of point 1 along the trail

pq2: array or tuple (float)
p and q coordinates of point 2 along the trail

width: float
Value (in pixel) of the full slit width to exclude

reset: bool
By default False, so the mask goes on top of the existing one If True, will reset the mask before building it.

extent: float
Value (in pixel) to extend the slit beyond the 2 extrema If 0, this means limiting it to the extrema themselves. Default is None, which mean infinitely long slit

`imshow`(***kwargs*)

Just showing the image

`mask_pixtable`(*mask_name=None*, ***kwargs*)

Use the Image Mask and create a new Pixtable

Input

mask_name: str
Name of the mask to be used (FITS file)

use_folder: bool
If True, use the same folder as the Pixtable Otherwise just write where you stand

suffix_out: str
Suffix for the name of the output Pixtable If provided, will overwrite the one in self.suffix_out

`save_mask`(*mask_name='dummy_mask.fits'*, *use_folder=True*)

Saving the mask from the Image into a fits file

Input

mask_name: str
Name of the fits file for the mask

use_folder: bool
If True (default) will look for the mask in the image_folder. If False, will just look for it where the command is run.

Creates

A fits file with the mask as 0 and 1

`pymusepipe.mpdaf_pipe.get_sky_spectrum`(*specname*)

Read sky spectrum from MUSE data reduction

`pymusepipe.mpdaf_pipe.integrate_spectrum`(*spectrum*, *wave_filter*, *throughput_filter*, *ao_mask=False*)

Integrate a spectrum using a certain Muse Filter file.

Input

spectrum: Spectrum
Input spectrum given as an mpdaf Spectrum

wave_filter: float array
Array of wavelength for the filter

throughput_filter: float array
Array of throughput (between 0 and 1) for the filter. Should be the same dimension (1D, N floats) as wave_filter

`pymusepipe.mpdaf_pipe.rotate_cube_wcs`(*cube_name*, *cube_folder=''*, *outwcs_folder=None*, *rotangle=0.0*, ***kwargs*)

Routine to remove potential Nan around an image and reconstruct an optimal WCS reference image. The rotation angle is provided as a way to optimise the extent of the output image, removing Nan along X and Y at that angle.

Parameters

- **cube_name** (*str*) – input image name. No default.
- **cube_folder** (*str*) – input image folder [‘’]

- **outwcs_folder** (*str*) – folder where to write the output frame. Default is None which means that it will use the folder of the input image.
- **rotangle** (*float*) – rotation angle in degrees [0]
- ****kwargs** – in_suffix (str): in suffix to remove from name ['prealign'] out_suffix (str): out suffix to add to name ['rotwcs'] margin_factor (float): factor to extend the image [1.1]

Returns:

pymusepipe.mpdaf_pipe.**rotate_image_wcs**(*ima_name*, *ima_folder=''*, *outwcs_folder=None*, *rotangle=0.0*, ***kwargs*)

Routine to remove potential Nan around an image and reconstruct an optimal WCS reference image. The rotation angle is provided as a way to optimise the extent of the output image, removing Nan along X and Y at that angle.

Input

ima_name: str
input image name. No default.

ima_folder: str default='', optional
input image folder

outwcs_folder: str, optional
folder where to write the output frame. Default is None which means that it will use the folder of the input image.

rotangle: float default=0, optional
rotation angle in degrees

in_suffix: str default='prealign'
in suffix to remove from name

out_suffix: str default='rotwcs'
out suffix to add to name

margin_factor: float
factor to extend the image [1.1]

pymusepipe.musepipe module

MUSE-PHANGS core module. This defines the main class (MusePipe) which can be used throughout this package.

This module is a complete rewrite of a pipeline wrapper for the MUSE dataset. All classes and objects were refactored.

However, the starting point of this package has been initially inspired by several pieces of python codes developed by various individiduals, including Kyriakos and Martina from the GTO MUSE MAD team and further rewritten by Mark van den Brok. Hence: a big Thanks to all three for this!

Note that several python packages exist which would provide similar (or better) functionalities.

Eric Emsellem adapted a version from early 2017, provided by Mark and adapted it for the needs of the PHANGS project (PI Schinnerer). It was further refactored starting from scratch but keeping a few initial ideas.

class pymusepipe.musepipe.**MusePipe**(*targetname=None*, *dataset=1*, *folder_config='Config/'*, *rc_filename=None*, *cal_filename=None*, *log_filename='MusePipe.log'*, *verbose=True*, *musemode='WFM-NOAO-N'*, *checkmode=True*, *strong_checkmode=False*, ***kwargs*)

Bases: `PipePrep`, `PipeRecipes`

Main Class to define and run the MUSE pipeline, given a certain galaxy name. This is the main class used throughout the running of the pipeline which contains functions and attributes all associated with the reduction of MUSE exposures.

It inherits from the PipePrep class, which prepares the recipes for the running of the MUSE pipeline, and Piperecipes which has the recipes described.

goto_folder(*newpath*, *addtolog=False*)

Changing directory and keeping memory of the old working one

Parameters

- **newpath** (`str`) – Path where to go to
- **addtolog** (`bool [False]`) – Adding the folder move to the log file

goto_origfolder(*addtolog=False*)

Go back to original folder

goto_prevfolder(*addtolog=False*)

Go back to previous folder

Parameters

addtolog (`bool [False]`) – Adding the folder move to the log file

init_raw_table(*reset=False*, ***kwargs*)

Create a fits table with all the information from the Raw files. Also create an astropy table with the same info

Parameters

reset (`bool [False]`) – Resetting the raw astropy table if True

property musemode

Mode for MUSE

print_musemodes()

Print out the list of allowed muse modes

read_all_astro_tables(*reset=False*)

Initialise all existing Astropy Tables

read_astropy_table(*expotype=None*, *stage='master'*)

Read an existing Masterfile data table to start the pipeline

retrieve_geoastro_name(*date_str*, *filetype='geo'*, *fieldmode='wfm'*)

Retrieving the astrometry or geometry fits file name

Parameters

- **date_str** (`str`) – Date as a string (DD/MM/YYYY)
- **filetype** (`str`) – 'geo' or 'astro', type of the needed file
- **fieldmode** (`str`) – 'wfm' or 'nfm' - MUSE mode

save_expo_table(*expotype*, *tpl_gtable*, *stage='master'*, *fits_tablename=None*, *aggregate=True*, *suffix=''*, *overwrite=None*, *update=None*)

Save the Expo (Master or not) Table corresponding to the expotype

set_fullpath_names()

Create full path names to be used

sort_raw_tables(*checkmode=None*, *strong_checkmode=None*)

Provide lists of exposures with types defined in the dictionary after excluding those with the wrong MUSE mode if checkmode is set up.

Input

checkmode: boolean
Checking the MUSE mode or not. Default to None, namely it won't use the value set here but the value predefined in self.checkmode.

strong_checkmode: boolean
Strong check, namely in case you still wish to force the MUSE mode even for files which are not mode specific (e.g., BIAS). Default to None, namely it uses the self.strong_checkmode which was already set up at start.

pymusepipe.prep_recipes_pipe module

MUSE-PHANGS preparation recipe module

class pymusepipe.prep_recipes_pipe.**PipePrep**(*first_recipe=1*, *last_recipe=None*)

Bases: *SofPipe*

PipePrep class prepare the SOF files and launch the recipes

get_align_group(*name_ima_reference=None*, *list_expo=[ ]*, *line=None*, *suffix=''*, *bygroup=False*, ***kwargs*)

Extract the needed information for a set of exposures to be aligned

static print_recipes()

Printing the list of recipes

run_align_bydataset(*sof_filename='exp_align_bydataset'*, *expotype='OBJECT'*, *list_expo=[ ]*, *stage='processed'*, *line=None*, *suffix=''*, *tpl='ALL'*, ***kwargs*)

Aligning the individual exposures from a dataset using the emission line region With the muse exp_align routine

Parameters

- **sof_filename** (*string (without the file extension)*) – Name of the SOF file which will contain the Bias frames
- **tpl** (*ALL by default or a special tpl time*) –

run_align_bygroup(*sof_filename='exp_align_bygroup'*, *expotype='OBJECT'*, *list_expo=[ ]*, *stage='processed'*, *line=None*, *suffix=''*, *tpl='ALL'*, ***kwargs*)

Aligning the individual exposures from a dataset using the emission line region With the muse exp_align routine

Parameters

- **sof_filename** (*string (without the file extension)*) – Name of the SOF file which will contain the Bias frames
- **tpl** (*ALL by default or a special tpl time*) –

run_autocal_sky(*sof_filename='scipost'*, *expotype='SKY'*, *AC_suffix='_AC'*, *tpl='ALL'*, ***extra_kwargs*)

Launch the scipost command to get individual exposures in a narrow band filter

run_bias(*sof_filename='bias'*, *tpl='ALL'*, *update=None*)

Reducing the Bias files and creating a Master Bias Will run the esorex muse_bias command on all Biases

Parameters

- **sof_filename** (*string (without the file extension)*) – Name of the SOF file which will contain the Bias frames
- **tpl** (*ALL by default or a special tpl time*) –

run_check_align(*name_offset_table*, *sof_filename='scipost'*, *expotype='OBJECT'*, *tpl='ALL'*, *line=None*, *suffix=''*, *folder_offset_table=None*, ***extra_kwargs*)

Launch the scipost command to get individual exposures in a narrow band filter to check if the alignments are ok (after rotation and using a given offset_table)

run_combine_dataset(*sof_filename='exp_combine'*, *expotype='OBJECT'*, *list_expo=[]*, *stage='processed'*, *tpl='ALL'*, *lambdaminmax=[4000.0, 10000.0]*, *suffix=''*, ***kwargs*)

Produce a cube from all frames in the dataset list_expo or tpl specific arguments can still reduce the selection if needed

run_fine_alignment(*name_ima_reference=None*, *nexpo=1*, *list_expo=[]*, *line=None*, *bygroup=False*, *reset=False*, ***kwargs*)

Run the alignment on this dataset using or not a reference image

run_flat(*sof_filename='flat'*, *tpl='ALL'*, *update=None*)

Reducing the Flat files and creating a Master Flat Will run the esorex muse_flat command on all Flats

Parameters

- **sof_filename** (*string (without the file extension)*) – Name of the SOF file which will contain the Bias frames
- **tpl** (*ALL by default or a special tpl time*) –

run_lsf(*sof_filename='lsf'*, *tpl='ALL'*, *update=None*)

Reducing the LSF files and creating the LSF PROFILE Will run the esorex muse_lsf command on all Flats

Parameters

- **sof_filename** (*string (without the file extension)*) – Name of the SOF file which will contain the Bias frames
- **tpl** (*ALL by default or a special tpl time*) –

run_phangs_recipes(*fraction=0.8*, *illum=True*, *skymethod='model'*, ***kwargs*)

Running all PHANGS recipes in one shot Using the basic set up for the general list of recipes

Input

fraction: float
Fraction of sky to consider in sky frames for the sky spectrum Default is 0.8.

illum: bool
Default is True (use illumination during twilight calibration)

skymethod: str
Default is "model".

run_prep_align(*sof_filename='scipost'*, *expotype='OBJECT'*, *tpl='ALL'*, *line=None*, *suffix=''*, ***extra_kwargs*)

Launch the scipost command to get individual exposures in a narrow band filter

run_recipes(***kwargs*)

Running all recipes in one shot

Input

fraction: float
Fraction of sky to consider in sky frames for the sky spectrum Default is 0.8.

illum: bool
Default is True (use illumination during twilight calibration)

skymethod: str
Default is "model".

filter_for_alignment: str
Default is defined in config_pipe

line: str
Default is None as defined in config_pipe

lambda_window: float
Default is 10.0 as defined in config_pipe

run_scibasic(*sof_filename='scibasic'*, *expotype='OBJECT'*, *tpl='ALL'*, *illum=True*, *update=True*, *overwrite=True*)

Reducing the files of a certain category and creating the PIXTABLES Will run the esorex muse_scibasic

Parameters

- **sof_filename** (*string (without the file extension)*) – Name of the SOF file which will contain the Bias frames
- **tpl** (*ALL by default or a special tpl time*) –

run_scibasic_all(*list_object=['OBJECT', 'SKY', 'STD']*, *tpl='ALL'*, *illum=True*, ***kwargs*)

Running scibasic for all objects in list_object Making different sof for each category

run_scipost(*sof_filename='scipost'*, *expotype='OBJECT'*, *tpl='ALL'*, *stage='processed'*, *list_expo=[]*, *lambdaminmax=[4000.0, 10000.0]*, *suffix=''*, ***kwargs*)

Scipost treatment of the objects Will run the esorex muse_scipost routine

Input

sof_filename: string (without the file extension)
Name of the SOF file which will contain the Bias frames

tpl: ALL by default or a special tpl time list_expo: list of integers

Exposure numbers. By default, an empty list which means that all exposures will be used.

lambdaminmax: tuple of 2 floats
Minimum and Maximum wavelength to pass to the muse_scipost recipe

suffix: str
Suffix to add to the input pixtables.

norm_skycontinuum: bool
Normalise the skycontinuum or not. Default is False. If normalisation is to be done, it will use the offset_table and the tabulated background value to renormalise the sky continuum.

skymethod: str
Type of skymethod. See MUSE manual.

offset_list: bool
If True, using an OFFSET list. Default is True.

name_offset_table: str
Name of the offset table table. If not provided, will use the default name produced during the pipeline run.

filter_for_alignment: str
Name of the filter used for alignment. Default is self.filter_for_alignment

filter_list: str
List of filters to be considered for reconstructed images. By Default will use the list in self.filter_list.

run_scipost_perexpo(*sof_filename='scipost'*, *expotype='OBJECT'*, *list_tplexpo='ALL'*, *stage='processed'*, *suffix=''*, *offset_list=False*, ***kwargs*)

Launch the scipost command exposure per exposure

Input

See run_scipost parameters

run_scipost_sky()

Run scipost for the SKY with no offset list and no skymethod

run_sky(*sof_filename='sky'*, *tpl='ALL'*, *fraction=0.8*, *update=None*, *overwrite=True*)

Reducing the SKY after they have been scibasic reduced Will run the esorex muse_create_sky routine

Parameters

- **sof_filename** (*string (without the file extension)*) – Name of the SOF file which will contain the Bias frames
- **tpl** (*ALL by default or a special tpl time*) –

run_standard(*sof_filename='standard'*, *tpl='ALL'*, *update=None*, *overwrite=True*)

Reducing the STD files after they have been obtained Running the muse_standard routine

Parameters

- **sof_filename** (*string (without the file extension)*) – Name of the SOF file which will contain the Bias frames
- **tpl** (*ALL by default or a special tpl time*) –

run_twilight(*sof_filename='twilight'*, *tpl='ALL'*, *update=None*, *illum=True*)

Reducing the files and creating the TWILIGHT CUBE. Will run the esorex muse_twilight command on all TWILIGHT

Parameters

- **sof_filename** (*string (without the file extension)*) – Name of the SOF file which will contain the Bias frames
- **tpl** (*ALL by default or a special tpl time*) –

run_wave(*sof_filename='wave'*, *tpl='ALL'*, *update=None*)

Reducing the WAVE-CAL files and creating the Master Wave Will run the esorex muse_wave command on all Flats

Parameters

- **sof_filename** (*string (without the file extension)*) – Name of the SOF file which will contain the Bias frames
- **tpl** (*ALL by default or a special tpl time*) –

save_fine_alignment(*name_offset_table=None*)

Save the fine dataset alignment

select_tpl_files(*expotype=None*, *tpl='ALL'*, *stage='raw'*)

Selecting a subset of files from a certain type

pymusepipe.prep_recipes_pipe.**add_listpath**(*suffix*, *paths*)

Add a suffix to a list of path and normalise them

pymusepipe.prep_recipes_pipe.**norm_listpath**(*paths*)

Normalise the path for a list of paths

pymusepipe.prep_recipes_pipe.**print_my_function_name**(*f*)

Function to provide a print of the name of the function Can be used as a decorator

pymusepipe.recipes_pipe module

MUSE-PHANGS recipe module

class pymusepipe.recipes_pipe.**PipeRecipes**(*nifu=-1*, *first_cpu=0*, *ncpu=24*, *list_cpu=[]*, *likwid='likwid-pin -c N:'*, *fakemode=False*, *domerge=True*, *nocache=False*, *nochecksum=True*)

Bases: `object`

PipeRecipes class containing all the esorex recipes for MUSE data reduction

property checksum

property esorex

joinprod(*name*)

property merge

recipe_align(*sof*, *dir_products*, *namein_products*, *nameout_products*, *tpl*, *group*, *threshold=10.0*, *srcmin=3*, *srcmax=80*, *fwhm=5.0*)

Running the muse_exp_align recipe

recipe_bias(*sof*, *dir_bias*, *name_bias*, *tpl*)

Running the esorex muse_bias recipe

recipe_combine(*sof*, *dir_products*, *name_products*, *tpl*, *expotype*, *suffix_products=''*, *suffix_prefinalnames=''*, *prefix_products=''*, *save='cube'*, *pixfrac=0.6*, *suffix=''*, *format_out='Cube'*, *filter_list='white'*, *lambdamin=4000.0*, *lambdamax=10000.0*)

Running the muse_exp_combine recipe for one single dataset

recipe_combine_pointings(*sof*, *dir_products*, *name_products*, *suffix_products=''*, *suffix_prefinalnames=''*, *prefix_products=''*, *save='cube'*, *pixfrac=0.6*, *suffix=''*, *format_out='Cube'*, *filter_list='white'*, *lambdamin=4000.0*, *lambdamax=10000.0*)

Running the muse_exp_combine recipe for pointings

recipe_flat(*sof*, *dir_flat*, *name_flat*, *dir_trace*, *name_trace*, *tpl*)

Running the esorex muse_flat recipe

recipe_lsf(*sof*, *dir_lsf*, *name_lsf*, *tpl*)

Running the esorex muse_lsf recipe

recipe_scibasic(*sof*, *tpl*, *expotype*, *dir_products=None*, *name_products=[]*, *suffix=''*)

Running the esorex muse_scibasic recipe

recipe_scipost(*sof*, *tpl*, *expotype*, *dir_products=''*, *name_products=['']*, *suffix_products=['']*, *suffix_prefinalnames=['']*, *suffix_postfinalnames=['']*, *list_expo=[]*, *save='cube,skymodel'*, *filter_list='white'*, *skymethod='model'*, *pixfrac=0.8*, *darcheck='none'*, *skymodel_frac=0.05*, *astrometry='TRUE'*, *lambdamin=4000.0*, *lambdamax=10000.0*, *suffix=''*, *autocalib='none'*, *rvcorr='bary'*, ***kwargs*)

Running the esorex muse_scipost recipe

recipe_sky(*sof*, *dir_sky*, *name_sky*, *tpl*, *iexpo=1*, *fraction=0.8*)

Running the esorex muse_stc recipe

recipe_std(*sof*, *dir_std*, *name_std*, *tpl*)

Running the esorex muse_stc recipe

recipe_twilight(*sof*, *dir_twilight*, *name_twilight*, *tpl*)

Running the esorex muse_twilight recipe

recipe_wave(*sof*, *dir_wave*, *name_wave*, *tpl*)

Running the esorex muse_wavecal recipe

run_oscommand(*command*, *log=True*)

Running an os.system shell command Fake mode will just spit out the command but not actually do it.

write_errlogfile(*text*)

Writing in log file

write_logfile(*text*, *addext=''*)

Writing in log file

write_outlogfile(*text*)

Writing in log file

pymusepipe.target_sample module

MUSE-PHANGS target sample module

class pymusepipe.target_sample.**MusePipeSample**(*TargetDic*, *rc_filename=None*, *cal_filename=None*, *folder_config=''*, *first_recipe=1*, ***kwargs*)

Bases: `object`

combine_target(*targetname=None*, ***kwargs*)

Run the combine recipe. Shortcut for combine[targetname].run_combine()

combine_target_per_pointing(*targetname=None*, *wcs_from_pointing=True*, ***kwargs*)

Run the combine recipe. Shortcut for combine[targetname].run_combine()

convolve_mosaic_per_pointing(*targetname=None*, *list_pointings=None*, *dict_psf={}*, *target_fwhm=0.0*, *target_nmoffat=None*, *target_function='gaussian'*, *suffix=None*, *best_psf=True*, *min_dfwhm=0.2*, *fakemode=False*, ***kwargs*)

Convolve the datacubes listed in a mosaic with some target function and FWHM. It will try to homogeneise all individual cubes to that target PSF.

Parameters

- **targetname** (*str*) – name of the target
- **list_pointings** (*list*) – list of pointing numbers for the list of pointings to consider
- **dict_psf** (*dict*) – dictionary providing individual PSFs per pointing
- **target_fwhm** (*float*) – target FWHM for the convolution [arcsec]
- **target_nmoffat** (*float*) – tail factor for the moffat function.
- **target_function** (*str*) – 'moffat' or 'gaussian' ['gaussian']
- **suffix** (*str*) – input string to be added
- **best_psf** (*bool*) – if True use the minimum overall possible value. If True it will overwrite all the target parameters.
- **min_dfwhm** (*float*) – minimum difference to be added in quadrature [in arcsec]
- **filter_list** (*list*) – list of filters to be used for reconstructing images
- **fakemode** (*bool*) – if True, will only initialise parameters but not proceed with the convolution.
- ****kwargs** –

Returns:

create_reference_wcs(*targetname=None*, *pointings_wcs=True*, *mosaic_wcs=True*, *wcs_refcube_name=None*, *refcube_name=None*, ***kwargs*)

Run the combine for individual exposures first building up a mask.

finalise_reduction(*targetname=None*, *rot_pixtab=False*, *create_wcs=True*, *create_expocubes=True*, *create_pixtables=True*, *create_pointingcubes=True*, *name_offset_table=None*, *folder_offset_table=None*, *list_datasets=None*, ***kwargs*)

Finalise the reduction steps by using the offset table, rotating the pixeltables, then reconstructing the PIXTABLE_REDUCED, produce reference WCS for each pointing, and then run the reconstruction of the final individual cubes

Parameters

- **targetname** (*str*) –
- **rot_pixtab** (*bool*) –
- **create_wcs** (*bool*) –
- **create_expocubes** (*bool*) –
- **name_offset_table** (*str*) –
- **folder_offset_table** (*str*) –
- ****kwargs** – include wcs_refcube_name: str

  Reference WCS (cube) to be used to project all cubes

  refcube_name: str
  Reference cube which will be use to build a reference WCS

  folder_refcube: str
  Name of the folder where to find the reference WCS or cube

Returns:

init_combine(*targetname=None*, *list_pointings=None*, *list_datasets=None*, *folder_offset_table=None*, *name_offset_table=None*, ***kwargs*)

Prepare the combination of targets. The use can provide a pointing table providing a given selection.

Input

targetname: str [None]
Name of target

list_pointings: list [or None=default= all pointings]
List of pointings (e.g., [1,2,3])

name_offset_table: str
Name of Offset table

****kwargs: additional keywords including**
pointing_table, pointing_table_folder, pointing_table_format

init_mosaic(*targetname=None*, *list_datasets=None*, *list_pointings=None*, *pointing_table=None*, ***kwargs*)

Prepare the combination of targets

Input

targetname: str [None]
Name of target

list_datasets: list [or None=default meaning all datasets]
List of datasets (e.g., [1,2,3])

pointing_table: PointingTable
Pointing Table to select a given set of exposures

`mosaic`(*targetname=None*, *list_pointings=None*, *init_mosaic=True*, *build_cube=True*, *build_images=True*, ***kwargs*)

Parameters

- `targetname` –
- `list_pointings` –
- `**kwargs` –

Returns:

`reduce_all_targets`(***kwargs*)

Reduce all targets already initialised

Input

first_recipe: int or str
One of the recipe to start with

last_recipe: int or str
One of the recipe to end with

`reduce_target`(*targetname=None*, *list_datasets=None*, ***kwargs*)

Reduce one target for a list of datasets

Input

targetname: str
Name of the target

list_datasets: list
Dataset numbers. Default is None (meaning all datasets indicated in the dictonary will be reduced)

first_recipe: str or int [1] last_recipe: str or int [max of all recipes]

Name or number of the first and last recipes to process

`reduce_target_postalign`(*targetname=None*, *list_datasets=None*, ***kwargs*)

Reduce target for all steps after pre-alignment

Input

targetname: str
Name of the target

list_datasets: list
Dataset numbers. Default is None (meaning all datasets indicated in the dictonary will be reduced)

reduce_target_prealign(*targetname=None*, *list_datasets=None*, ***kwargs*)

Reduce target for all steps before pre-alignment (included)

Input

targetname: str
Name of the target

list_datasets: list
Dataset numbers. Default is None (meaning all datasets indicated in the dictionary will be reduced)

rotate_pixtables_target(*targetname=None*, *list_datasets=None*, *folder_offset_table=None*, *name_offset_table=None*, *fakemode=False*, ***kwargs*)

Rotate all pixel table of a certain targetname and datasets

run_target_recipe(*recipe_name*, *targetname=None*, *list_datasets=None*, ***kwargs*)

Run just one recipe on target

Input

recipe_name: str targetname: str

Name of the target

list_datasets: list
Pointing numbers. Default is None (meaning all datasets indicated in the dictonary will be reduced)

run_target_scipost_perexpo(*targetname=None*, *list_datasets=None*, *list_pointings=None*, *folder_offset_table=None*, *name_offset_table=None*, ***kwargs*)

Build the cube per exposure using a given WCS

Parameters

- **targetname** –
- **list_datasets** –
- ****kwargs** –

Returns:

set_pipe_target(*targetname=None*, *list_datasets=None*, ***kwargs*)

Create the musepipe instance for that target and list of datasets

Input

targetname: str
Name of the target

list_datasets: list
Dataset numbers. Default is None (meaning all datasets indicated in the dictonary will be reduced)

config_args: dic
Dictionary including extra configuration parameters to pass to MusePipe. This allows to define a global configuration. If self.__phangs is set to True, this is overwritten with the default PHANGS configuration parameters as provided in config_pipe.py.

class pymusepipe.target_sample.**MusePipeTarget**(*targetname=''*, *subfolder='P100'*, *list_datasets=None*)

Bases: object

class pymusepipe.target_sample.**PipeDict**(**args*, ***kwargs*)

Bases: dict

Dictionary with extra attributes

run_on_all_keys(*funcname*)

Runs the given function on all the keys

setdefault(*key*, *value=None*)

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

update([*E*], ***F*) → None. Update D from dict/iterable E and F.

If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

pymusepipe.target_sample.**insert_suffix**(*filename*, *suffix=''*)

Create a new filename including the suffix in the name

Input

filename: str suffix: str

pymusepipe.target_sample.**update_calib_file**(*filename*, *subfolder=''*, *folder_config=''*)

Update the rcfile with a new root

Input

filename: str
Name of the input filename

folder_config: str
Default is "". Name of folder for filename

subfolder: str
Name of subfolder to add in the path

pymusepipe.util_image module

Utility functions for images in pymusepipe

class pymusepipe.util_image.**CircleZone**

Bases: *SelectionZone*

Define a Circular zone, defined by a center and a radius

select(*xin*, *yin*)

Define a selection within a circle

Input

xin, yin: 2d arrays
Input positions for the spaxels

class pymusepipe.util_image.**PointingTable**(*input_table=None*, ***kwargs*)

Bases: object

assign_pointings(***kwargs*)

Assign pointing according to distance rules. Will also update the centre values.

****kwargs: additional keywords including**
verbose: bool default=self.verbose overwrite: bool default=False

overwrite the pointing

Updates

The values of the 'pointing' column for the selected filenames ('select'==1)

property dict_allnames_in_datasets

Dictionary of the names per dataset

property dict_allnames_in_pointings

Dictionary of the names per pointing

property dict_names_in_datasets

Dictionary of the names per dataset

property dict_names_in_pointings

Dictionary of the names per pointing

property dict_tplexpo_per_dataset

property dict_tplexpo_per_pointing

property fullnameout

property fulltablename

list_colnames_ptable = ['filename', 'dataset', 'tpls', 'expo']

property list_datasets

List of unique datasets in the pointing table

property list_pointings

List of unique pointings in the pointing table

read(***kwargs*)

Read the input filename in given folder assuming a given format.

Input

filename: str, optional
Name of the filename

folder: str default='', optional
Name of the folder where to find the filename

table_format: str default='ascii'

rtype
self.qtable with the content of the file

scan_folder(*folder=None*, ***kwargs*)

Scan a folder to create a full pointing table

Input

folder: str default to None
If not provided, will use the default self.folder

****kwargs:**
Other keywords are passed to create_pointing_table_from_folder

Creates

Attribute qtable

select_datasets(***kwargs*)

Select all filenames with a given list of datasets

Input

list_datasets: list of int, optional

Updates

'select' values in the astropy pointing table according to the list of datasets

select_filename(*filename*, *verbose=False*)

Select the filename as provided, by putting the value of column 'select' to 1

Input

filename: str
Name of the file (see column 'filename')

Put the right value of the 'select' column to 1

`select_pointings`(***kwargs*)

Select all filenames with pointings in the pointing list

Input

list_pointings: list of int, optional

Updates

'select' values in the astropy pointing table according to the list of pointings

`select_pointings_and_datasets`(***kwargs*)

Select all filenames with that pointing or dataset number.

Input

list_pointings: list of int, optional list_datasets: list of int, optional

Updates

'select' values in the astropy pointing table according to the list of pointings and datasets

`property selected_filenames`

Return the list of filenames following the selection

Return type
list_filename

`set_select_value`(*filename*, *value=1*, *verbose=False*)

Set the value of the select column to 1, according to a given filename

filename: str value: int default 1

`unselect_filename`(*filename*, *verbose=False*)

Select the filename as provided, by putting the value of column 'select' to 1

Input

filename: str
Name of the file (see column 'filename')

Put the right value of the 'select' column to 1

`write(overwrite=False, **kwargs)`
Write out the table on disk, using the nameout and provided folder.

Parameters

- **overwrite** (*bool default=False*) –
- ****kwargs** – Valid keywords are folder: str nameout: str Extra keywords are passed to the astropy QTable.write() function
- **disk** (*Writes the pointing table on*) –

`class pymusepipe.util_image.RectangleZone`
Bases: *SelectionZone*

Define a rectangular zone, given by a center, a length, a width and an angle

`select(xin, yin)`

Define a selection within a rectangle
It can be rotated by an angle theta (in degrees)

Input

xin, yin: 2d arrays
Input positions for the spaxels

`class pymusepipe.util_image.SelectionZone(params=None)`
Bases: `object`

Parent class for Rectangle_Zone and Circle_Zone

Input

params: list of floats
List of parameters for the selection zone

`class pymusepipe.util_image.TrailZone`
Bases: *SelectionZone*

Define a Trail zone, defined by two points and a width

`select(xin, yin)`
Define a selection within trail

Input

xin, yin: 2d arrays
Input positions for the spaxels

`pymusepipe.util_image.check_column_set(input_table)`

Check the minimum column set for the Pointing table

Input

input_table: astropy Table

returns
bool

rtype
True if all names are in the table, False otherwise

`pymusepipe.util_image.chunk_stats(list_arrays, chunk_size=15)`

Cut the datasets in 2d chunks and take the median Return the set of medians for all chunks.

Parameters

- **list_arrays** (*list of np.arrays*) – List of arrays with the same sizes/shapes
- **chunk_size** (*int*) – number of pixel (one D of a 2D chunk) of the chunk to consider (Default value = 15)

Returns
median, standard – for the given datasets analysed in chunks.

Return type
2 arrays of the medians and standard deviations

`pymusepipe.util_image.compute_diagnostics(pointing_dict, center_dict)`

Compute the average and std of the distance between the exposures belonging to the same pointing.

Input

pointing_dict: dict
Dictionary for the pointings

center_dict: dict
dictionary of the files to be used

returns
Diagnostic – Each pointing key has its [mean, std] as value of the distionary

rtype
dict

`pymusepipe.util_image.create_offset_table(image_names, table_folder='', table_name='dummy_offset_table.fits', overwrite=False)`

Create an offset list table from a given set of images. It will use the MJD and DATE as read from the descriptors of the images. The names for these keywords is stored in the dictionary default_offset_table from config_pipe.py

Parameters

- **image_names** (*list of str*) – List of image names to be considered. (Default value = [])
- **table_folder** (*str*) – folder of the table (Default value = "")
- **table_name** (*str*) – name of the table to save ['dummy_offset_table.fits'] (Default value = "dummy_offset_table.fits")
- **overwrite** (*bool*) – if the table exists, it will be overwritten if set to True only. (Default value = False)
- **overwrite** – if the table exists, it will be overwritten if set to True only. (Default value = False)

Return type
A fits table with the output given name. (Default value = False)

`pymusepipe.util_image.crop_data(data, border=10)`

Crop a 2D data and return it cropped after a border has been removed (number of pixels) from each edge (borderx2 pixels are removed from each dimension)

Input

data: 2d array
Array which has the signal to be cropped

border: int
Number of pixels to be cropped at each edge

returns
cdata – Cropped data array

rtype
2d array

`pymusepipe.util_image.filter_list_with_pointingtable(input_list, pointing_table=None, verbose=True, str_dataset='OB', ndigits=3, list_pointings=None, filtername=None)`

Input

input_list: list of str
Input list of filenames to filter

pointing_table: PointingTable or QTable or Table str_dataset: str default=default_str_dataset ndigits: int default=default_ndigits filtername: str default=None verbose: bool default=True

`pymusepipe.util_image.filtermed_image(data, border=0, filter_size=2, keepnan=False)`

Process image by removing the borders and filtering it via a median filter

Input

data: 2d array
Array to be processed

border: int
Number of pixels to remove at each edge

filter_size: float
Size of the filtering (median)

returns
cdata – Processed array

rtype
2d array

`pymusepipe.util_image.flatclean_image(data, border=10, dynamic_range=10, median_window=10, threshold=0.0, squeeze=True, remove_bkg=True)`

Process image by squeezing the range, removing the borders and filtering it. The image is first filtered, then it is cropped. All values below a given minimum are set to 0 and all Nan set to 0 or infinity accordingly.

Input

data: 2d ndarray
Input array to process

dynamic_range: float [10]
Dynamic range used to squash the bright pixels down

median_window: int [10]
Size of the window used for the median filtering.

threshold: float [0]
Value of the minimum value allowed.

squeeze: bool
Squeeze the dynamic range by using the dynamic_range variable

crop: bool
Crop the borders using border as the variable

remove_bkg: remove the filter_medianed background

returns
flatcleaned_array

rtype
2d ndarray

`pymusepipe.util_image.get_centre_from_image_or_cube(filename, ext=1, dtype='image')`

Compute the coordinate of the center of the FOV from an image. Only pixels with actual signal are considered.

Input

file_name: st
name of the file to analyse

ext: int default=1, optional
extension where the data and WCS info are located. Defaults to 1.

dtype: str default='image', optional
type of file to be analyzed. It can be either image or cube. Defaults to image.

returns
SkyCoord – Coordinate of the center of the FOV

rtype
astropy.coordinates.SkyCoord

`pymusepipe.util_image.get_centre_from_pixtable(pixtable_name)`

Get the center of the FOV from pixtables

Input

pixtable_name: str
name of the pixeltable

returns
SkyCoord – Coordinates of the center of the field

rtype
astropy.coordinates.Skycoord

`pymusepipe.util_image.get_flux_range(data, border=15, low=2, high=98)`

Get the range of fluxes within the array by looking at percentiles.

Input

data: 2d array
Input array with signal to process

low, high: two floats (10, 99)
Percentiles to consider to filter

returns
lperc, hperc – Low and high percentiles

rtype
2 floats

`pymusepipe.util_image.get_normfactor(array1, array2, median_filter=True, border=0, convolve_data1=0.0, convolve_data2=0.0, chunk_size=10, threshold=0.0, add_background1=0)`

Get the normalisation factor for shifted and projected images. This function only consider the input images given by their data (numpy) arrays.

Input

array1: 2d np.array array2: 2d np.array

Input arrays. Should be the same size

median_filter: bool
If True, will median filter

convolve_muse: float [0]
Will convolve the image with index nima with a gaussian with that sigma. 0 means no convolution

convolve_reference: float [0]
Will convolve the reference image with a gaussian with that sigma. 0 means no convolution

border: int
Number of pixels to crop

threshold: float [None]
Threshold for the input image flux to consider

returns

- **data** (*2d array*)
- **refdata** (*2d array*) – The 2 arrays (input, reference) after processing
- **polypar** (*the result of an ODR regression*)

`pymusepipe.util_image.get_polynorm`(*array1*, *array2*, *chunk_size=15*, *threshold1=0.0*, *threshold2=0*, *percentiles=(0.0, 100.0)*, *sigclip=0*)

Find the normalisation factor between two arrays.

Including the background and slope. This uses the function regress_odr which is included in align_pipe.py and itself makes use of ODR in scipy.odr.ODR.

Parameters

- **array1** (*2D np.array*) –
- **array2** (*2D np.array*) – 2 arrays (2D) of identical shapes
- **chunk_size** (*int*) – Default value = 15
- **threshold1** (*float*) – Lower threshold for array1 (Default value = 0.)
- **threshold2** (*float*) – Lower threshold for array2 (Default value = 0)
- **percentiles** (*list of 2 floats*) – Percentiles (Default value = [0., 100.])
- **sigclip** (*float*) – Sigma clipping factor (Default value = 0)

Returns
result – Result of the regression (ODR)

Return type
python structure

`pymusepipe.util_image.group_exposures_per_pointing`(*list_files*, *target_path=''*, *limit=10.0*, *unit=Unit('arcsec')*, *ext=1*, *dtype='image'*)

Separate a list of files in pointings based on their proximity.

This function assign each file to a pointing. Pointings are defined as groups of exposures whose distance between the centers falls within a certain limit. Once the groups of exposures have been defined, they are sorted, and then a pointing number starting from 1 is assigned to all of them. Some info on the average std of the eparation between exposures can be optionally computed.

Input

list_files: list
list of files to be reorganized in pointings

target_path: str default="
path of the target files

limit: float default=10
maximum separation for files to belong to the same pointing

unit: astropy unit default=u.arcsec
Unit of spatial distance (e.g., astropy.unit.arcsec)

ext: int default=1, optional
header extension where the WCS information is located.

dtype: str default 'image', optional
type of file to be analyzed. It can be pixtable, image or cube. Defaults to image.

returns

- **Pointing dictionary** (*Dict of [int, list]*) – Dictionary grouping the input files by pointing. The keys of the dictionary are the pointing numbers, and to each pointing a list of filenames is associated.
- **Diagnostic dictionary** (*Dict of [int, list], only if diagnostics*) – Dictionary containing basic information on the distance between exposures belonging to the same pointing. For each pointing, the mean and std of the distance with respect to a reference exposure is reported.

`pymusepipe.util_image.`**`group_xy_per_fieldofview`**(*center_dict*, *limit=<Quantity 10. arcsec>*)

Group exposures in pointings based on their proximity.

Input

center_dict: dict
Dictionary containing a list of filenames and their coordinates.

limit: Quantity default=10*.u.arcsec, optional
maximum separation for files to belong to the same pointing. Defaults to 10*u.arcsec.

Returns: Dict: [int, list]

Dictionary grouping the input files by pointing. The keys of the dictionary are the pointing numbers, and to each pointing a list of filenames is associated.

`pymusepipe.util_image.`**`mask_point_sources`**(*ima*, *fwhm=3*, *mask_radius=30.0*, *brightest=5*, *sigma=3.0*, *verbose=False*)

Find and mask point sources in an image by adding NaN

Input

ima: ndarray
Image to mask

fwhm: float
guess for the FWHM in pixel of the PSF. Defaults to 3.

mask_radius: float
Radius in pixels to mask around sources

brightest: int
Maximum number of bright stars to mask. Defaults to 5.

sigma: float
Sigma to clip the image

verbose: bool

returns
ima – with NaN where the mask applied

rtype
np.array

`pymusepipe.util_image.`**`my_linear_model`**(*b*, *x*)

Linear function for the regression.

Input

b
[1D np.array of 2 floats] Input 1D polynomial parameters (0=constant, 1=slope)

x
[np.array] Array which will be multiplied by the polynomial

rtype
An array = b[1] * (x + b[0])

`pymusepipe.util_image.`**`prepare_image`**(*data*, *median_filter=True*, *sigma=0.0*, *border=0*)

Median filter plus convolve the input image

Input

data: 2D np.array
Data to process

median_filter: bool
If True, will median filter

convolve float [0]
Will convolve the data with this gaussian width (sigma) 0 means no convolution

returns
data

rtype
2d array

`pymusepipe.util_image.regress_odr`(*x*, *y*, *sx*, *sy*, *beta0=(0.0, 1.0)*, *percentiles=(0.0, 100.0)*, *sigclip=0.0*)

Return an ODR linear regression using scipy.odr.ODR

Parameters

- **x** – numpy.array
- **y** – numpy.array Input array with signal
- **sx** – numpy.array
- **sy** – numpy.array Input array (as x,y) with standard deviations
- **beta0** – list or tuple of 2 floats Initial guess for the constant and slope
- **percentiles** – tuple or list of 2 floats Two numbers providing the min and max percentiles
- **sigclip** – float sigma factor for sigma clipping. If 0, no sigma clipping is performed

Returns
result of the ODR analysis

Return type
result

`pymusepipe.util_image.scan_filenames_from_folder`(*folder=''*, *prefix=''*, *suffix=''*, *ext='fits'*, ***kwargs*)

Scan a given folder and look for a set of filenames that could enter a pointing table. Those names are decrypted following a given scheme (extracting tpls, expo, dataset)

Input

folder: str default=''
Name of the folder to scan

prefix: str default=''
Prefix to be used to filter the filenames

suffix: str default=''
End of the word before the stem (extension)

ext: str default='fits'
Extension

returns
filename_table

rtype
astropy QTable with columns 'filename' 'tpls' 'dataset' 'expo'

`pymusepipe.util_image.scan_filenames_from_list`(*list_files*, ***kwargs*)

Extract values of dataset, tpls, expo from a list of names

Input

list_files: list of str kwargs: additional keywords including

str_dataset: str ndigits: int filtername: str

rtype
QTable including filenames, tpls, dataset, expo

`pymusepipe.util_image.select_spaxels(maskdict, maskname, x, y)`
Selecting spaxels defined by their coordinates using the masks defined by Circle or Rectangle Zones

pymusepipe.util_pipe module

MUSE-PHANGS utility functions for pymusepipe

`class pymusepipe.util_pipe.ExposureInfo(targetname, dataset, tpl, nexpo)`
Bases: `object`

`class pymusepipe.util_pipe.TimeStampDict(description='', myobject=None)`
Bases: `OrderedDict`

Class which builds a time stamp driven dictionary of objects

`create_new_timestamp(myobject=None)`
Create a new item in dictionary using a time stamp

`delete_timestamp(tstamp=None)`
Delete a key in the dictionary

`pymusepipe.util_pipe.abspath(path)`
Normalise the path to get it short but absolute

`pymusepipe.util_pipe.add_key_dataset_expo(imaname, iexpo, dataset)`
Add dataset and expo number to image

Input

imaname: str iexpo: int dataset: int

`pymusepipe.util_pipe.add_string(text, word='_', loc=0)`
Adding string at given location Default is underscore for string which are not empty.

Input

text (str): input text word (str): input word to be added loc (int): location in 'text'. [Default is 0=start]

If None, will be added at the end.

rtype
Updated text

`pymusepipe.util_pipe.analyse_musemode(musemode, field, delimiter='-')`
Extract the named field from the musemode

Input

musemode: str
Mode of the MUSE data to be analysed

field: str
Field to analyse ('ao', 'field', 'lambda_range')

delimiter: str
Character to delimit the fields to analyse

returns
val – Value of the field which was analysed (e.g., 'AO' or 'NOAO')

rtype
str

`pymusepipe.util_pipe.`**`append_file`**(*filename*, *content*)

Append in ascii file

`pymusepipe.util_pipe.`**`append_value_to_dict`**(*mydict*, *key*, *value*)

Append a value to key within a given dictionary. If the key does not exist it creates a list of 1 element for that key

Input

mydict: dict key: value:

rtype
Updated dictionary

`pymusepipe.util_pipe.`**`build_dict_datasets`**(*data_path=''*, *str_dataset='OB'*, *ndigits=3*)

Build a dictionary of datasets for each target in the sample

Input

data_path: str
Path of the target data

str_dataset: str default=default_str_dataset
Prefix string for datasets (see config_pipe.py)

ndigits: int default=default_ndigits
Number of digits to format the name of the dataset (see config_pipe.py)

returns
dict_dataset

rtype
dict

`pymusepipe.util_pipe.`**`build_dict_exposures`**(*target_path=''*, *str_dataset='OB'*, *ndigits=3*, *show_pointings=False*)

Build a dictionary of exposures using the list of datasets found for the given dataset path

Input

target_path: str
Path of the target data

str_dataset: str
Prefix string for datasets

ndigits: int
Number of digits to format the name of the dataset

returns
dict_expo – Dictionary of exposures in each dataset

rtype
dict

`pymusepipe.util_pipe.`**`check_filter_list`**(*filter_list*)

`pymusepipe.util_pipe.`**`create_time_name`**()

Create a time-link name for file saving purposes

Return: a string including the YearMonthDay_HourMinSec

`pymusepipe.util_pipe.`**`filter_list_to_str`**(*filter_list*)

`pymusepipe.util_pipe.`**`filter_list_with_suffix_list`**(*list_names*, *included_suffix_list=[]*, *excluded_suffix_list=[]*, *name_list=''*)

Filter a list using suffixes (to exclude or include)

Input

list_names: list of str included_suffix_list: list of str excluded_suffix_list: list of str name_list: str default=""

`pymusepipe.util_pipe.`**`formatted_time`**()

Return: a string including the formatted time

`pymusepipe.util_pipe.`**`get_dataset_name`**(*dataset=1*, *str_dataset='OB'*, *ndigits=3*)

Formatting for the dataset/pointing names using the number and the number of digits and prefix string

Input

dataset: int
Dataset (or Pointing) number

str_dataset: str
Prefix representing the dataset (or pointing)

ndigits: int
Number of digits to be used for formatting

rtype
string for the dataset/pointing name prefix

`pymusepipe.util_pipe.`**`get_dataset_tpl_nexpo`**(*filename*, *str_dataset='OB'*, *ndigits=3*, *filtername=None*)

Get the tpl and nexpo from a filename assuming it is at the end of the filename

Input

filename: str
Input filename

returns
tpl, nexpo

rtype
str, int

`pymusepipe.util_pipe.get_list_datasets(target_path='', str_dataset='OB', ndigits=3, verbose=False)`

Getting the list of existing datasets for a given target path

Input

target_path: str
Path of the target data

str_dataset: str
Prefix string for datasets

ndigits: int
Number of digits to format the name of the dataset

returns
list_datasets

rtype
list of int

`pymusepipe.util_pipe.get_list_exposures(dataset_path='', object_folder='Object/')`

Getting a list of exposures from a given path

Input

dataset_path: str
Folder name where the dataset is

returns
list_expos

rtype
list of int

`pymusepipe.util_pipe.get_list_reduced_pixtables(target_path='', list_datasets=None, suffix='', str_dataset='OB', ndigits=3, **kwargs)`

Provide a list of reduced pixtables

Input

target_path: str
Path for the target folder

list_datasets: list of int
List of integers, providing the list of datasets to consider

suffix: str
Additional suffix, if needed, for the names of the PixTables.

`pymusepipe.util_pipe.`**`get_list_targets`**(*folder=''*)

Getting a list of existing targets given path. This is done by simply listing the existing folders. This may need to be filtered.

Input

folder: str
Folder name where the targets are

returns
list_targets

rtype
list of str

`pymusepipe.util_pipe.`**`get_pointing_name`**(*pointing=1*, *str_pointing='P'*, *ndigits=3*)

Formatting for the names using the number and the number of digits and prefix string

Input

pointing: int
Pointing number

str_pointing: str
Prefix representing the pointing

ndigits: int
Number of digits to be used for formatting

rtype
string for the dataset/pointing name prefix

`pymusepipe.util_pipe.`**`get_tpl_nexpo`**(*filename*)

Get the tpl and nexpo from a filename assuming it is at the end of the filename

Input

filename: str
Input filename

returns
tpl, nexpo

rtype
str, int

`pymusepipe.util_pipe.lower_allbutfirst_letter(mystring)`

Lowercase all letters except the first one

`pymusepipe.util_pipe.lower_rep(text)`

Lower the text and return it after removing all underscores

Parameters
text (*str*) – text to treat

Returns
updated text (with removed underscores and lower-cased)

`pymusepipe.util_pipe.merge_dict(dict1, dict2)`

Merging two dictionaries by appending keys which are duplicated

Input

dict1: dict dict2: dict

returns
dict1 – merged dictionary

rtype
dict

`pymusepipe.util_pipe.normpath(path)`

Normalise the path to get it short

`pymusepipe.util_pipe.print_debug(text, **kwargs)`

Print debugging information

Input

text: str pipe: musepipe [None]

If provided, will print the text in the logfile

`pymusepipe.util_pipe.print_endline(text, **kwargs)`

`pymusepipe.util_pipe.print_error(text, **kwargs)`

Print error information

Input

text: str pipe: musepipe [None]

If provided, will print the text in the logfile

`pymusepipe.util_pipe.print_info(text, **kwargs)`

Print processing information

Input

text: str pipe: musepipe [None]

If provided, will print the text in the logfile

`pymusepipe.util_pipe.print_warning(text, **kwargs)`

`pymusepipe.util_pipe.reconstruct_filter_images(cubename, filter_list='white,Johnson_B,Johnson_V,Cousins_R,SDSS_g,SDSS_r,SDSS_ filter_fits_file='filter_list.fits')`

Reconstruct all images in a list of Filters cubename: str

Name of the cube

filter_list: str
List of filters, e.g., "Cousins_R,Johnson_I" By default, the default_filter_list from pymusepipe.config_pipe

filter_fits_file: str
Name of the fits file containing all the filter characteristics Usually in filter_list.fits (MUSE default)

`pymusepipe.util_pipe.safely_create_folder(path, verbose=True)`

Create a folder given by the input path This small function tries to create it and if it fails it checks whether the reason is that it is not a path and then warn the user

Input

path: str verbose: bool

Creates

A new folder if the folder does not yet exist

pymusepipe.version module

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Module contents

Copyright (C) 2017 ESO/Centre de Recherche Astronomique de Lyon (CRAL) print pymusepipe.__LICENSE__ for the terms of use

This package is a wrapper around the MUSE pipeline commands to reduce muse raw data frames. It includes modules for aligning and convolving the frames. It also has some basic routines wrapped around mpdaf, the excellent python package built around the MUSE PIXTABLES and reduced data.

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p