soxspipe Documentation

Release v0.10.2

Dave Young

2024

CONTENTS

The data-reduction pipeline for the SOXS instrument (a python package with command-line tools).

Documentation for soxspipe is hosted by Read the Docs (development version and master version). The code lives on github. Please report any issues you find here.

CHAPTER ONE

INSTALLATION

The best way to install or upgrade soxspipe is to use `conda` to install the package in its own isolated environment, as shown here:

```
conda create -n soxspipe python=3.9 soxspipe -c conda-forge
conda activate soxspipe
```

If you have previously installed soxspipe, a warning will be issued stating that a conda environment already exists; select 'y' when asked to remove the existing environment.

To check installation was successful run `soxspipe -v`. This should return the version number of the install.

CHAPTER TWO

HOW TO CITE SOXSPIPE

If you use soxspipe in your work, please cite using the following BibTeX entry:

```
@software{Young_soxspipe,
    author = {Young, David R.},
    doi = {10.5281/zenodo.8038264},
    license = {GPL-3.0-only},
    title = ,
    url = {https://zenodo.org/doi/10.5281/zenodo.8038264}
}
```

2.1 Quickstart Guide

Warning: This quickstart guide is subject to (much) change during the development of the pipeline. New features and ways of operating the pipeline are still being added. Current data taken in stare mode can be reduced to the point of sky subtraction.

2.1.1 Install

The best way to install soxspipe is to use conda and install the package in its own isolated environment (using either Anaconda or Minicoda), as shown here:

```
conda create -n soxspipe python=3.9 soxspipe -c conda-forge
conda activate soxspipe
```

If you have previously installed soxspipe, a warning will be issued stating that a conda environment already exists; select y when asked to remove the existing environment. This has proven to be the cleanest way to upgrade soxspipe.

To check installation was successful run soxspipe -v. This should return the version number of the installation.

2.1.2 Demo Data

The demo XShooter data (stare-mode) is of the X-ray binary SAX J1808.4-3658 taken during a 2019 outburst. You can download and unpack the data with the following commands:

```
curl -L "https://www.dropbox.com/s/t3adwc86bcwonkj/soxspipe-quickstart-demo-lite.tgz?
↪dl=1" > soxspipe-quickstart-demo.tgz
tar -xzvf soxspipe-quickstart-demo.tgz
```

You may also retrieve the raw data directly from the ESO archive with the following parameters:

```
RA = 18 08 27.54
Dec = -36 58 44.3
Night = 2019 08 30
Spectroscopy = XSHOOTER/VLT
Science
```

2.1.3 Preparing the Data-Reduction Workspace

Now you have a sample data set to work with, it is time to prepare the `soxspipe-quickstart-demo` workspace. Change into the `soxspipe-quickstart-demo` directory and run the `soxspipe prep` command:

```
cd soxspipe-quickstart-demo
soxspipe prep .
```

Once the workspace has been prepared, you should find it contains the following files and folders:

- `misc/`: a lost-and-found archive of non-fits files
- `raw_frames/`: all raw-frames to be reduced
- `sessions/`: directory of data-reduction sessions
- `sof/`: the set-of-files (sof) files required for each reduction step
- `soxspipe.db`: a sqlite database needed by the data-organiser, please do not delete

soxspipe reduces data within a `reduction session` and an initial `base` session is automatically created when running the `prep` command.

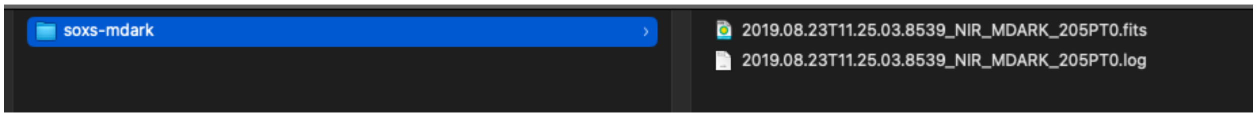
2.1.4 Reduce the Data

In most use case, you will want to reduce all of the raw frames contained within your workspace. To do this run the command:

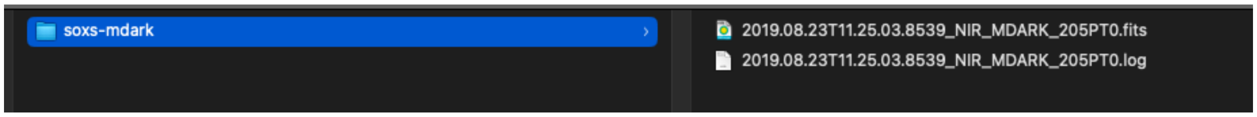
```
soxspipe reduce all .
```

2.2 Logging

When running a recipe, `soxspipe` writes informative logs to the terminal (stdout), allowing the user to keep track of the reduction progress in real time. For the sake of provenance, this same information is written to a log file adjacent to the recipe's product file(s).


soxs-mdark
2019.08.23T11.25.03.8539_NIR_MDARK_205PT0.fits
2019.08.23T11.25.03.8539_NIR_MDARK_205PT0.log

If the recipe happens to fail, a separate error log is written to the directory the product file should have been written to had the recipe succeeded. Error logs are named with a *"_ERROR.log"* suffix.


soxs-mdark
2019.08.23T11.25.03.8539_NIR_MDARK_205PT0.fits
2019.08.23T11.25.03.8539_NIR_MDARK_205PT0.log

2.3 Data Reduction Sessions

soxspipe reduces data within a 'reduction session'. These sessions are designed to be self-contained and isolated from other sessions, allowing a user to reduce the same set of raw data with multiple different pipeline settings. When a workspace is first prepared (using the `soxspipe prep` command), an initial `base` session is automatically created. Sessions are stored in the `sessions` directory of the workspace and contain their own settings file, products and QC directories. soxspipe will remember and use the latest session you were working with unless you create a new session or switch to another one.

To see which reduction session you are working within and all other sessions available, run the command:

```
soxspipe session ls
```

To create a new session, run the command:

```
soxspipe session new
```

The new session will be named with the date-time stamp of its creation date. Alternatively, you can also supply a memorable name for the session. The name can be up to 16 characters long and use alpha-numeric characters, - or _.

```
soxspipe session new my_supernova
```

Or to switch to an existing session, run the command:

```
soxspipe session <sessionId>
```

For user convenience, when you switch to a new session, the symbolic links found within the workspace root folder are automatically switched to point to the current session assets (`products`, `qc`, `sof`, `soxspipe.yaml`,`soxspipe.db` etc). If you run `ls -lrt` within your workspace root directory you will see these symlinks reported:

```
products -> ./sessions/base/products
qc -> ./sessions/base/qc
soxspipe.db -> ./sessions/base/soxspipe.db
soxspipe.yaml -> ./sessions/base/soxspipe.yaml
sof -> ./sessions/base/sof
```

2.4 `reduce`

In the simplest use case, you will want to reduce all of the raw frames contained within your workspace. To do this, first change directory to the base of your workspace, and then run the commands:

```
soxspipe prep .
soxspipe reduce all .
```

```
soxspipe
```

will now attempt to reduce all of the data in your workspace.

2.5 A Primer on SOXS Observation Modes

The glow of thermal emission from the earth's atmosphere, and even the telescope itself, start to dominate photon counts at NIR wavelengths, seriously contaminating signal from target sources. This thermal emission glow is typically dubbed *'sky background'* (despite the fact it originates in the foreground of the target source).

The sole reason SOXS has multiple observation modes is to provide astronomers with flexibility and a choice of methods they can employ to attempt to isolate and remove the noise generated by the sky-background from their science frames.

The 3 main observation modes are (click links for more detail):

1. Stare mode where the on-source sky-background is modelled, fitted and removed with software. Although removal of the background is not as accurate as the other observation modes, the time lost in overheads is lower.
2. Nodding mode employs an observational technique to measure the sky-background of an empty patch of sky near in time and spatially close to the source. This spectral measurement of the sky-background flux is simply removed from the on-source spectrum. Although generating more overhead time than stare-mode, unlike offset-mode, this technique allows for source and sky to be observed on the same frames.
3. Offset mode employs the same basic 'observe and subtract' method to remove the sky-background as nod mode, but is typically favoured for extended sources where it is not possible to observe a blank patch of sky and the source within the same 11 arcsec slit. As sky and source are observed in separate frames this observation mode has the highest fraction of time lost to overheads.

2.6 Recipes

SOXSPIPE borrows the informative concept of `recipes' employed by ESO's data reduction pipelines to define the modular components of the pipeline. These recipes can be strung together to create an end-to-end workflow that takes as input the raw and calibration frames from the instrument and telescope and processes them all the way through to fully reduced, calibrated, ESO Phase III compliant science products.

2.6.1 Standard Calibrations

2.6.1.1 `soxs_mbias`

A zero-second exposure will contain only read-noise and ~half of pixels within this Gaussian distribution centred around zero count will always contain negative flux. To avoid negative counts an offset *bias* voltage is applied at the amplifier stage so that even when no photons are detected the A/D converter will always register a positive value. This bias-voltage offset must be accounted for in the data reduction process.

The purpose of the `soxs_mbias` recipe is to provide a master-bias frame that can be subtracted from science/calibration frames to remove the contribution of pixel counts resulting from the bias-voltage.

Input

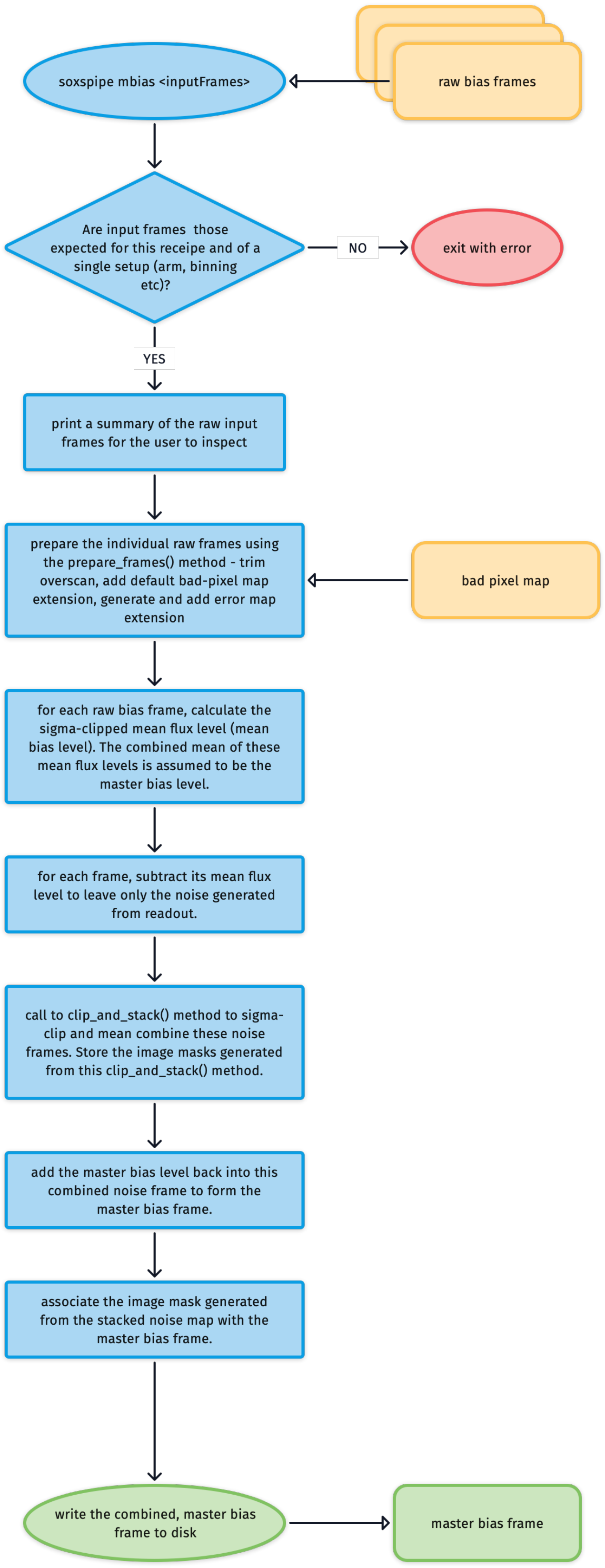
Data Type	Content	Related OB
images	raw bias frames (UV-VIS/AC exposures with exptime = 0)	`SOXS_img_cal_Bias`, `SOXS_gen_cal_VISBias`

Parameters

Parameter	Description	Type	Entry Point	Related Util
-clipping-sigma	number of from the median *frame* flux beyond which pixel is added to the bad-pixel mask	float	settings file	`clip_and_stack`
-iteration-count	number of sigma-clipping iterations to perform when added pixels to the bad-pixel mask	int	settings file	`clip_and_stack`
-clipping-sigma	number of deviations from the median *pixel* flux beyond which pixel is excluded from stack	float	settings file	`clip_and_stack`
-clipping-iterations	number of -clipping iterations to perform before stacking	float	settings file	`clip_and_stack`

Method

The purpose of the `soxs_mbias` recipe is to stack raw bias-frames together (using the `clip_and_stack` utility) into master-bias frames and in the process clipping rogue pixels from the individual raw frames and reducing the read-noise contribution.


soxspipe mbias <inputFrames>
raw bias frames
Are input frames those expected for this receipe and of a single setup (arm, binning etc)?
NO
exit with error
YES
print a summary of the raw input frames for the user to inspect
prepare the individual raw frames using the prepare_frames() method - trim overscan, add default bad-pixel map extension, generate and add error map extension
bad pixel map
for each raw bias frame, calculate the sigma-clipped mean flux level (mean bias level). The combined mean of these mean flux levels is assumed to be the master bias level.
for each frame, subtract its mean flux level to leave only the noise generated from readout.
call to clip_and_stack() method to sigma-clip and mean combine these noise frames. Store the image masks generated from this clip_and_stack() method.
add the master bias level back into this combined noise frame to form the master bias frame.
associate the image mask generated from the stacked noise map with the master bias frame.
write the combined, master bias frame to disk
master bias frame

Output

Data Type	Content
image	Master bias frame (frame containing typical bias-voltage applied to the detector)

QC Metrics

Metric	Description
	. . .

Recipe API

class soxs_mbias(*log*, *settings=False*, *inputFrames=[]*, *verbose=False*, *overwrite=False*)

The `soxs_mbias` *recipe is used to generate a master-bias frame from a set of input raw bias frames. The recipe is used only for the UV-VIS arm as NIR frames have bias (and dark current) removed by subtracting an off-frame of equal expsoure length.*

Key Arguments

- `log` – logger
- `settings` – the settings dictionary
- `inputFrames` – input fits frames. Can be a directory, a set-of-files (SOF) file or a list of fits frame paths.
- `verbose` – verbose. True or False. Default *False*
- `overwrite` – overwrite the prodcut file if it already exists. Default *False*

Usage

```
from soxspipe.recipes import soxs_mbias
mbiasFrame = soxs_mbias(
    log=log,
    settings=settings,
    inputFrames=fileList
).produce_product()
```

Todo:

- add a tutorial about `soxs_mbias` to documentation

verify_input_frames()

verify the input frame match those required by the soxs_mbias recipe

If the fits files conform to required input for the recipe everything will pass silently, otherwise an exception shall be raised.

produce_product()

generate a master bias frame

Return:

- `productPath` – the path to the master bias frame

qc_bias_structure(*combined_bias_mean*)

calculate the structure of the bias

Key Arguments:

- `combined_bias_mean` – the mbias frame

Return:

- `structx` – slope of BIAS in X direction
- `structx` – slope of BIAS in Y direction

Usage:

```
structx, structy = self.qc_bias_structure(combined_bias_mean)
```

qc_periodic_pattern_noise(*frames*)

calculate the periodic pattern noise based on the raw input bias frames

A 2D FFT is applied to each of the raw bias frames and the standard deviation and median absolute deviation calcualted for each result. The maximum std/mad is then added as the ppnmax QC in the master bias frame header.

Key Arguments:

- `frames` – the raw bias frames (imageFileCollection)

Return:

```
- ``ppnmax``
```

Usage:

```
self.qc_periodic_pattern_noise(frames=self.inputFrames)
```

2.6.1.2 `soxs_mdark`

Every raw CCD image contains counts resulting from a 'dark current', electrons released due to the thermal effects in the CCD material. For both the UVB-VIS (< 0.00012 e$^-$/s/pixel) and NIR detectors (< 0.005 e$^-$/s/pixel) the dark-current is almost negligible. Not all pixels will have the same dark-current, some will have a high than typical current. These are so-called 'hot-pixels' and it's important that these are identified and recorded (using the `create_noise_map` utility).

The purpose of the `soxs_mdark` recipe is to generate a master-dark frame used to remove flux attributed to the dark-current from other frames.

Input

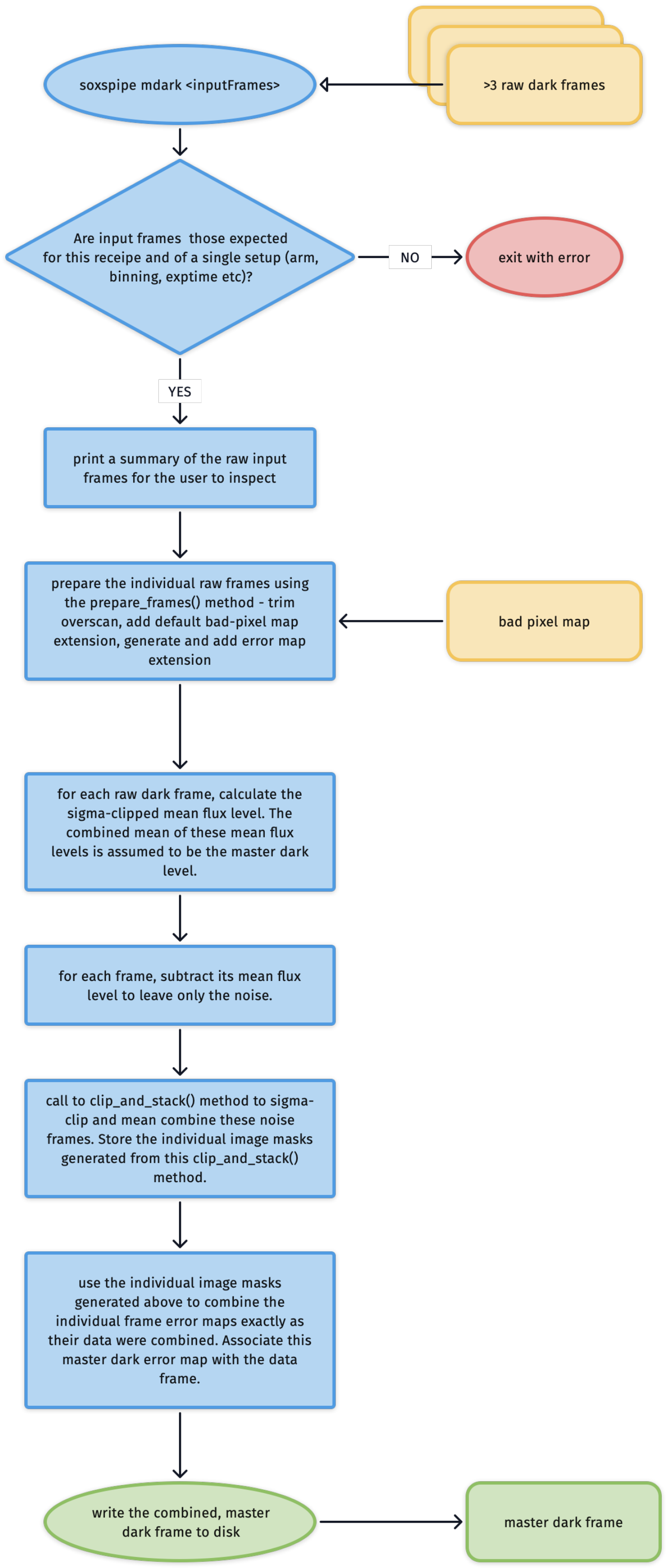
Data Type	Content	Related OB
images	raw dark frames (exposures with identical exposure time and detectors readout parameters)	`SOXS_gen_cal_VISDark`, `SOXS_gen_cal_NIRDark`, `SOXS_img_cal_Dark`

Parameters

Parameter	Description	Type	Entry Point	Related Util
-clipping-sigma	number of from the median *frame* flux beyond which pixel is added to the bad-pixel mask	float	settings file	`clip_and_stack`
-iteration-count	number of sigma-clipping iterations to perform when added pixels to the bad-pixel mask	int	settings file	`clip_and_stack`
-clipping-sigma	number of deviations from the median *pixel* flux beyond which pixel is excluded from stack	float	settings file	`clip_and_stack`
-clipping-iterations	number of -clipping iterations to perform before stacking	float	settings file	`clip_and_stack`

Method

Stack raw dark-frames together (using the `clip_and_stack` utility) into master-dark frames and in the process clipping rogue pixels from the individual raw frames and reducing the read-noise contribution.


soxspipe mdark <inputFrames>
>3 raw dark frames
Are input frames those expected for this receipe and of a single setup (arm, binning, exptime etc)?
NO
exit with error
YES
print a summary of the raw input frames for the user to inspect
prepare the individual raw frames using the prepare_frames() method - trim overscan, add default bad-pixel map extension, generate and add error map extension
bad pixel map
for each raw dark frame, calculate the sigma-clipped mean flux level. The combined mean of these mean flux levels is assumed to be the master dark level.
for each frame, subtract its mean flux level to leave only the noise.
call to clip_and_stack() method to sigma-clip and mean combine these noise frames. Store the individual image masks generated from this clip_and_stack() method.
use the individual image masks generated above to combine the individual frame error maps exactly as their data were combined. Associate this master dark error map with the data frame.
write the combined, master dark frame to disk
master dark frame

Output

Data Type	Content
dark frame	frame containing typical dark-current flux accumulated over the exposure time of the input frames

QC Metrics

Metric	Description
	…

Recipe API

class soxs_mdark(*log*, *settings=False*, *inputFrames=[]*, *verbose=False*, *overwrite=False*)

The soxs_mdark recipe

Key Arguments

- `log` – logger
- `settings` – the settings dictionary
- `inputFrames` – input fits frames. Can be a directory, a set-of-files (SOF) file or a list of fits frame paths.
- `verbose` – verbose. True or False. Default *False*
- `overwrite` – overwrite the prodcut file if it already exists. Default *False*

Usage

```
from soxspipe.recipes import soxs_mdark
mdarkFrame = soxs_mdark(
    log=log,
    settings=settings,
    inputFrames=fileList
)..produce_product()
```

Todo:

- add a tutorial about `soxs_mdark` to documentation

verify_input_frames()

verify input frame match those required by the soxs_mdark recipe

If the fits files conform to required input for the recipe everything will pass silently, otherwise an exception shall be raised.

produce_product()

generate a master dark frame

Return:

- `productPath` – the path to master dark frame

2.6.1.3 `soxs_mflat`

The purpose of the `soxs_mflat` recipe is to create a single normalised master-flat frame used to correct for non-uniformity in response to light across the detector plain.

Sources of this non-uniformity include varying pixel sensitivities, obstructions in the optical path (e.g. dust or pollen grains), vignetting at the edges of the detector. A flat-frame is ideally an image taken where the illumination is uniform across the light collecting pixels of the detector. This evenly exposed image can be used to identify irregularities in the response in the detector.

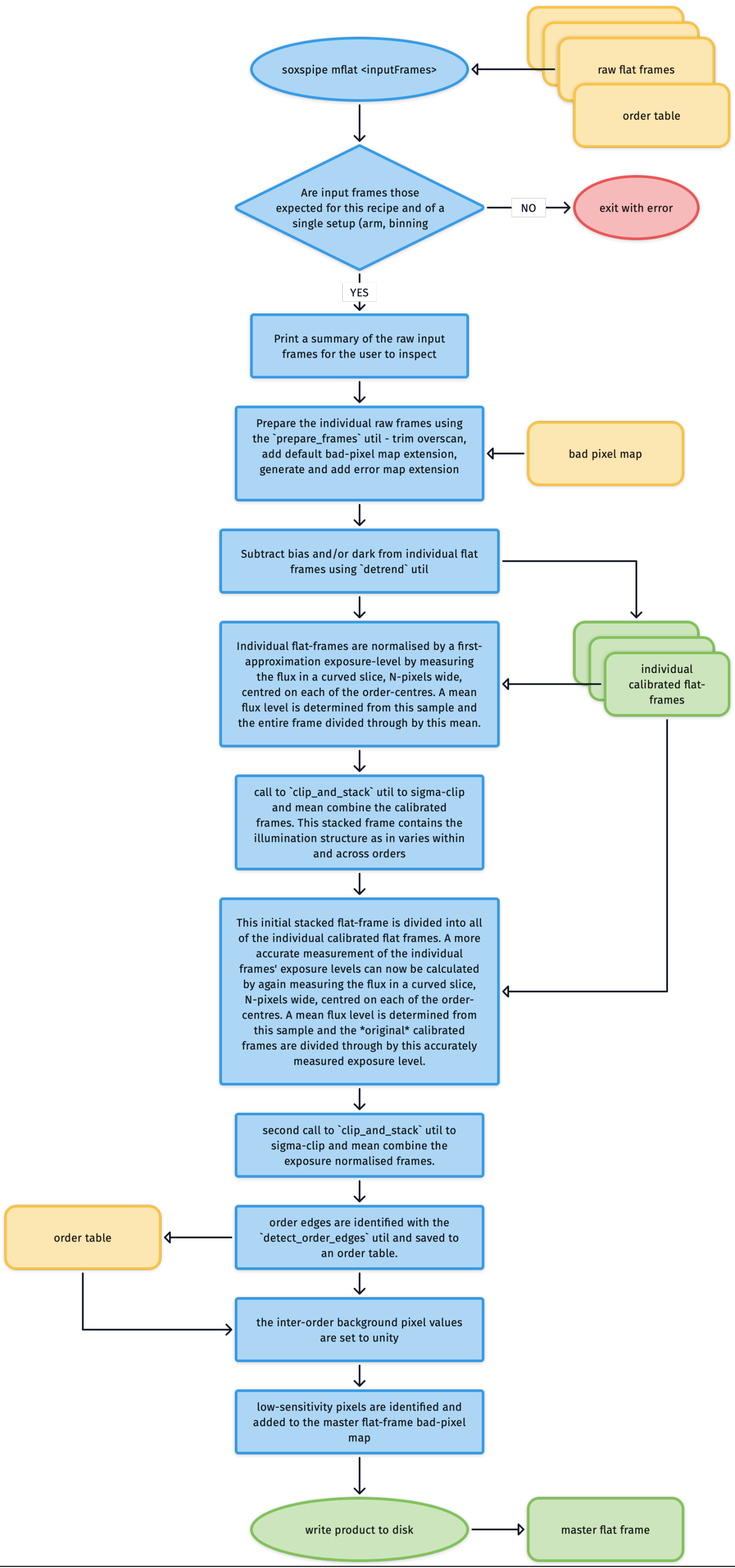
Input

Data Type	Content	Related OB
images	raw flats frames (exposures with identical exposure time and detectors readout parameters). UV-VIS requires separate sets D-Lamp and QTH-Lamp flats.	`SOXS_slt_cal_NIRLampFlat`, `SOXS_slt_cal_NIRLampFlatAtt`, `SOXS_slt_cal_VISLampFlat`, `SOXS_slt_cal_VISLampFlatAtt`
Image	Master Bias Frame (UV-VIS only)	-
Table	order table containing coefficients to the polynomial fits describing the order centre locations. UV-VIS requires separate tables for D-Lamp and QTH-Lamp.	

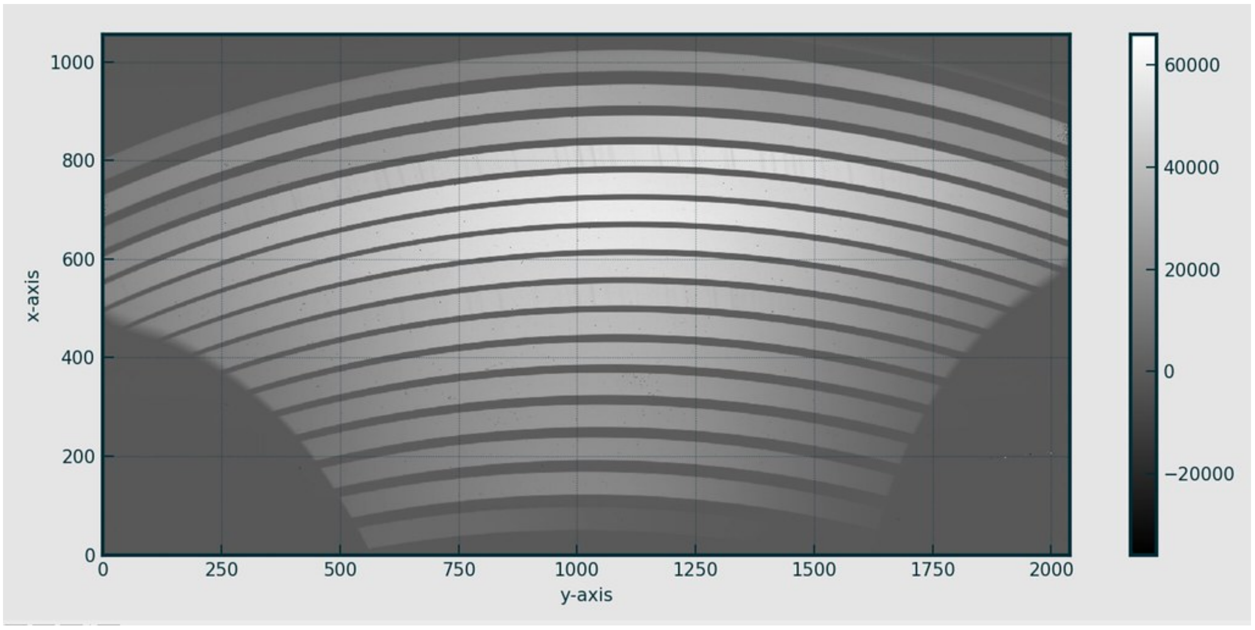
Parameters

Parameter	Description	Type	Entry Point	Related Util
-clipping-sigma	number of from the median *frame* flux beyond which pixel is added to the bad-pixel mask	float	settings file	clip_and_stack
-iteration-count	number of sigma-clipping iterations to perform when added pixels to the bad-pixel mask	int	settings file	clip_and_stack
-clipping-sigma	number of deviations from the median *pixel* flux beyond which pixel is excluded from stack	float	settings file	clip_and_stack
-clipping-iterations	number of -clipping iterations to perform before stacking	float	settings file	clip_and_stack
-order-window	the width of the slice to cut along the centre of each order when determining mean exposure level	int	settings file	-
-length-for-edge-detection	length of image slice to take across orders when detecting edges	int	settings file	detect_order_edges
-width-for-edge-detection	width of image slice to take across orders when detecting edges	int	settings file	detect_order_edges
-percentage-threshold-for-edge-detection	minimum value flux can drop to as percentage of central flux and be counted as an order edge	int	settings file	detect_order_edges
-percentage-threshold-for-edge-detection	maximum value flux can claim to as percentage of central flux and be counted as an order edge	int	settings file	detect_order_edges
-axis-deg	degree of dispersion axis component of polynomial fit to order edges	int	settings file	detect_order_edges
-deg	degree of order component of polynomial fit to order edges	int	settings file	detect_order_edges
-fitting-residual-clipping-sigma	number of deviations from the median fit residual beyond which individual data points are removed when iterating towards a fit of order edges	int	settings file	detect_order_edges
-clipping-iteration-limit	number of sigma-clipping iterations to perform before settings on a polynomial fit for the order edges	int	settings file	detect_order_edges
-sensitivity-clipping-sigma	number of deviations below the median flux of a master-flat frame beyond which a pixel is added to the bad-pixel mask	int	settings file	-

Method


raw flat frames
order table
soxspipe mflat <inputFrames>
Are input frames those expected for this recipe and of a single setup (arm, binning
NO
exit with error
YES
Print a summary of the raw input frames for the user to inspect
Prepare the individual raw frames using the `prepare_frames` util - trim overscan, add default bad-pixel map extension, generate and add error map extension
bad pixel map
Subtract bias and/or dark from individual flat frames using `detrend` util
individual calibrated flat-frames
Individual flat-frames are normalised by a first-approximation exposure-level by measuring the flux in a curved slice, N-pixels wide, centred on each of the order-centres. A mean flux level is determined from this sample and the entire frame divided through by this mean.
call to `clip_and_stack` util to sigma-clip and mean combine the calibrated frames. This stacked frame contains the illumination structure as in varies within and across orders
This initial stacked flat-frame is divided into all of the individual calibrated flat frames. A more accurate measurement of the individual frames' exposure levels can now be calculated by again measuring the flux in a curved slice, N-pixels wide, centred on each of the order-centres. A mean flux level is determined from this sample and the *original* calibrated frames are divided through by this accurately measured exposure level.
second call to `clip_and_stack` util to sigma-clip and mean combine the exposure normalised frames.
order edges are identified with the `detect_order_edges` util and saved to an order table.
order table
the inter-order background pixel values are set to unity
low-sensitivity pixels are identified and added to the master flat-frame bad-pixel map
write product to disk
master flat frame

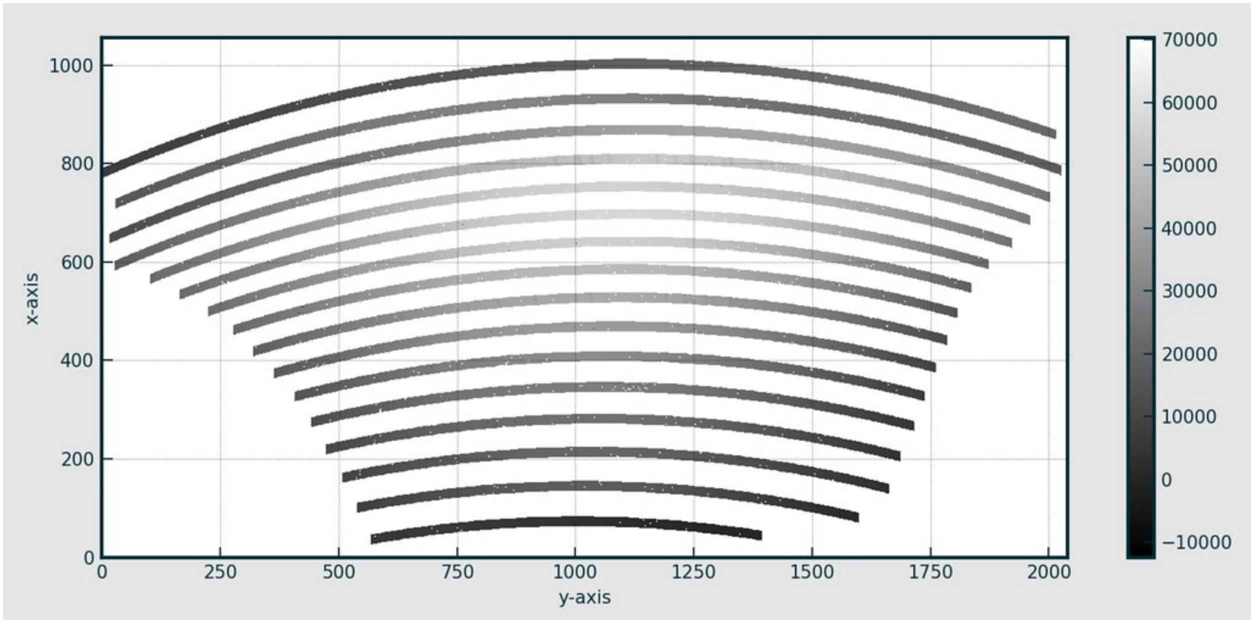
The individual flat field frames need to have bias and dark signatures removed before they are combined. This is achieved with the `detrend` utility. Here is an example of one such calibrated flat frame:



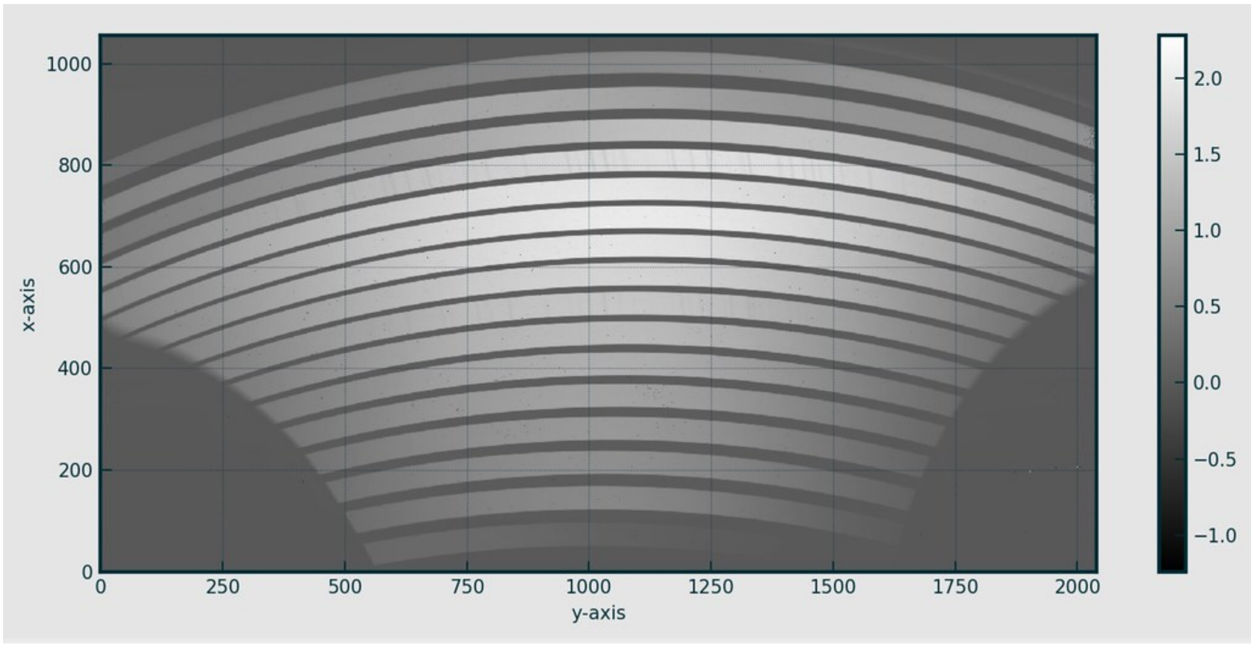
Normalising Exposure Levels in Individual Flat Frames

Once calibrated, exposure-levels in the individual flat frames need to be normalised as *total* illumination will vary from frame-to-frame. The individual frame exposure levels are calculated in two stages.

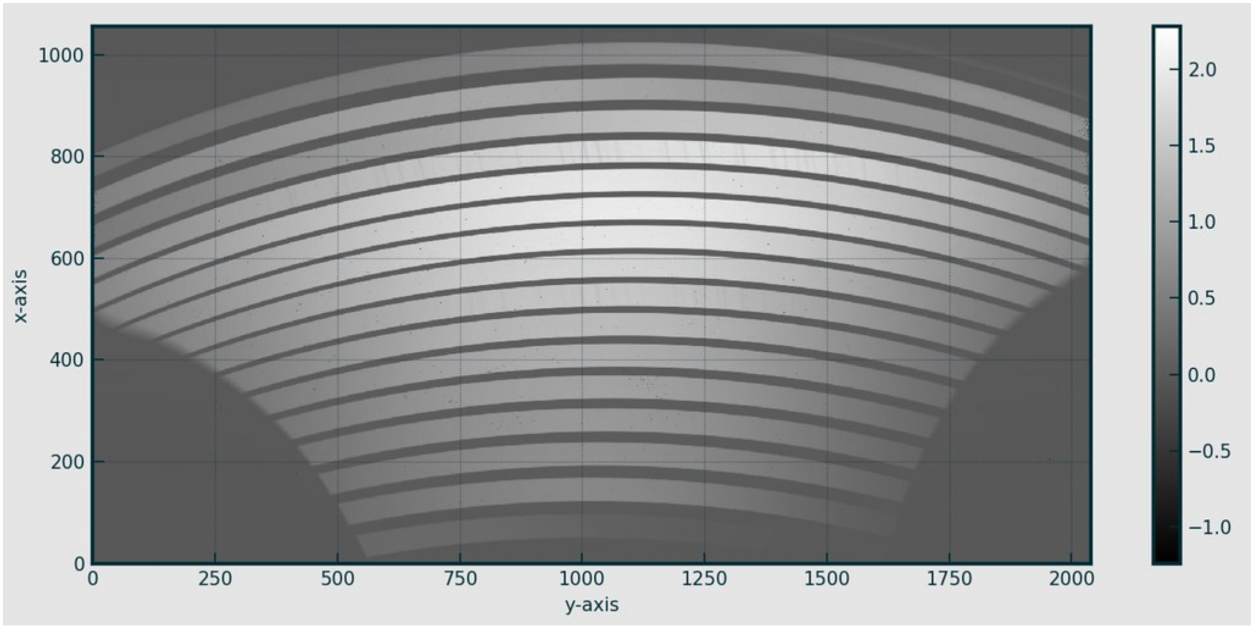
In the first stage the mean inner-order pixel-value across the frame is used as a *first approximation* of an individual frame's exposure level. To calculate this mean value, the order locations are used to identify a curved slice N-pixels wide centred on each of the order-centres and bad pixels are masked (see image below). The collected inner order pixel values are then sigma-clipped to excluded out-lying values and a mean value calculated.



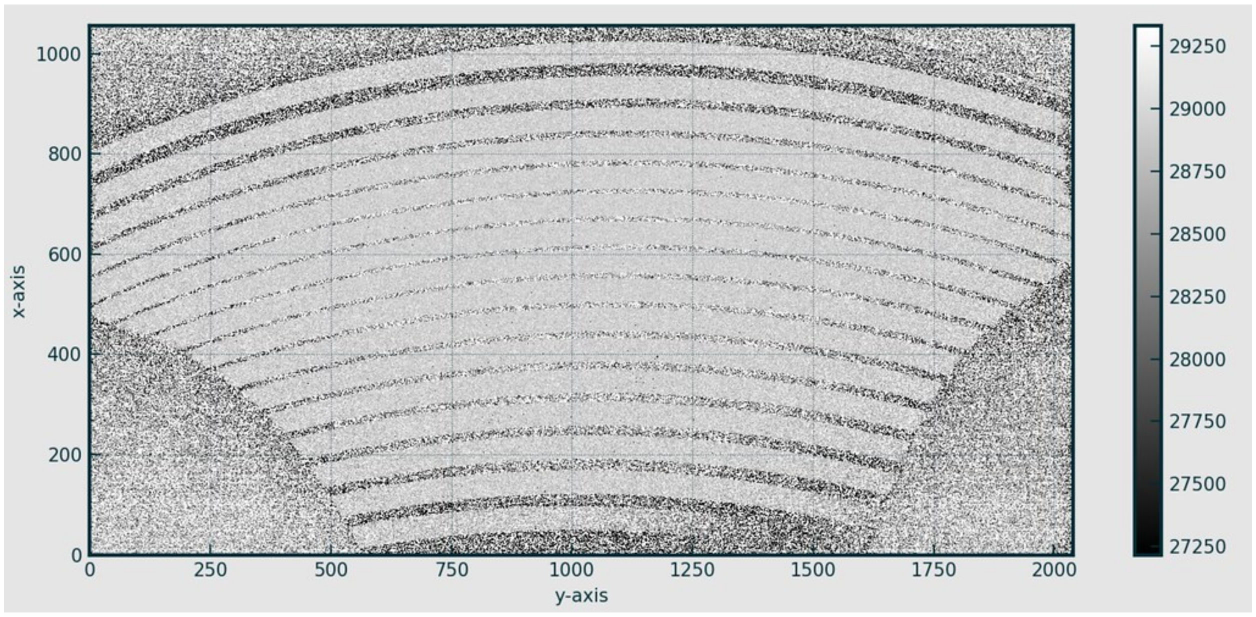
Individual frames are then divided through by their mean inner-order pixel value in this first attempt to normalise the exposure-levels of the frames.



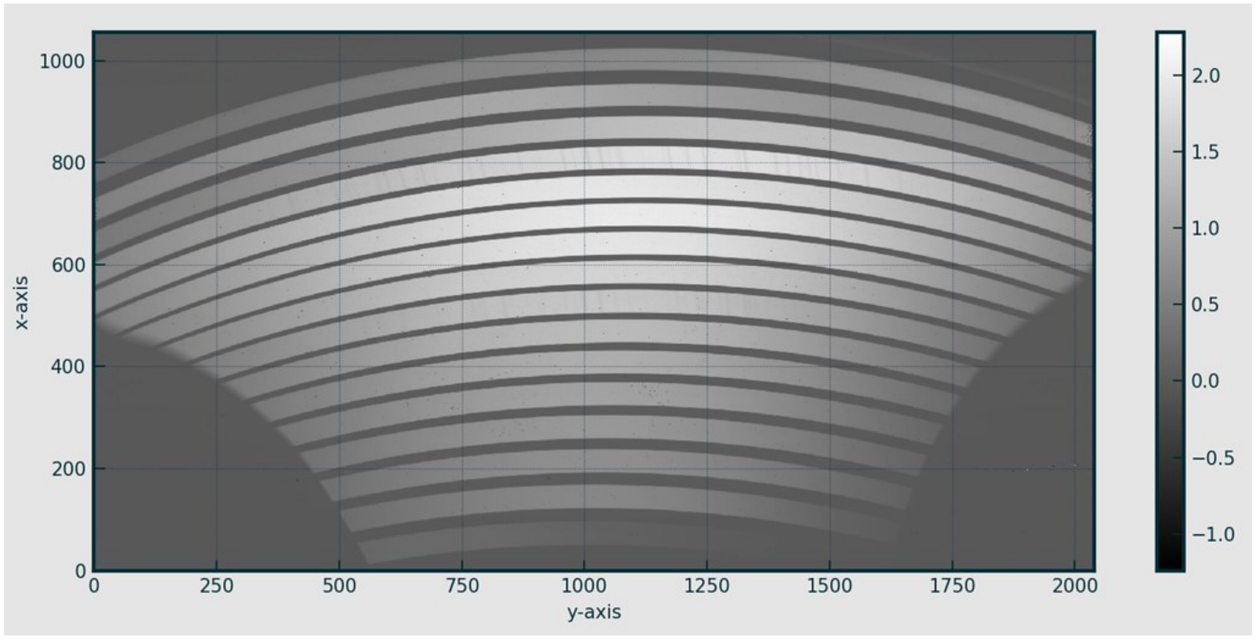
The normalised flat-frames are then combined using the `clip_and_stack` utility into a first-pass master-flat frame:



The second stage then is to divide each original dark and bias subtracted flat frame by this first-pass master flat (see example below). This removes the *typical* cross-plane illumination and so now the mean inner-order pixel-value across the frame will give a much better estimate of each frame's intrinsic exposure level.

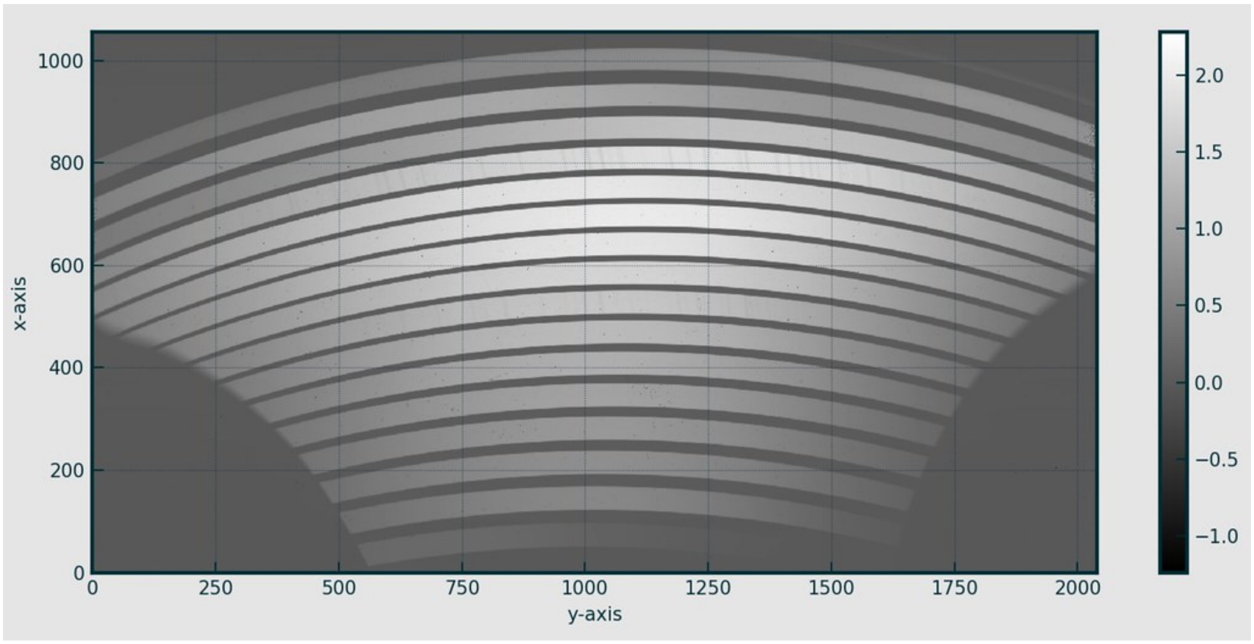


The mean inner-order pixel-value is calculated again on this frame and the original dark and bias subtracted flat is re-normalised by divided through by this accurate measurement of its intrinsic exposure level.



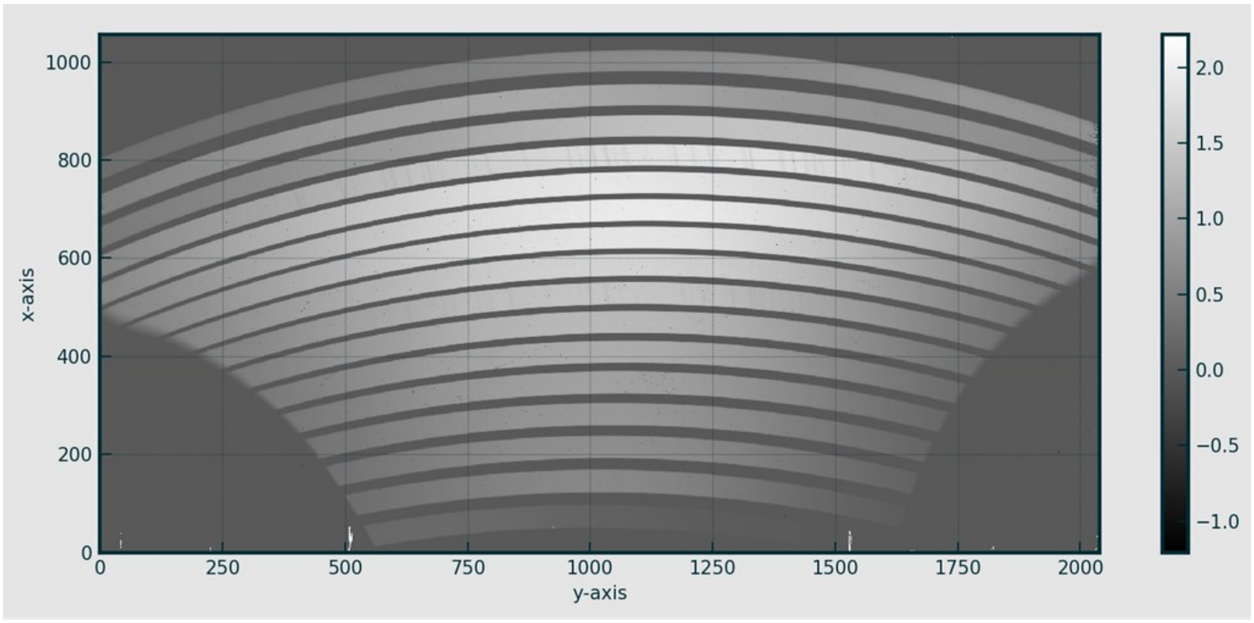
Building a Final Master-Flat Frame

These re-normalised flats are then combined for a second time into a master-flat frame.

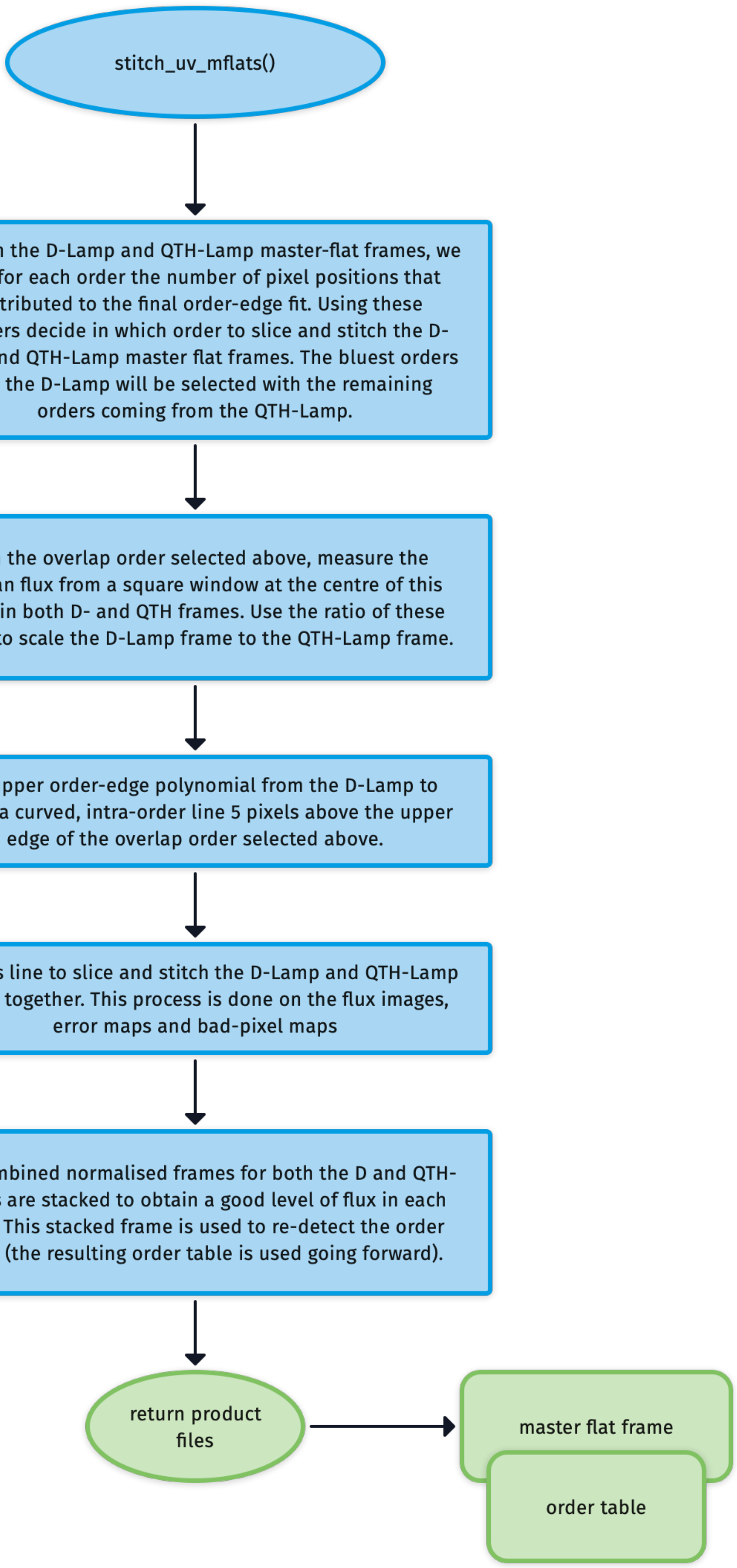


Finally order edges are located with the `detect_order_edges` utility and the inter-order area pixel value are set to 1.

Low-sensitivity pixels are flagged and added to the bad-pixel map and a final master-flat frame written to file.



UV Master Flat Frame Stitching


stitch_uv_mflats()
For both the D-Lamp and QTH-Lamp master-flat frames, we have for each order the number of pixel positions that contributed to the final order-edge fit. Using these numbers decide in which order to slice and stitch the D-Lamp and QTH-Lamp master flat frames. The bluest orders from the D-Lamp will be selected with the remaining orders coming from the QTH-Lamp.
With the overlap order selected above, measure the median flux from a square window at the centre of this order in both D- and QTH frames. Use the ratio of these fluxes to scale the D-Lamp frame to the QTH-Lamp frame.
Use upper order-edge polynomial from the D-Lamp to define a curved, intra-order line 5 pixels above the upper edge of the overlap order selected above.
Use this line to slice and stitch the D-Lamp and QTH-Lamp orders together. This process is done on the flux images, error maps and bad-pixel maps
The combined normalised frames for both the D and QTH-Lamps are stacked to obtain a good level of flux in each order. This stacked frame is used to re-detect the order edges (the resulting order table is used going forward).
return product files
master flat frame
order table

As the UV-VIS uses a combination of D-Lamp and QTH-Lamp flat sets, a further step is required to stitch the best orders from each of these master-flats together into a dual lamp master-flat.

For both the D-Lamp and QTH-Lamp master-flat frames, we have for each order the number of pixel positions that contributed to the final order-edge fit. We use these numbers to decide which orders to slice and stitch from the D-Lamp to the QTH-Lamp master flat frame.

With a crossover order now selected, the median flux from a square window at the centre of this order in both D- and QTH frames is measured. Using the ratio of these fluxes the D-Lamp frame is scaled to the QTH-Lamp frame.

From the upper order-edge polynomial for the D-Lamp we define a curved, intra-order line 5 pixels above the upper edge of the crossover order selected previously. This line is used to slice and stitch the D-Lamp and QTH-Lamp orders together. This process is done on the flux images, error maps and bad-pixel maps. Typically the bluest orders from the D-Lamp will be selected with the remaining orders coming from the QTH-Lamp.

Finally, the combined normalised frames for both the D and QTH-Lamps are stacked to obtain a good level of flux in each order. This stacked frame is used to re-detect the order edges (the resulting order table is used going forward).

Output

Data Type	Content
flat frame	frame used correct for non-uniformity in response to light across the detector plain (including blaze)

QC Metrics

Metric	Description
	...

Recipe API

class soxs_mflat(*log*, *settings=False*, *inputFrames=[]*, *verbose=False*, *overwrite=False*)

The soxs_mflat recipe

Key Arguments

- `log` – logger
- `settings` – the settings dictionary
- `inputFrames` – input fits frames. Can be a directory, a set-of-files (SOF) file or a list of fits frame paths.
- `verbose` – verbose. True or False. Default *False*
- `overwrite` – overwrite the prodcut file if it already exists. Default *False*

Usage

```
from soxspipe.recipes import soxs_mflat
recipe = soxs_mflat(
    log=log,
    settings=settings,
    inputFrames=fileList
)
mflatFrame = recipe.produce_product()
```

Todo:

- add a tutorial about `soxs_mflat` to documentation

---

verify_input_frames()
verify the input frames match those required by the soxs_mflat recipe

If the fits files conform to required input for the recipe everything will pass silently, otherwise an exception will be raised.

produce_product()
generate the master flat frames updated order location table (with egde detection)

Return:

- `productPath` – the path to the master flat frame

calibrate_frame_set()
given all of the input data calibrate the frames by subtracting bias and/or dark

Return:

- `calibratedFlats` – the calibrated frames

normalise_flats(*inputFlats*, *orderTablePath*, *firstPassMasterFlat=False*, *lamp=''*)
determine the median exposure for each flat frame and normalise the flux to that level

Key Arguments:

- `inputFlats` – the input flat field frames
- `orderTablePath` – path to the order table
- `firstPassMasterFlat` – the first pass of the master flat. Default *False*

```
- `lamp` -- a lamp tag for QL plots
```

Return:

- `normalisedFrames` – the normalised flat-field frames (CCDData array)

mask_low_sens_pixels(*frame*, *orderTablePath*, *returnMedianOrderFlux=False*, *writeQC=True*)
add low-sensitivity pixels to bad-pixel mask

Key Arguments:

- `frame` – the frame to work on
- `orderTablePath` – path to the order table
- `returnMedianOrderFlux` – return a table of the median order fluxes. Default *False*.
- `writeQC` – add the QCs to the QC table?

Return:

- `frame` – with BPM updated with low-sensitivity pixels
- `medianOrderFluxDF` – data-frame of the median order fluxes (if `returnMedianOrderFlux` is True)

stitch_uv_mflats(*medianOrderFluxDF*, *orderTablePath*)
return a master UV-VIS flat frame after slicing and stitch the UV-VIS D-Lamp and QTH-Lamp flat frames

Key Arguments:

- `medianOrderFluxDF` – data frame containing median order fluxes for D and QTH frames
- `orderTablePath` – the original order table paths from order-centre tracing

Return:

- `stitchedFlat` – the stitch D and QTH-Lamp master flat frame

Usage:

```
mflat = self.stitch_uv_mflats(medianOrderFluxDF)
```

find_uvb_overlap_order_and_scale(*dcalibratedFlats*, *qcalibratedFlats*)

find uvb order where both lamps produce a similar flux. This is the order at which the 2 lamp flats will be scaled and stitched together

Key Arguments:

- `qcalibratedFlats` – the QTH lamp calibration flats.
- `dcalibratedFlats` – D2 lamp calibration flats

Return:

- `order` – the order number where the lamp fluxes are similar

Usage:

```
overlapOrder = self.find_uvb_overlap_order_and_
↪scale(dcalibratedFlats=dcalibratedFlats, qcalibratedFlats=qcalibratedFlats)
```

2.6.2 Dispersion and Spatial Solutions

There is a strong curvature in the traces of the NIR orders and spectral-lines do not run perpendicular to the dispersion direction, but are highly tilted. Therefore wavelength cannot be expressed as simply a function of pixel position, but instead detector pixel positions (X, Y) much be mapped as a function of:

1. wavelength λ
2. order number n, and
3. slit position s

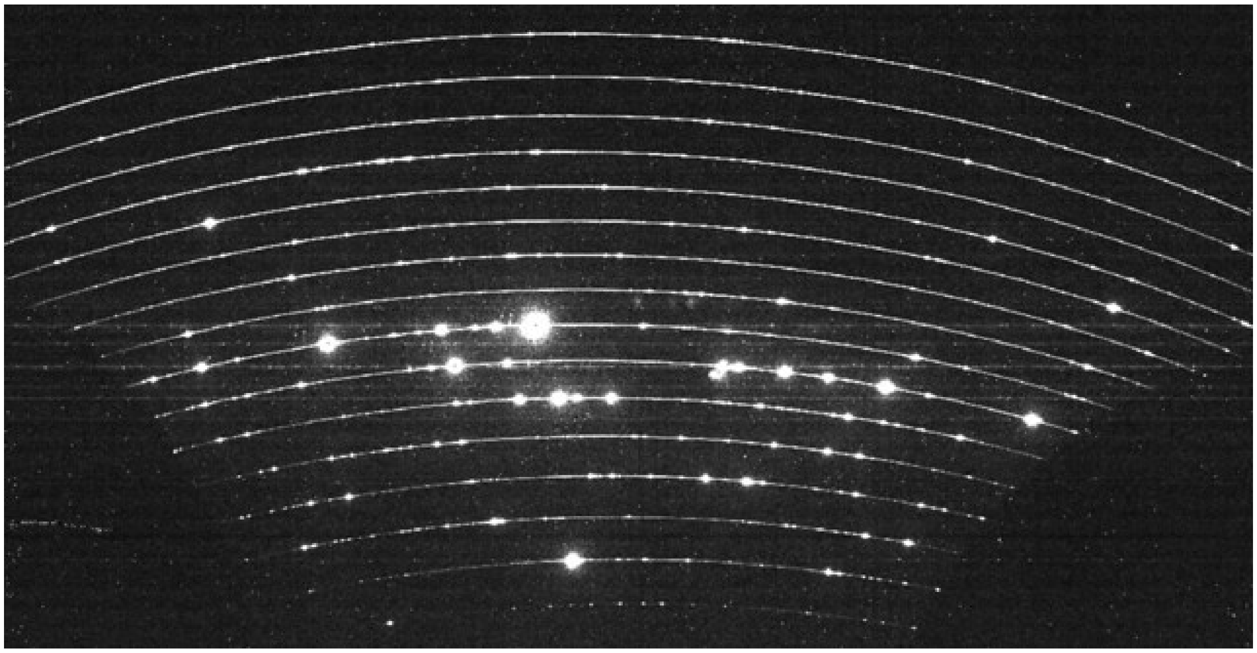
This 2D mapping function is determined incrementally via the `soxs_disp_solution`, `soxs_order_centres` and `soxs_spatial_solution` recipes. The `soxs_straighten` recipe can then be used to transform spectral images from detector pixel-space to wavelength and slit-position space

2.6.2.1 `soxs_disp_solution`

The purpose of the `soxs_disp_solution` is to use a single-pinhole arc-lamp frame (example image above) to generate a first guess dispersion solution.

Input

As input this recipes accepts the Pinhole Map file.



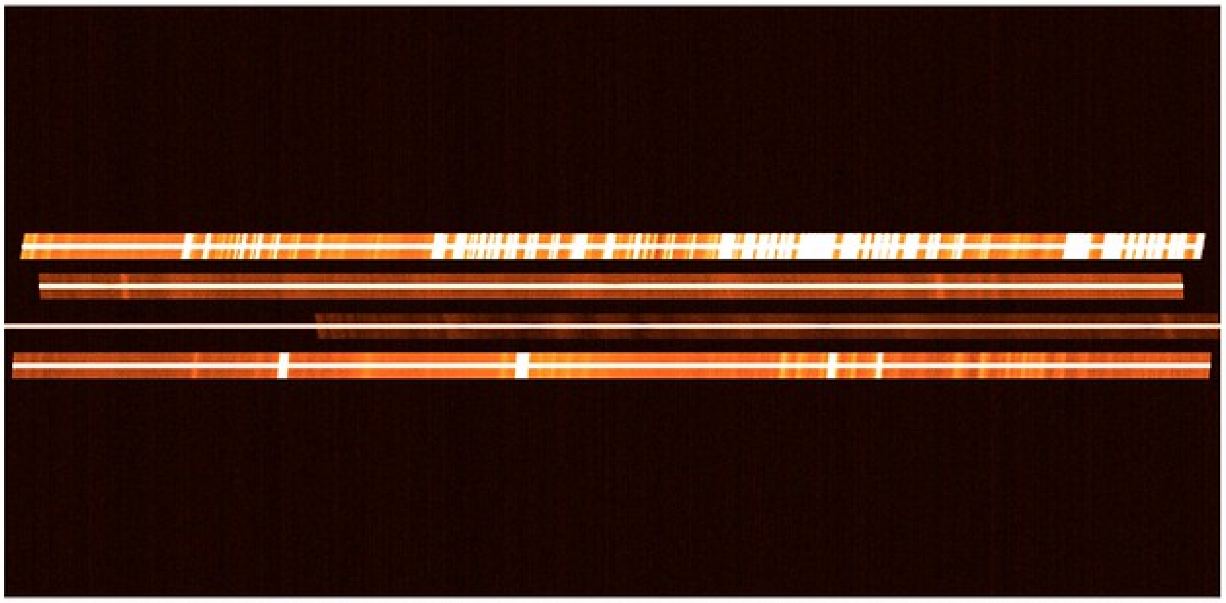


Figure 4. Full UV-VIS arm with a bright (V=8) object and different tilts on the slit for each pseudo-order

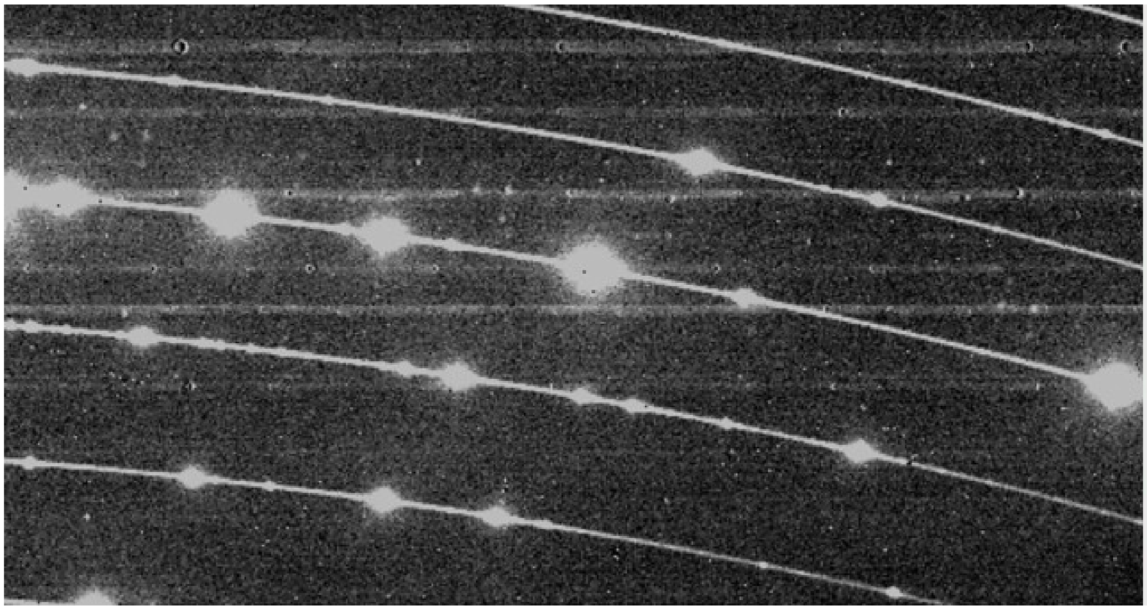
Data Type	Content	Related OB
Image	Arc Lamp through single pinhole mask	SOXS_slt_cal_VISArcsPinhole, SOXS_slt_cal_NIRArcsPinhole
Image	Master Dark Frame (VIS only)	-
Image	Master Bias Frame (VIS only)	-
Image	Dark frame (Lamp-Off) of equal exposure length as single pinhole frame (Lamp-On) (NIR only)	SOXS_slt_cal_NIRArcsPinhole
File	Pinhole Map	

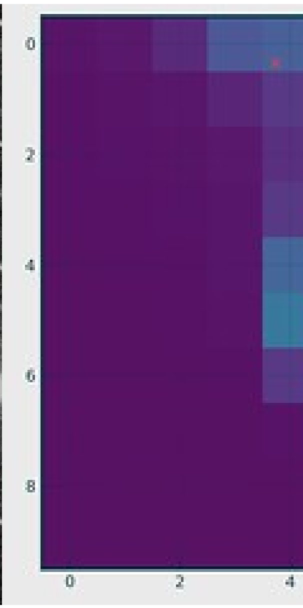
Parameters

Parameter	Description	Type	Entry Point	Related Util
-window-size	the side-length (in pixels) of the square window used to search for arc-line detection	int	settings file	create_dispersion_map
-deg	the order of polynomial used to fit spectral-orders of detected arc-lines	int	settings file	create_dispersion_map
-deg	the order of polynomial used to fit wavelengths of detected arc-lines	int	settings file	create_dispersion_map
-fitting-residual-clipping-sigma	sigma distance limit, where distance is the difference between the detected and polynomial fitted positions of an arc-line, outside of which to remove lines from the fit	float	settings file	create_dispersion_map
-clipping-iteration-limit	number of sigma-clipping iterations to perform before settings on a polynomial fit for the dispersion solution	int	settings file	create_dispersion_map

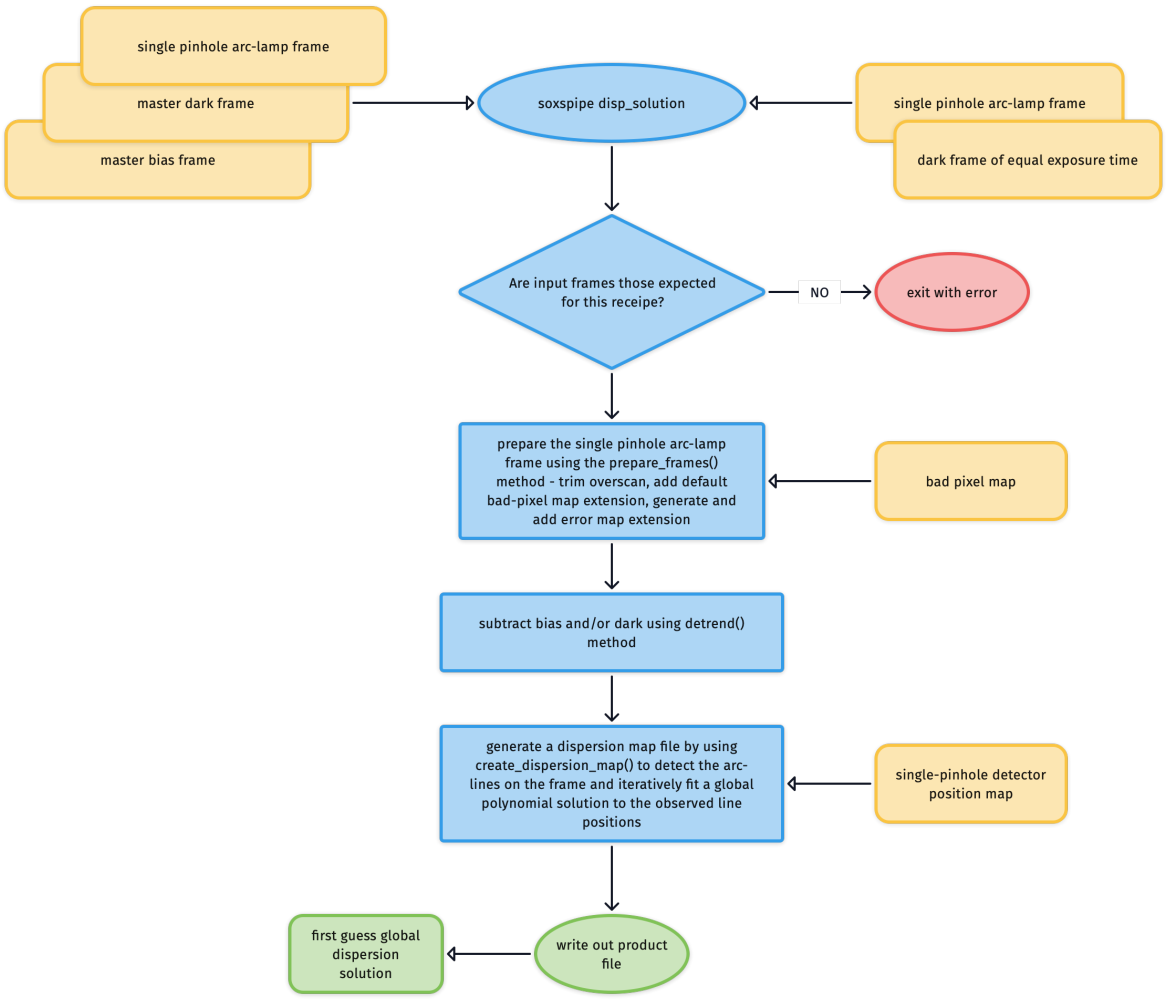
Method

After preparing and calibrating the single-pinhole arc-lamp frame (using the `detrend`), the `create_dispersion_map`) util is employed to detect and measure the positions of the arc lines on the frame. Below you can see the bright arc-lines outshining the traces of the order-centres and the detection of one of these lines during the `create_dispersion_map`) util.





Once the line positions have been measured, a dispersion solution is generated by iteratively fitting a global polynomial against the observed line-positions (see `create_dispersion_map`) for details). The final product is a Dispersion Map file.

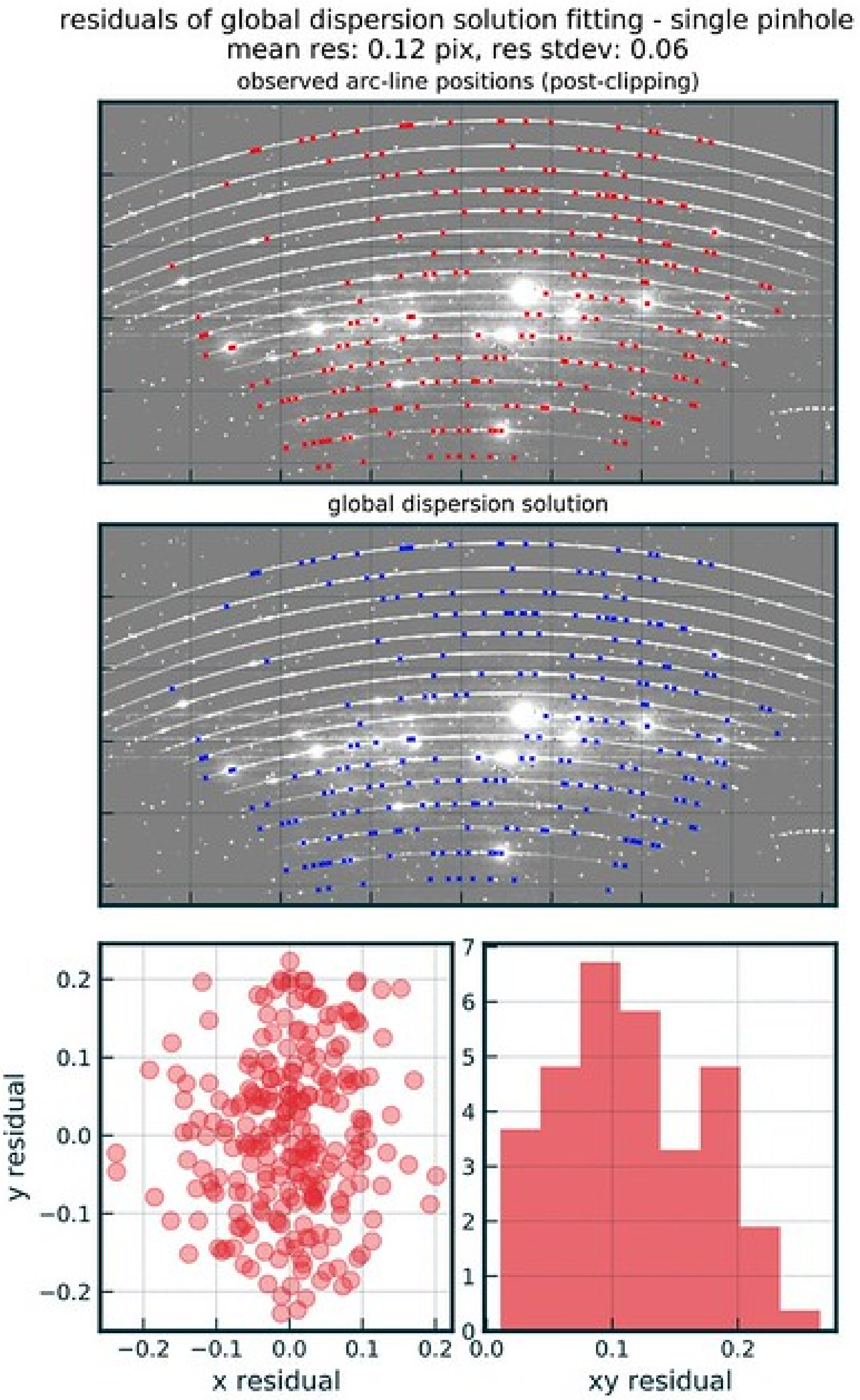

single pinhole arc-lamp frame
master dark frame
master bias frame
soxspipe disp_solution
single pinhole arc-lamp frame
dark frame of equal exposure time
Are input frames those expected for this receipe?
NO
exit with error
prepare the single pinhole arc-lamp frame using the prepare_frames() method - trim overscan, add default bad-pixel map extension, generate and add error map extension
bad pixel map
subtract bias and/or dark using detrend() method
generate a dispersion map file by using create_dispersion_map() to detect the arc-lines on the frame and iteratively fit a global polynomial solution to the observed line positions
single-pinhole detector position map
write out product file
first guess global dispersion solution

Output

Data Type	Content
file (subject to change)	First guess Dispersion Map

QC Metrics

The typical solution for the `soxs_disp_solution` recipe has sub-pixel residuals.


residuals of global dispersion solution fitting - single pinhole
mean res: 0.12 pix, res stdev: 0.06
observed arc-line positions (post-clipping)
global dispersion solution
y residual
x residual
xy residual
0.2
0.1
0.0
−0.1
−0.2
0
1
2
3
4
5
6
7

Metric	Description
	...

Recipe API

class soxs_disp_solution(*log*, *settings=False*, *inputFrames=[]*, *verbose=False*, *overwrite=False*)

generate a first approximation of the dispersion solution from single pinhole frames

Key Arguments

- `log` – logger
- `settings` – the settings dictionary
- `inputFrames` – input fits frames. Can be a directory, a set-of-files (SOF) file or a list of fits frame paths.
- `verbose` – verbose. True or False. Default *False*
- `overwrite` – overwrite the prodcut file if it already exists. Default *False*

Usage

```
from soxspipe.recipes import soxs_disp_solution
disp_map_path = soxs_disp_solution(
    log=log,
    settings=settings,
    inputFrames=sofPath
).produce_product()
```

Todo:

- add a tutorial about `soxs_disp_solution` to documentation

verify_input_frames()

verify input frames match those required by the ``soxs_disp_solution`` recipe

If the fits files conform to required input for the recipe everything will pass silently, otherwise an exception shall be raised.

produce_product()

generate a fisrt guess of the dispersion solution

Return:

- `productPath` – the path to the first guess dispersion map

2.6.2.2 `soxs_order_centres`

The purpose of the `soxs_order_centres` recipe is to find and fit the order centres with low-level polynomials.

Input

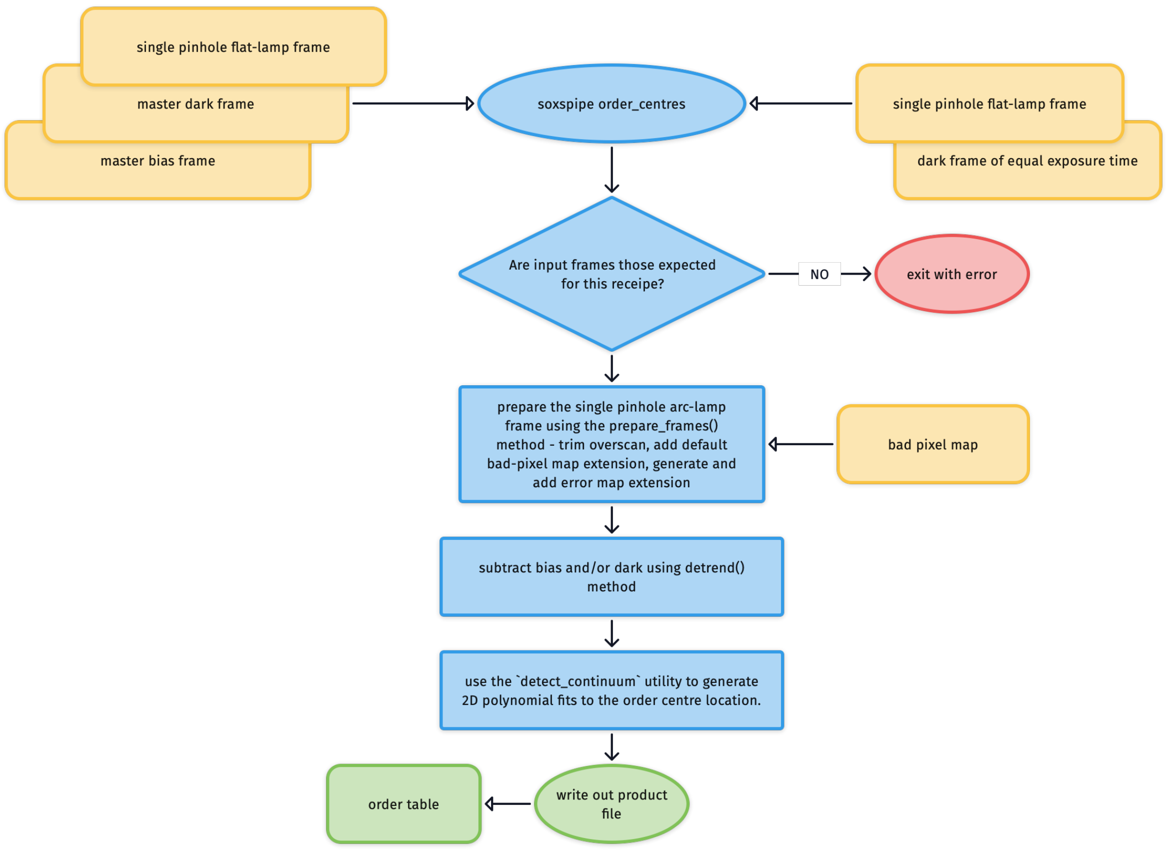
Data Type	Content	Related OB
Image	Flat lamp through a single-pinhole mask	`SOXS_slt_cal_VISLampFlatPinhole`, `SOXS_slt_cal_NIRLampFlatPinhole`
Image	Master Dark Frame (VIS only)	-
Image	Master Bias Frame (VIS only)	-
Image	Dark frame (Lamp-Off) of equal exposure length as single-pinhole frame (Lamp-On) (NIR only)	`SOXS_slt_cal_NIRLampFlatPinhole`
File	First guess dispersion solution	-

Parameters

Parameter	Description	Type	Entry Point	Related Util
-sample-count	number of times along the order in the dispersion direction to measure the order-centre trace	int	settings file	detect_continuum utility
-sigma-limit	minimum value a peak must be above the median value of pixel to be considered for order-trace fitting	int	settings file	detect_continuum utility
-axis-deg	degree of dispersion axis component of polynomal fit to order-centre traces		int	settings file
-deg	degree of order component of polynomal fit to order-centre traces		int	settings file
-fitting-residual-clipping-sigma	sigma distance limit, where distance is the difference between the detected and polynomial fitted positions of the order-trace, outside of which to remove lines from the fit	float	settings file	detect_continuum utility
-clipping-iteration-limit	number of sigma-clipping iterations to perform before settings on a polynomial fit for the order-centre traces	int	settings file	detect_continuum utility

Method

Once the single-pinhole flat-lamp frame has had the bias, dark and background subtracted it is passed to the detect_continuum utility to fit the order centres.

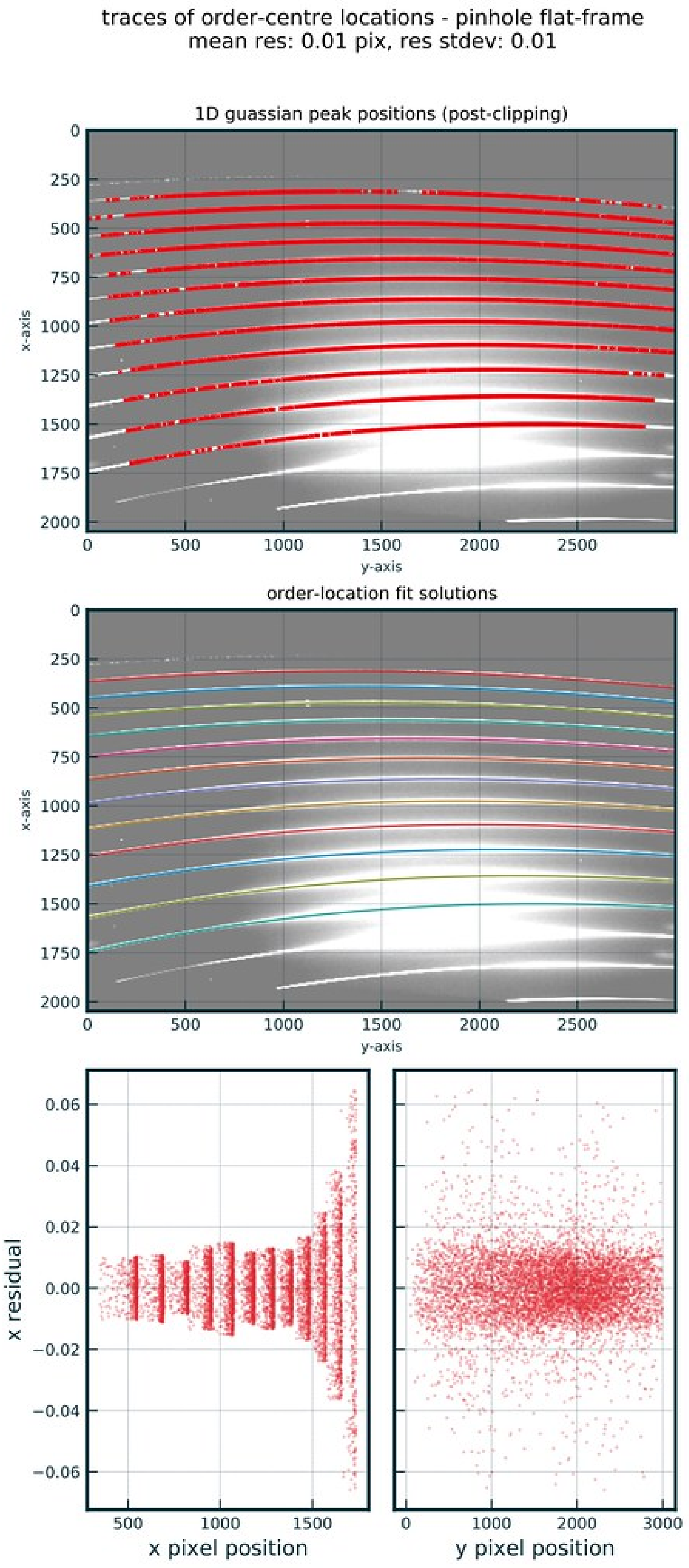

single pinhole flat-lamp frame
master dark frame
master bias frame
soxspipe order_centres
single pinhole flat-lamp frame
dark frame of equal exposure time
Are input frames those expected for this receipe?
NO
exit with error
prepare the single pinhole arc-lamp frame using the prepare_frames() method - trim overscan, add default bad-pixel map extension, generate and add error map extension
bad pixel map
subtract bias and/or dark using detrend() method
use the `detect_continuum` utility to generate 2D polynomial fits to the order centre location.
write out product file
order table

Output

Data Type	Content
File	order table containing coefficients to the polynomial fits describing the order centre locations

QC Metrics

Plots similar to the one below are generated after each execution of `soxs_order_centres`.


traces of order-centre locations - pinhole flat-frame
mean res: 0.01 pix, res stdev: 0.01
1D guassian peak positions (post-clipping)
x-axis
y-axis
order-location fit solutions
x-axis
y-axis
x residual
x pixel position
y pixel position

Metric	Description
	. . .

Recipe API

class soxs_order_centres(*log*, *settings=False*, *inputFrames=[]*, *verbose=False*, *overwrite=False*)

The soxs_order_centres recipe

Key Arguments

- `log` – logger
- `settings` – the settings dictionary
- `inputFrames` – input fits frames. Can be a directory, a set-of-files (SOF) file or a list of fits frame paths.
- `verbose` – verbose. True or False. Default *False*
- `overwrite` – overwrite the prodcut file if it already exists. Default *False*

Usage

```
from soxspipe.recipes import soxs_order_centres
order_table = soxs_order_centres(
    log=log,
    settings=settings,
    inputFrames=a["inputFrames"]
).produce_product()
```

Todo:

- add a tutorial about `soxs_order_centres` to documentation

verify_input_frames()

verify input frames match those required by the soxs_order_centres recipe

Return:

```
- ``None``
```

If the fits files conform to required input for the recipe everything will pass silently, otherwise an exception shall be raised.

produce_product()

generate the order-table with polynomal fits of order-centres

Return:

- `productPath` – the path to the order-table

2.6.2.3 `soxs_spatial_solution` - PLANNED

The purpose of this recipe is to further enhance the wavelength solution achieved with `soxs_disp_solution` by expanding the solution into the spatial dimension (along the slit). This 2-dimensional solution will then account for any tilt in the spectral lines.[1]

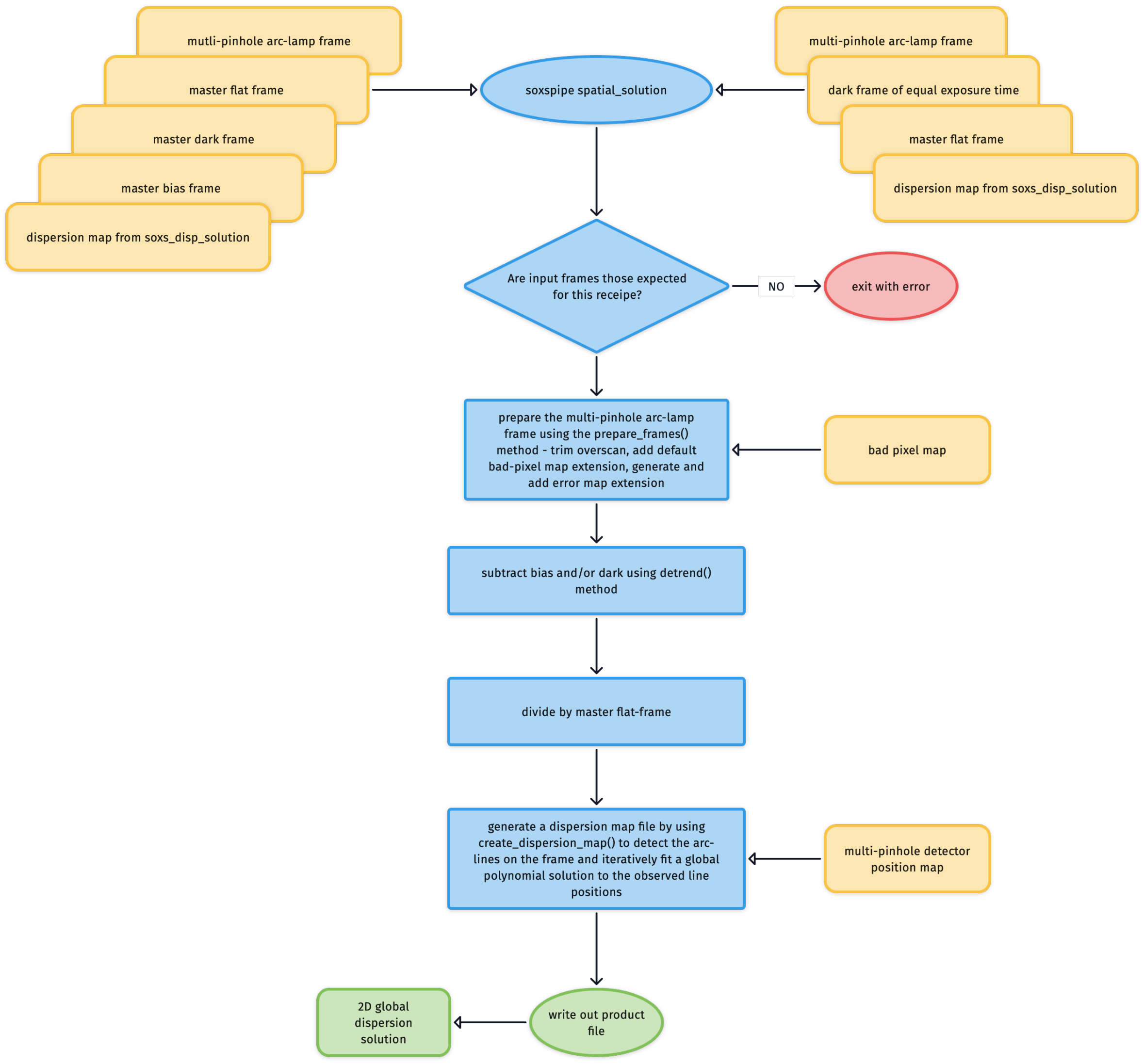
Each pinhole in the multi-pinhole mask is 0.5" in diameter and the 9 pinholes are evenly spaced along the 11" slit with a 1.4" gap between adjacent holes. This knowledge affords us the ability to now map the dispersion solution along the spatial direction.

Input

Data Type	Content	Related OB
Image	Arc Lamp through multi-pinhole mask	`SOXS_slt_cal_VISArcsMultiplePinhole`, `SOXS_slt_cal_NIRArcsMultiplePinhole`
Image	Master Dark Frame (VIS only)	-
Image	Master Bias Frame (VIS only)	-
Image	Dark frame (Lamp-Off) of equal exposure length as multi-pinhole frame (Lamp-On) (NIR only)	`SOXS_slt_cal_NIRArcsMultiplePinhole`
File	First-guess Dispersion Map table	
File	Pinhole Map	

Method

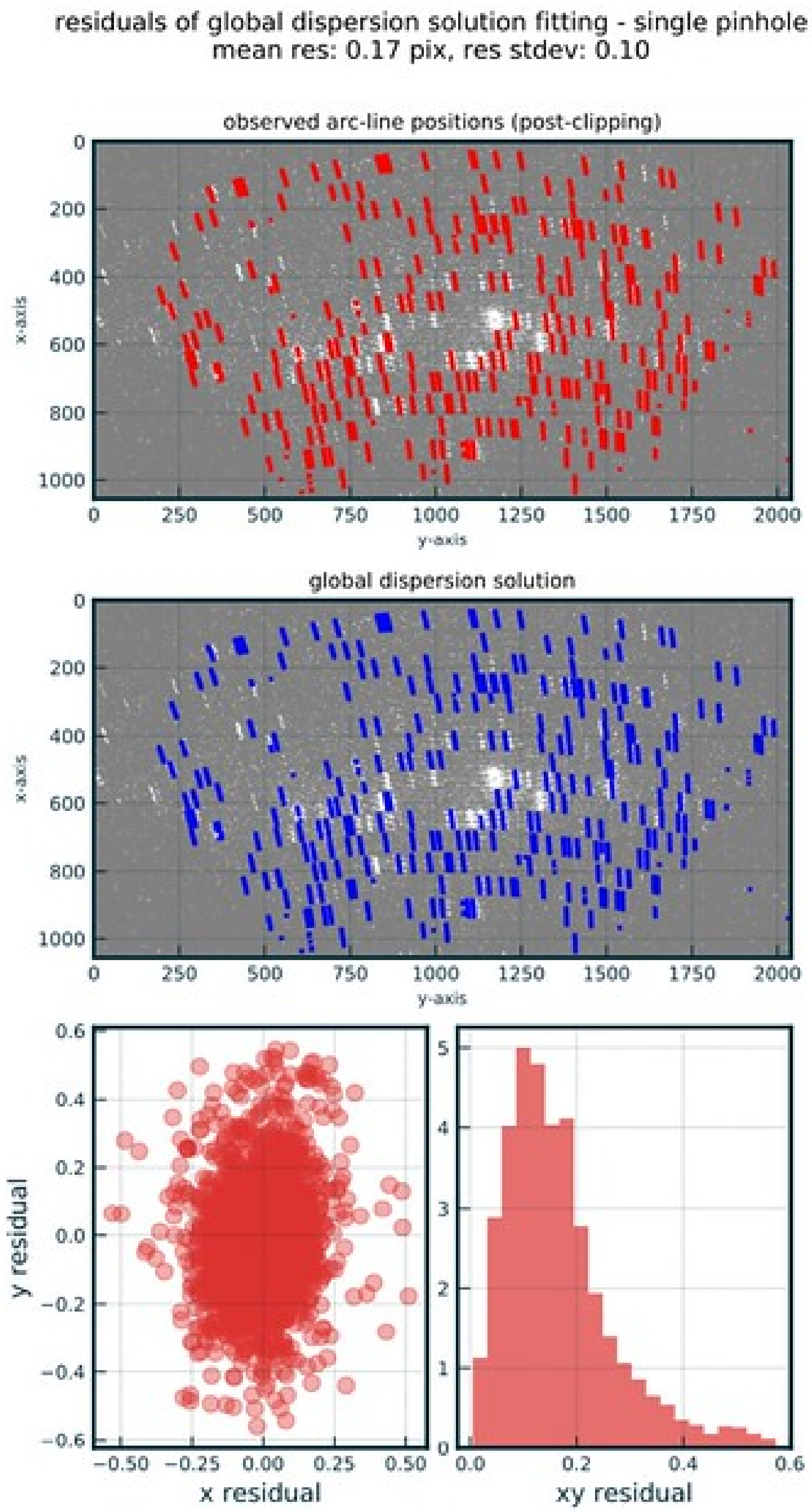
Having prepared the multi-pinhole frame the bias and dark signatures are removed and the frame is divided through by the master flat frame. The calibrated frame and the first-guess dispersion map are passed to the `create_dispersion_map` utility to produce a 2D dispersion solution covering both the spectral and spatial dimensions.


mutli-pinhole arc-lamp frame
master flat frame
master dark frame
master bias frame
dispersion map from soxs_disp_solution
soxspipe spatial_solution
multi-pinhole arc-lamp frame
dark frame of equal exposure time
master flat frame
dispersion map from soxs_disp_solution
Are input frames those expected for this receipe?
NO
exit with error
prepare the multi-pinhole arc-lamp frame using the prepare_frames() method - trim overscan, add default bad-pixel map extension, generate and add error map extension
bad pixel map
subtract bias and/or dark using detrend() method
divide by master flat-frame
generate a dispersion map file by using create_dispersion_map() to detect the arc-lines on the frame and iteratively fit a global polynomial solution to the observed line positions
multi-pinhole detector position map
write out product file
2D global dispersion solution

Output

Data Type	Content
File (subject to change)	Dispersion Map table giving coefficients of polynomials describing 2D dispersion/spatial solution

QC Metrics


residuals of global dispersion solution fitting - single pinhole
mean res: 0.17 pix, res stdev: 0.10
observed arc-line positions (post-clipping)
x-axis
y-axis
global dispersion solution
x-axis
y-axis
y residual
x residual
xy residual

Metric	Description
	…

Recipe API

class soxs_spatial_solution(*log*, *settings=False*, *inputFrames=[]*, *verbose=False*, *overwrite=False*, *create2DMap=True*, *polyOrders=False*)

The soxs_spatial_solution recipe

Key Arguments

- `log` – logger
- `settings` – the settings dictionary
- `inputFrames` – input fits frames. Can be a directory, a set-of-files (SOF) file or a list of fits frame paths
- `verbose` – verbose. True or False. Default *False*
- `overwrite` – overwrite the prodcut file if it already exists. Default *False*
- `create2DMap` – create the 2D image map of wavelength, slit-position and order from disp solution.
- `polyOrders` – the orders of the x-y polynomials used to fit the dispersion solution. Overrides parameters found in the yaml settings file. e.g 345435 is order_x=3, order_y=4 ,wavelength_x=5 ,wavelength_y=4, slit_x=3 ,slit_y=5. Default *False*.

See `produce_product` method for usage.

Todo:

- add a tutorial about `soxs_spatial_solution` to documentation

verify_input_frames()

verify input frames match those required by the ``soxs_spatial_solution`` recipe

If the fits files conform to required input for the recipe everything will pass silently, otherwise an exception shall be raised.

produce_product()

generate the 2D dispersion map

Return:

- `productPath` – the path to the 2D dispersion map

Usage

```
from soxspipe.recipes import soxs_spatial_solution
recipe = soxs_spatial_solution(
    log=log,
    settings=settings,
    inputFrames=fileList
)
disp_map = recipe.produce_product()
```

1. relative to the perpendicular of the dispersion direction

2.6.2.4 `soxs_straighten` - PLANNED

This recipe takes the full dispersion map given by `soxs_spatial_solution` and uses it to map images from their original representation of the detector surface to one that presents the signal in a wavelength by slit-position coordinate system.

Input

Data Type	Content	Related OB
File	Coefficients of polynomials providing a full dispersion-spatial solution	
Image	An associated spectral image requiring rectification	Many

Parameters

Parameter	Description	Type	Entry Point	Related Util
`straighten_grid_res_wavelength`	size of the grid cell in wavelength dimension (nm)	float	settings file	
`straighten_grid_res_split`	size of the grid cell in slit dimension (arcsec)	float	settings file	

Method

We now have a pair of polynomials that can be used to give the exact pixel on the detector containing flux resulting from a specific order, with a given wavelength and slit position.

$$X = \sum_{ijk} c_{ijk} \times n^i \times \lambda^j \times s^k$$

$$Y = \sum_{ijk} c_{ijk} \times n^i \times \lambda^j \times s^k$$

To begin we want to create a full wavelength and slit-position map; a 2D grid of wavelengths along one axis and slit-position along the other. Using the polynomial solutions above, we populate each cell in the grid with its corresponding detector pixel coordinate. The 2D grid is of fine enough resolution so that many cells in the grid are mapped to each individual detector pixel. With this map in hand we can now assign flux recorded in each detector pixel to the corresponding cells in the 2D wavelength and slit-position grid. The flux from each detector is evenly distributed between all cells found to be associate with that pixel; so if 9 cells are associated then each cell gets 1/9th of the pixel flux.

The error and bad-pixel extensions go through the same mapping process.

recipes/soxs_straighten.png

Output

Data Type	Content
Images	The straightened images containing flux represented in wavelength space; one for each order

QC Metrics

Metric	Description
	…

Recipe API

2.7 Utilities

The soxspipe utilities can be viewed as the tool-kit with which the soxspipe recipes are built. Many of these utilities are used across multiple recipes and they can either be stand-alone objects or methods of the recipe itself.

2.7.1 `clip_and_stack`

`clip_and_stack` mean combines input frames after sigma-clipping outlying pixels using a median value with median absolute deviation (mad) as the deviation function.

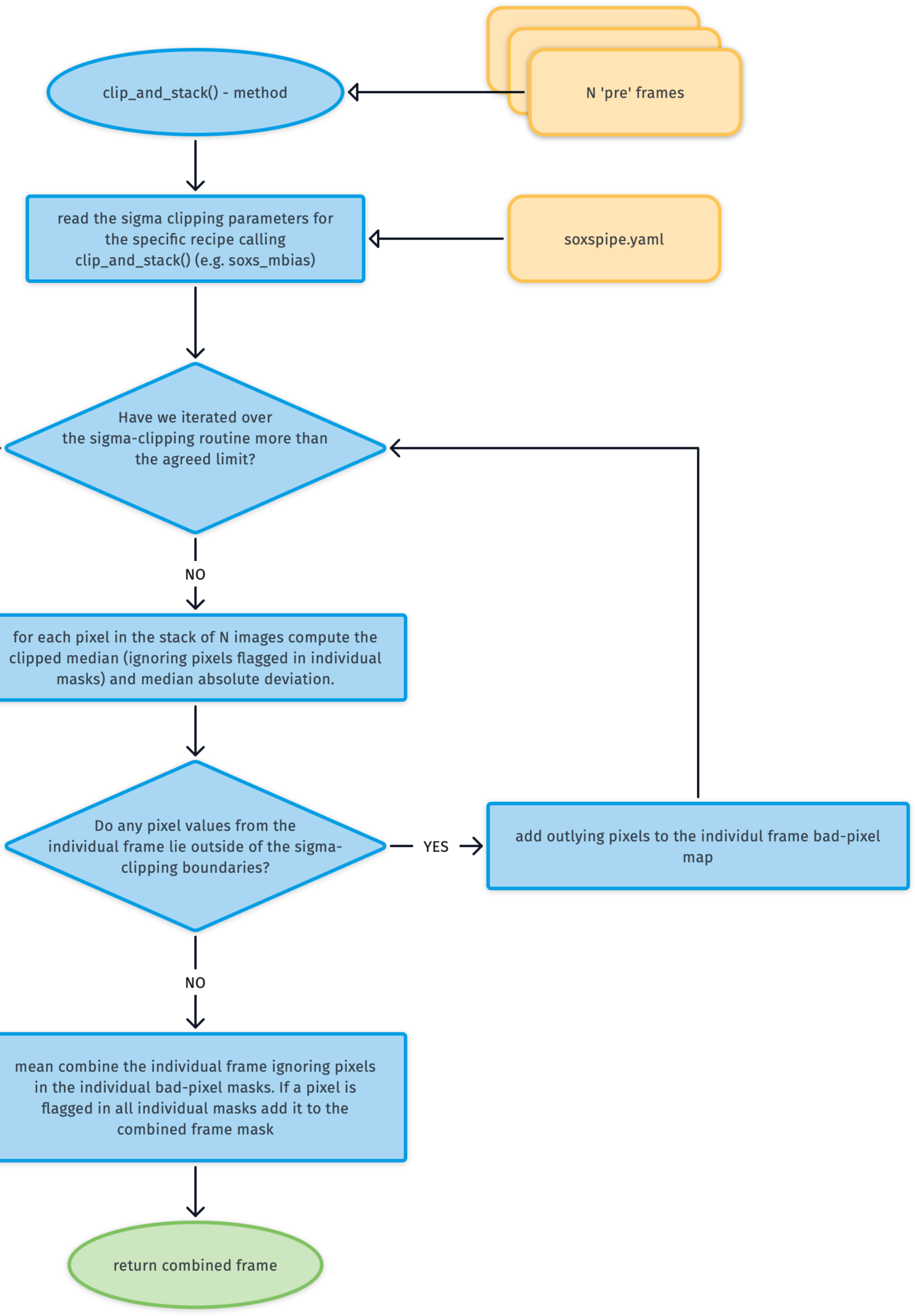
Before combining the frames we want to 'clip' any outlying pixel values found in the individual frames that are to be stacked. We isolate and remove pixels from any averaging calculation (mean or median) that have a value that strays too far from the 'typical' pixel value.

Using the median pixel value as the 'typical' value and the *median absolute deviation* (MAD) as a proxy for the standard-deviation we can accurately identify rogue pixels. For any given set of pixel values:

$$MAD = \frac{1}{N}\sum_{i=0}^{N}|x_i - \mathrm{median}(x)|.$$

The clipping is done iteratively so newly found rogue pixels are masks, median values are recalculated and clipping repeated. The iterative process stops whenever either no more bad-pixels are to be found or the maximum number of iterations has been reached.

After the clipping has been completed individual frames are mean-combined, ignoring pixels in the individual bad-pixel masks. If a pixel is flagged as 'bad' in all individual masks it is added to the combined frame bad-pixel mask.


clip_and_stack() - method
N 'pre' frames
read the sigma clipping parameters for the specific recipe calling clip_and_stack() (e.g. soxs_mbias)
soxspipe.yaml
Have we iterated over the sigma-clipping routine more than the agreed limit?
NO
YES
for each pixel in the stack of N images compute the clipped median (ignoring pixels flagged in individual masks) and median absolute deviation.
Do any pixel values from the individual frame lie outside of the sigma-clipping boundaries?
YES
add outlying pixels to the individul frame bad-pixel map
NO
mean combine the individual frame ignoring pixels in the individual bad-pixel masks. If a pixel is flagged in all individual masks add it to the combined frame mask
return combined frame

`_base_recipe_.`**`clip_and_stack`**(*frames*, *recipe*, *ignore_input_masks=False*, *post_stack_clipping=True*)

mean combine input frames after sigma-clipping outlying pixels using a median value with median absolute deviation (mad) as the deviation function

Key Arguments:

- `frames` – an ImageFileCollection of the frames to stack or a list of CCDData objects
- `recipe` – the name of recipe needed to read the correct settings from the yaml files
- `ignore_input_masks` – ignore the input masks during clip and stacking?
- `post_stack_clipping` – allow cross-plane clipping on combined frame. Clipping settings in setting file. Default *True*.

Return:

- `combined_frame` – the combined master frame (with updated bad-pixel and uncertainty maps)

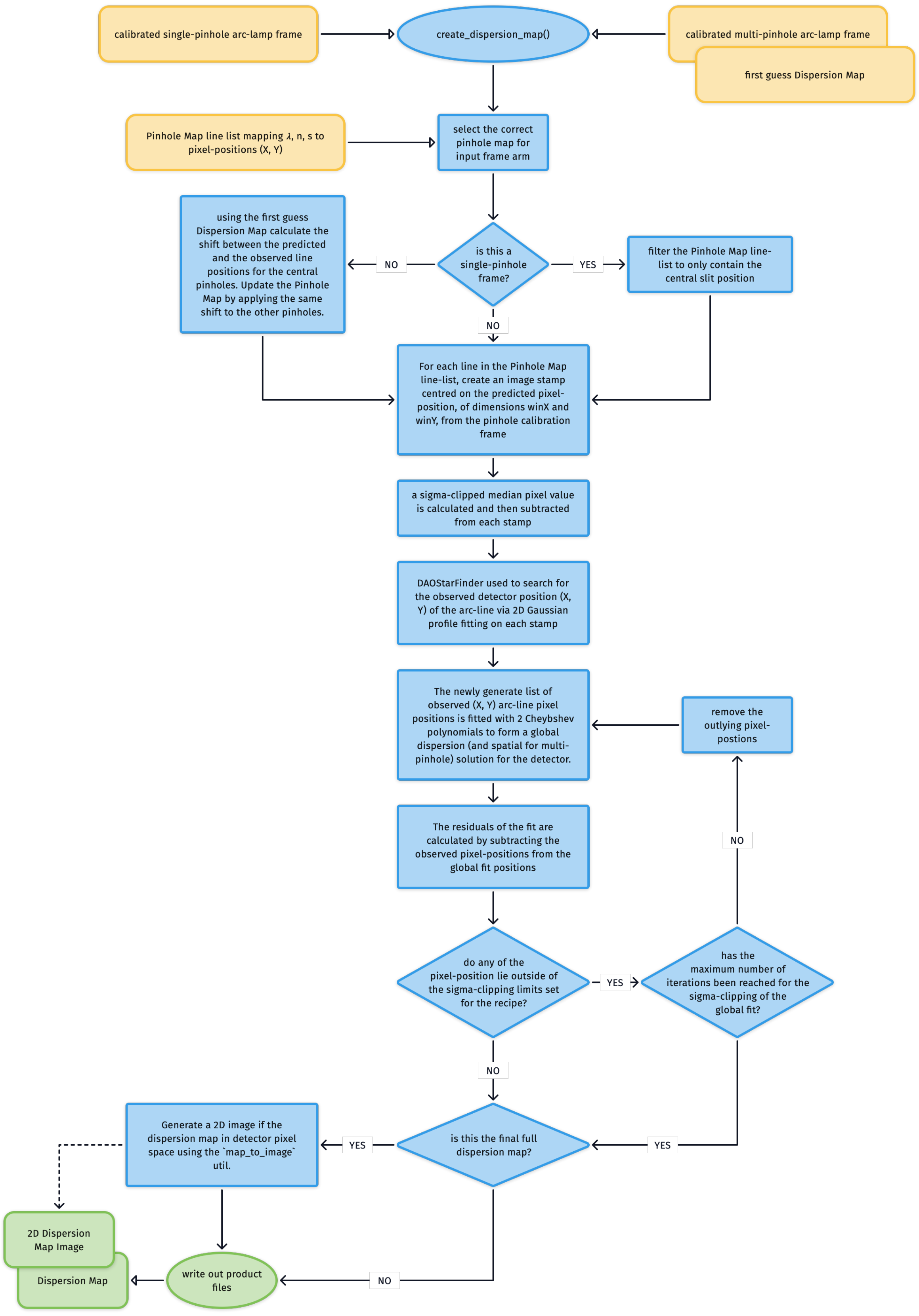
Usage:

This snippet can be used within the recipe code to combine individual (using bias frames as an example):

```
combined_bias_mean = self.clip_and_stack(
    frames=self.inputFrames, recipe="soxs_mbias", ignore_input_masks=False, post_
↪stack_clipping=True)
```

2.7.2 `create_dispersion_map`

The `create_dispersion_map` utility is used to search for arc-lines in the single/multi-pinhole arc-lamp frames and then iteratively fit a global polynomial dispersion solution (and spatial-solution in the case of multi-pinhole frame) with the observed line-positions. It is used by both the `soxs_disp_solution`) and `soxs_spatial_solution`) solution recipes.


calibrated single-pinhole arc-lamp frame
create_dispersion_map()
calibrated multi-pinhole arc-lamp frame
first guess Dispersion Map
Pinhole Map line list mapping λ, n, s to pixel-positions (X, Y)
select the correct pinhole map for input frame arm
using the first guess Dispersion Map calculate the shift between the predicted and the observed line positions for the central pinholes. Update the Pinhole Map by applying the same shift to the other pinholes.
NO
is this a single-pinhole frame?
YES
filter the Pinhole Map line-list to only contain the central slit position
NO
For each line in the Pinhole Map line-list, create an image stamp centred on the predicted pixel-position, of dimensions winX and winY, from the pinhole calibration frame
a sigma-clipped median pixel value is calculated and then subtracted from each stamp
DAOStarFinder used to search for the observed detector position (X, Y) of the arc-line via 2D Gaussian profile fitting on each stamp
The newly generate list of observed (X, Y) arc-line pixel positions is fitted with 2 Cheybshev polynomials to form a global dispersion (and spatial for multi-pinhole) solution for the detector.
remove the outlying pixel-postions
The residuals of the fit are calculated by subtracting the observed pixel-positions from the global fit positions
NO
do any of the pixel-position lie outside of the sigma-clipping limits set for the recipe?
YES
has the maximum number of iterations been reached for the sigma-clipping of the global fit?
NO
Generate a 2D image if the dispersion map in detector pixel space using the `map_to_image` util.
YES
is this the final full dispersion map?
YES
2D Dispersion Map Image
Dispersion Map
write out product files
NO

In the static calibration suite we have Pinhole Maps listing the wavelength λ, order number n and slit position s of the spectral lines alongside a first approximation of their (X, Y) pixel-positions on the detector.

If the input frame is a single-pinhole frame, we can filter the Pinhole Map to contain just the central pinhole positions. If however input is the multi-pinhole frame then we use the first guess Dispersion Map (created with `soxs_disp_solution`) to calculate the shift between the predicted and the observed line positions for the central pinholes. We then update the Pinhole Map by applying the same shift to the other pinholes.

For each line in the Pinhole Map line-list:

- an image stamp centred on the predicted pixel-position (X_o, Y_o), of dimensions winX and winY, is generated from the pinhole calibration frame
- a sigma-clipped median pixel value is calculated and then subtracted from each stamp
- DAOStarFinder is employed to search for the *observed* detector position (X, Y) of the arc-line via 2D Gaussian profile fitting on the stamp

We now have a list of arc-line wavelengths and their observed pixel-positions and the order they were detected in. These values are used to iteratively fit two polynomials that describe the global dispersion solution for the detector. In the case of the single-pinhole frames these are:

$$X = \sum_{ij} c_{ij} \times n^i \times \lambda^j$$

$$Y = \sum_{ij} c_{ij} \times n^i \times \lambda^j$$

where λ is wavelength and n is the echelle order number.

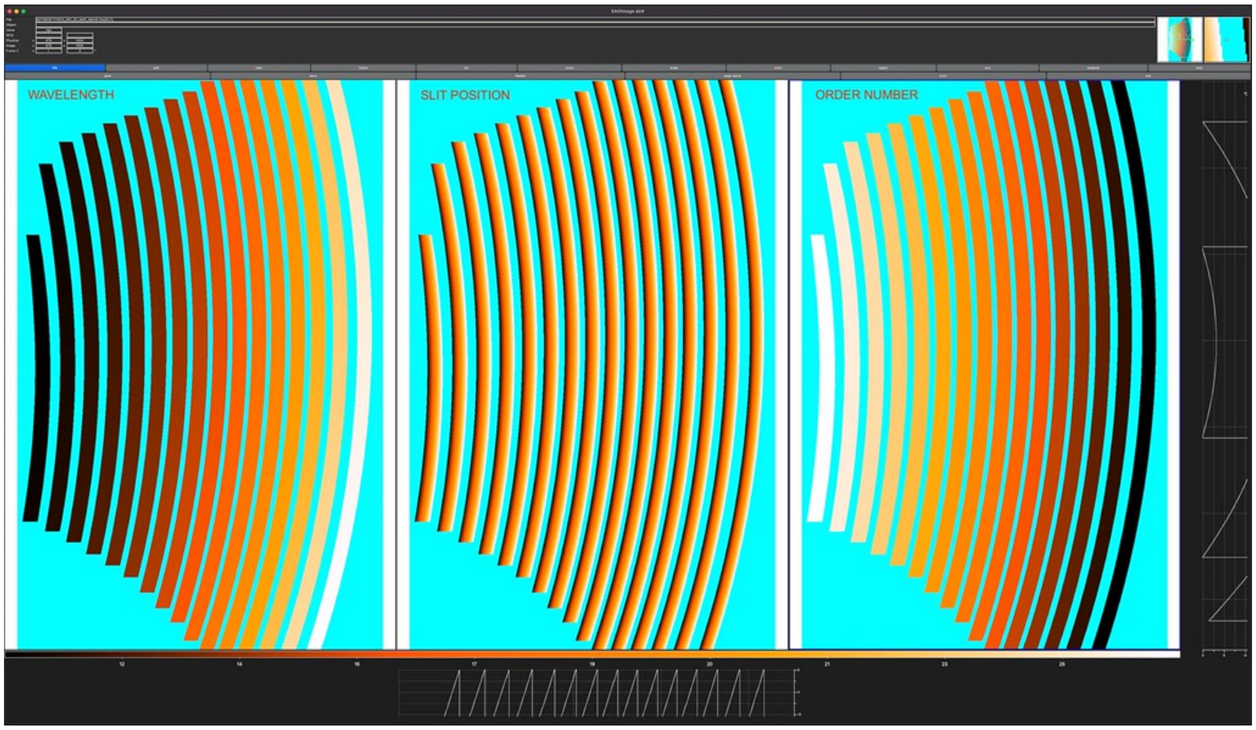
In the case of the multi-pinhole we also have the slit position s and so adding a spatial solution to the dispersion solution:

$$X = \sum_{ijk} c_{ijk} \times n^i \times \lambda^j \times s^k$$

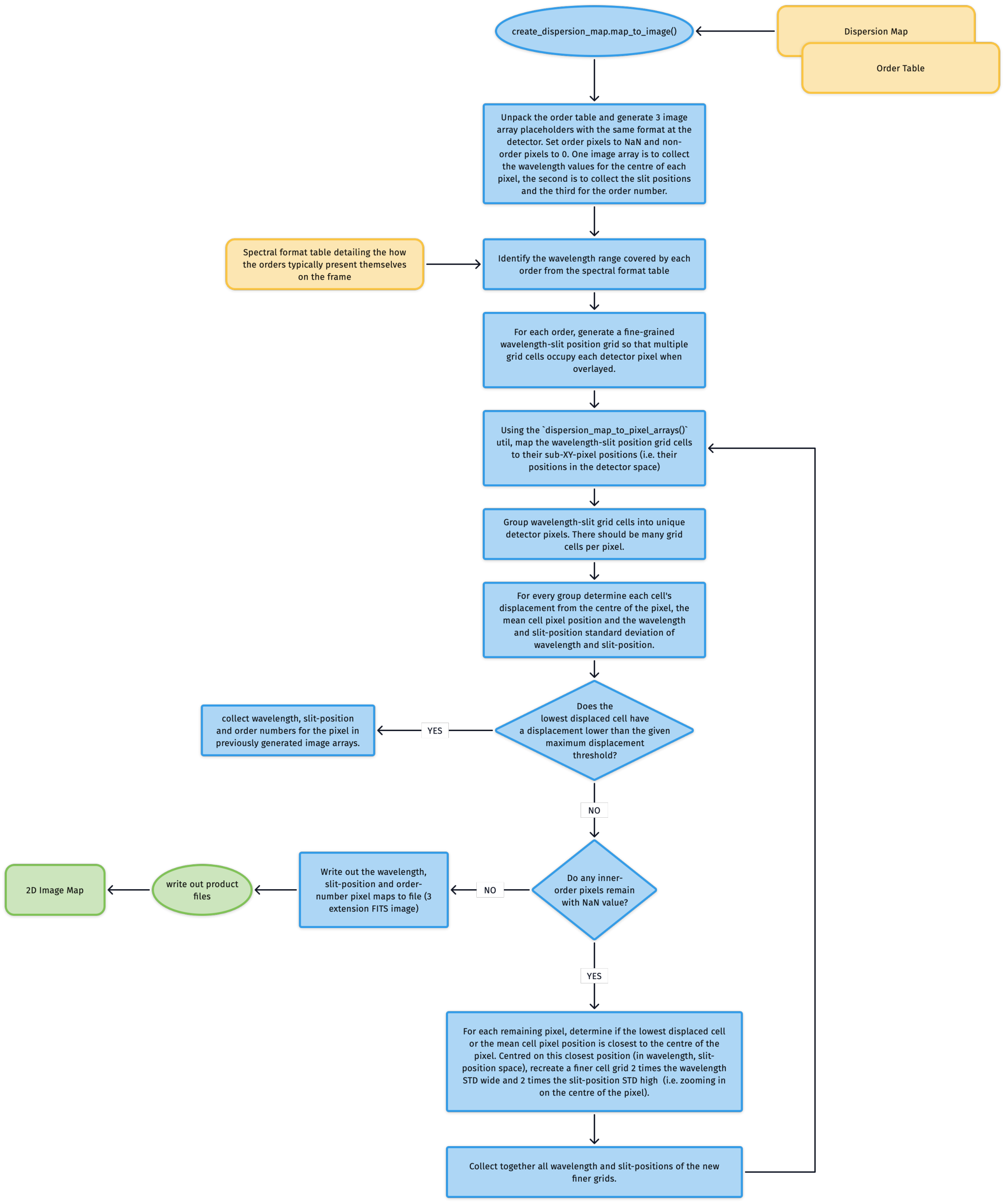
$$Y = \sum_{ijk} c_{ijk} \times n^i \times \lambda^j \times s^k$$

Upon each iteration the residuals between the fits and the measured pixel-positions are calculated and sigma-clipping is employed to eliminate measurements that stray to far from the fit. Once the maximum number of iterations is reach, or all outlying lines have been clipped, the coefficients of the polynomials are written to a Dispersion Map file.

2.7.2.1 2D Image Map



The Dispersion Map is used to generate a triple extension FITS file with each extension image exactly matching the dimensions of the detector. The first extension contains the wavelength value at the centre of each pixel location, the second the slit-position and the third the order number. The solutions for these images are iteratively converged on in a brute force manner (see workflow diagram below). These image maps are used in sky-background subtraction and object extraction utilities.


create_dispersion_map.map_to_image()
Dispersion Map
Order Table
Unpack the order table and generate 3 image array placeholders with the same format at the detector. Set order pixels to NaN and non-order pixels to 0. One image array is to collect the wavelength values for the centre of each pixel, the second is to collect the slit positions and the third for the order number.
Spectral format table detailing the how the orders typically present themselves on the frame
Identify the wavelength range covered by each order from the spectral format table
For each order, generate a fine-grained wavelength-slit position grid so that multiple grid cells occupy each detector pixel when overlayed.
Using the `dispersion_map_to_pixel_arrays()` util, map the wavelength-slit position grid cells to their sub-XY-pixel positions (i.e. their positions in the detector space)
Group wavelength-slit grid cells into unique detector pixels. There should be many grid cells per pixel.
For every group determine each cell's displacement from the centre of the pixel, the mean cell pixel position and the wavelength and slit-position standard deviation of wavelength and slit-position.
Does the lowest displaced cell have a displacement lower than the given maximum displacement threshold?
YES
collect wavelength, slit-position and order numbers for the pixel in previously generated image arrays.
NO
Do any inner-order pixels remain with NaN value?
NO
Write out the wavelength, slit-position and order-number pixel maps to file (3 extension FITS image)
write out product files
2D Image Map
YES
For each remaining pixel, determine if the lowest displaced cell or the mean cell pixel position is closest to the centre of the pixel. Centred on this closest position (in wavelength, slit-position space), recreate a finer cell grid 2 times the wavelength STD wide and 2 times the slit-position STD high (i.e. zooming in on the centre of the pixel).
Collect together all wavelength and slit-positions of the new finer grids.

class create_dispersion_map(*log, settings, recipeSettings, pinholeFrame, firstGuessMap=False, orderTable=False, qcTable=False, productsTable=False, sofName=False, create2DMap=True*)

detect arc-lines on a pinhole frame to generate a dispersion solution

Key Arguments:

- `log` – logger
- `settings` – the settings dictionary
- `recipeSettings` – the recipe specific settings
- `pinholeFrame` – the calibrated pinhole frame (single or multi)
- `firstGuessMap` – the first guess dispersion map from the `soxs_disp_solution` recipe (needed in `soxs_spat_solution` recipe). Default *False*.
- `orderTable` – the order geometry table
- `qcTable` – the data frame to collect measured QC metrics
- `productsTable` – the data frame to collect output products
- `sofName` – name of the originating SOF file
- `create2DMap` – create the 2D image map of wavelength, slit-position and order from disp solution.

Usage:

```
from soxspipe.commonutils import create_dispersion_map
mapPath, mapImagePath, res_plots, qcTable, productsTable = create_dispersion_map(
    log=log,
    settings=settings,
    pinholeFrame=frame,
    firstGuessMap=False,
    qcTable=self.qc,
    productsTable=self.products
).get()
```

get()
generate the dispersion map

Return:

- `mapPath` – path to the file containing the coefficients of the x,y polynomials of the global dispersion map fit

get_predicted_line_list()
lift the predicted line list from the static calibrations

Return:

- `orderPixelTable` – a panda's data-frame containing wavelength,order,slit_index,slit_position,detector_x,detector_y

detect_pinhole_arc_line(*predictedLine*, *iraf=True*)
detect the observed position of an arc-line given the predicted pixel positions

Key Arguments:

- `predictedLine` – single predicted line coordinates from predicted line-list
- `iraf` – use IRAF star finder to generate a FWHM

Return:

- `predictedLine` – the line with the observed pixel coordinates appended (if detected, otherwise nan)

write_map_to_file(*xcoeff*, *ycoeff*, *orderDeg*, *wavelengthDeg*, *slitDeg*)
write out the fitted polynomial solution coefficients to file

Key Arguments:

- `xcoeff` – the x-coefficients
- `ycoeff` – the y-coefficients
- `orderDeg` – degree of the order fitting
- `wavelengthDeg` – degree of wavelength fitting
- `slitDeg` – degree of the slit fitting (False for single pinhole)

Return:

- `disp_map_path` – path to the saved file

calculate_residuals(*orderPixelTable*, *xcoeff*, *ycoeff*, *orderDeg*, *wavelengthDeg*, *slitDeg*, *writeQCs=False*, *pixelRange=False*)
calculate residuals of the polynomial fits against the observed line positions

Key Arguments:

- `orderPixelTable` – the predicted line list as a data frame
- `xcoeff` – the x-coefficients
- `ycoeff` – the y-coefficients
- `orderDeg` – degree of the order fitting
- `wavelengthDeg` – degree of wavelength fitting
- `slitDeg` – degree of the slit fitting (False for single pinhole)
- `writeQCs` – write the QCs to dataframe? Default *False*
- `pixelRange` – return centre pixel *and* +- 2nm from the centre pixel (to measure the pixel scale)

Return:

- `residuals` – combined x-y residuals
- `mean` – the mean of the combine residuals
- `std` – the stdev of the combine residuals
- `median` – the median of the combine residuals

fit_polynomials(*orderPixelTable*, *wavelengthDeg*, *orderDeg*, *slitDeg*, *missingLines=False*)
iteratively fit the dispersion map polynomials to the data, clipping residuals with each iteration

Key Arguments:

- `orderPixelTable` – data frame containing order, wavelengths, slit positions and observed pixel positions
- `wavelengthDeg` – degree of wavelength fitting
- `orderDeg` – degree of the order fitting
- `slitDeg` – degree of the slit fitting (0 for single pinhole)
- `missingLines` – lines not detected on the image

Return:

- `xcoeff` – the x-coefficients post clipping
- `ycoeff` – the y-coefficients post clipping

- `goodLinesTable` – the fitted line-list with metrics
- `clippedLinesTable` – the lines that were sigma-clipped during polynomial fitting

create_placeholder_images(*order=False*, *plot=False*, *reverse=False*)

create CCDData objects as placeholders to host the 2D images of the wavelength and spatial solutions from dispersion solution map

Key Arguments:

- `order` – specific order to generate the placeholder pixels for. Inner-order pixels set to NaN, else set to 0. Default *False* (generate all orders)
- `plot` – generate plots of placeholder images (for debugging). Default *False*.
- `reverse` – Inner-order pixels set to 0, else set to NaN (reverse of default output).

Return:

- `slitMap` – placeholder image to add pixel slit positions to
- `wlMap` – placeholder image to add pixel wavelength values to

Usage:

```
slitMap, wlMap, orderMap = self._create_placeholder_images(order=order)
```

map_to_image(*dispersionMapPath*)

convert the dispersion map to images in the detector format showing pixel wavelength values and slit positions

Key Arguments:

- `dispersionMapPath` – path to the full dispersion map to convert to images

Return:

- `dispersion_image_filePath` – path to the FITS image with an extension for wavelength values and another for slit positions

Usage:

```
mapImagePath = self.map_to_image(dispersionMapPath=mapPath)
```

order_to_image(*orderInfo*)

convert a single order in the dispersion map to wavelength and slit position images

Key Arguments:

- `orderInfo` – tuple containing the order number to generate the images for, the minimum wavelength to consider (from format table) and maximum wavelength to consider (from format table).

Return:

- `slitMap` – the slit map with order values filled
- `wlMap` – the wavelengths map with order values filled

Usage:

```
slitMap, wlMap = self.order_to_image(order=order,minWl=minWl, maxWl=maxWl)
```

convert_and_fit(*order*, *bigWlArray*, *bigSlitArray*, *slitMap*, *wlMap*, *iteration*, *plots=False*)

convert wavelength and slit position grids to pixels

Key Arguments:

- `order` – the order being considered
- `bigWlArray` – 1D array of all wavelengths to be converted
- `bigSlitArray` – 1D array of all split-positions to be converted (same length as `bigWlArray`)
- `slitMap` – place-holder image hosting fitted pixel slit-position values
- `wlMap` – place-holder image hosting fitted pixel wavelength values
- `iteration` – the iteration index (used for CL reporting)
- `plots` – show plot of the slit-map

Return:

- `orderPixelTable` – dataframe containing unfitted pixel info
- `remainingCount` – number of remaining pixels in orderTable

Usage:

```
orderPixelTable = self.convert_and_fit(
        order=order, bigWlArray=bigWlArray, bigSlitArray=bigSlitArray,␣
↪slitMap=slitMap, wlMap=wlMap)
```

create_new_static_line_list(*dispersionMapPath*)

using a first pass dispersion solution, use a line atlas to generate a more accurate and more complete static line list

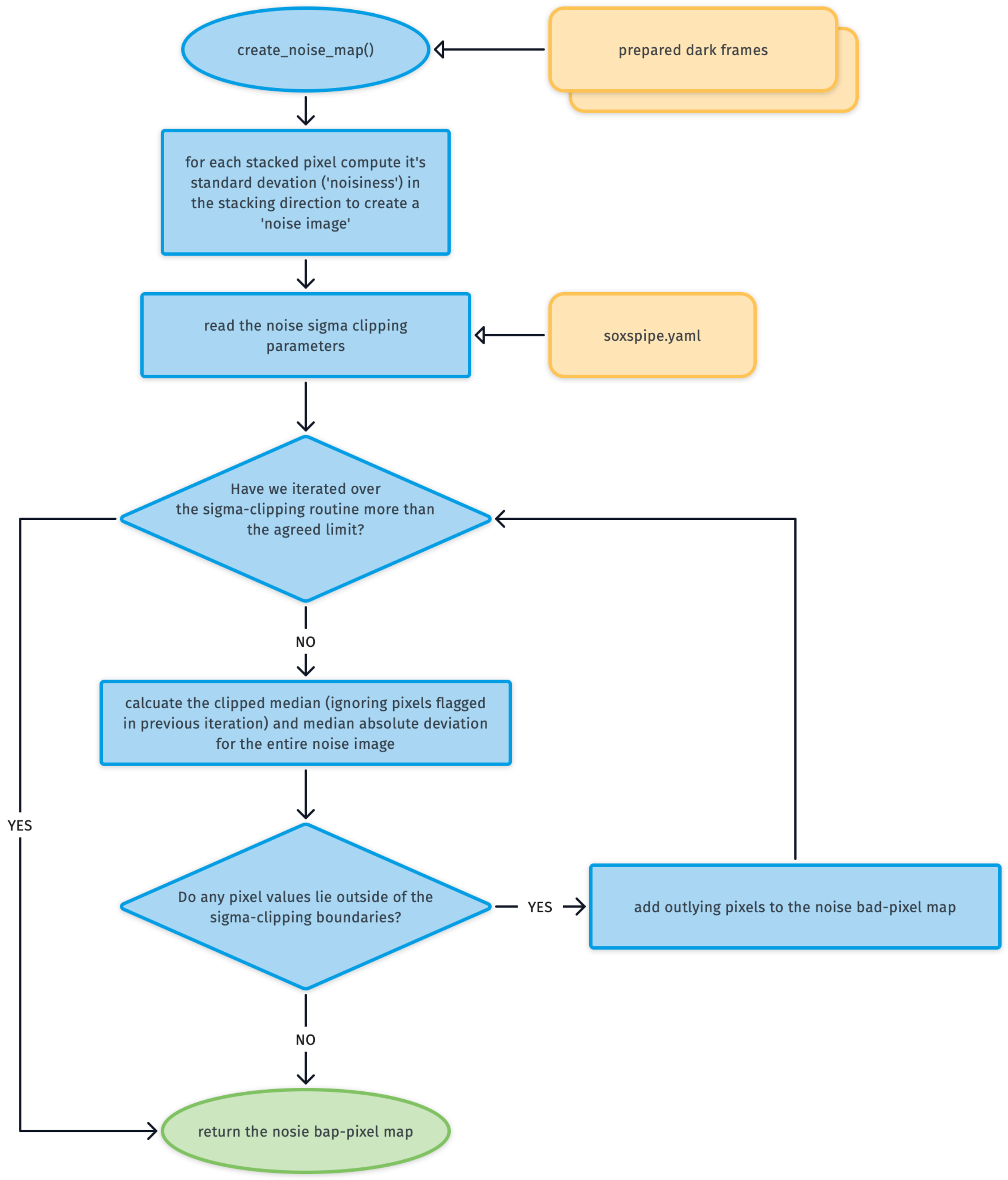
Key Arguments:

```
- `dispersionMapPath` -- path to the first pass dispersion solution
```

Return:

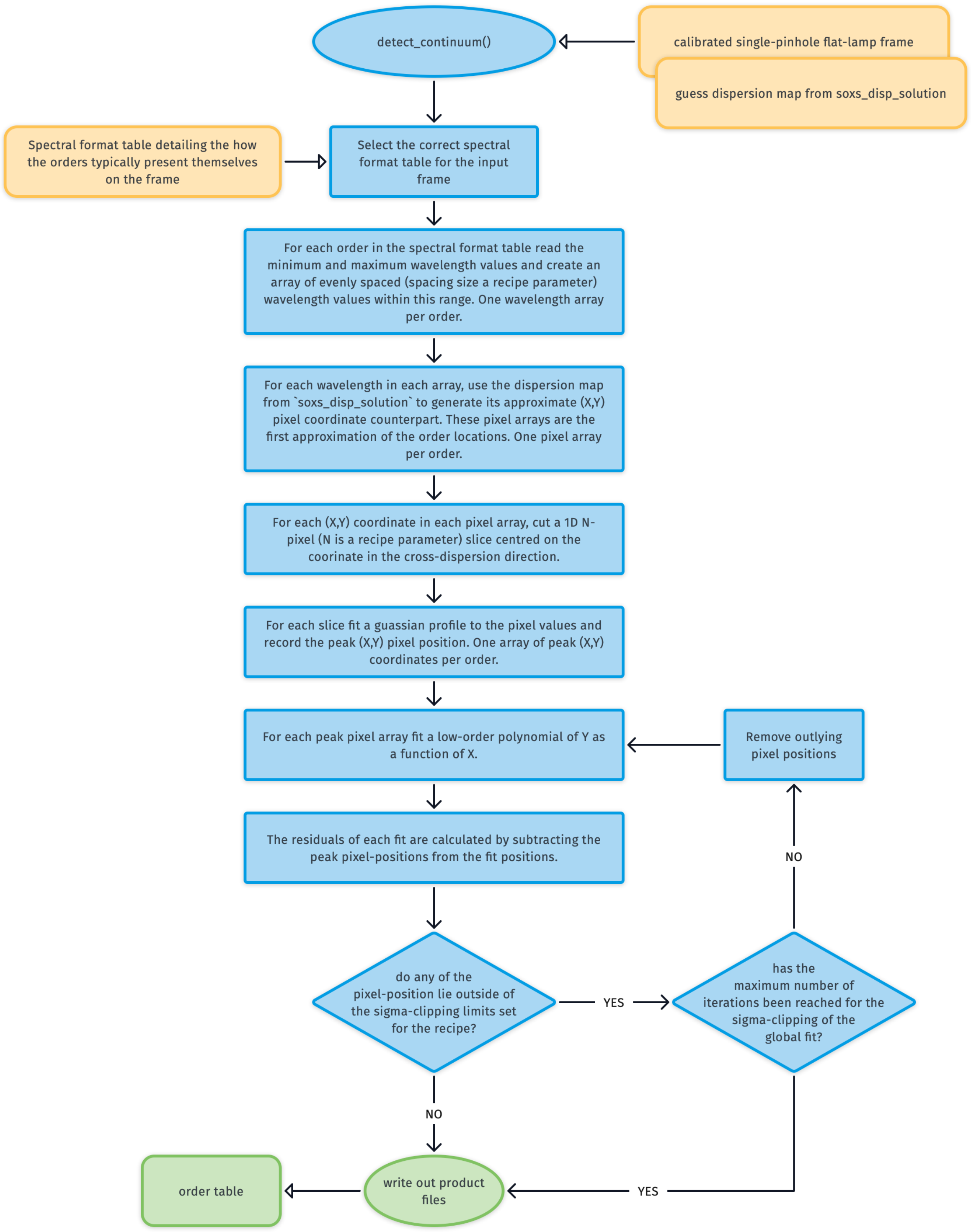
```
- `newPredictedLineList` -- a new predicted line list (to replace the static␣
↪calibration line-list)
```

2.7.3 `create_noise_map` - PLANNED

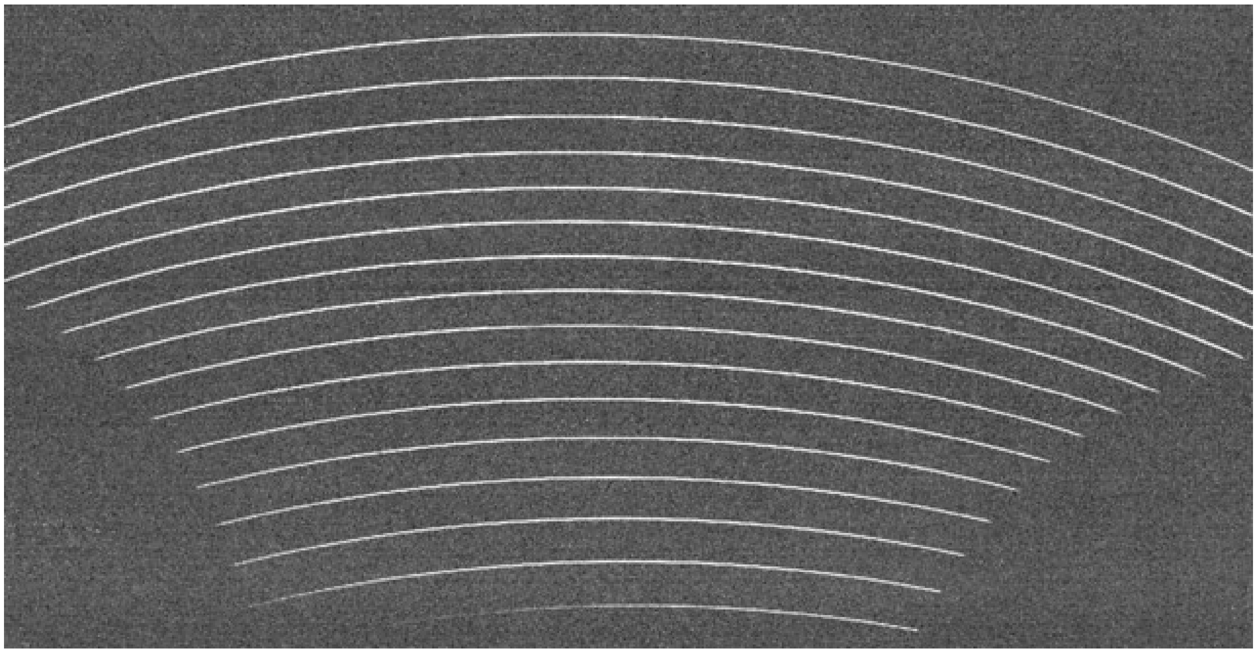

create_noise_map()
prepared dark frames
for each stacked pixel compute it's standard devation ('noisiness') in the stacking direction to create a 'noise image'
read the noise sigma clipping parameters
soxspipe.yaml
Have we iterated over the sigma-clipping routine more than the agreed limit?
NO
YES
calcuate the clipped median (ignoring pixels flagged in previous iteration) and median absolute deviation for the entire noise image
Do any pixel values lie outside of the sigma-clipping boundaries?
YES
add outlying pixels to the noise bad-pixel map
NO
return the nosie bap-pixel map

2.7.4 detect_continuum

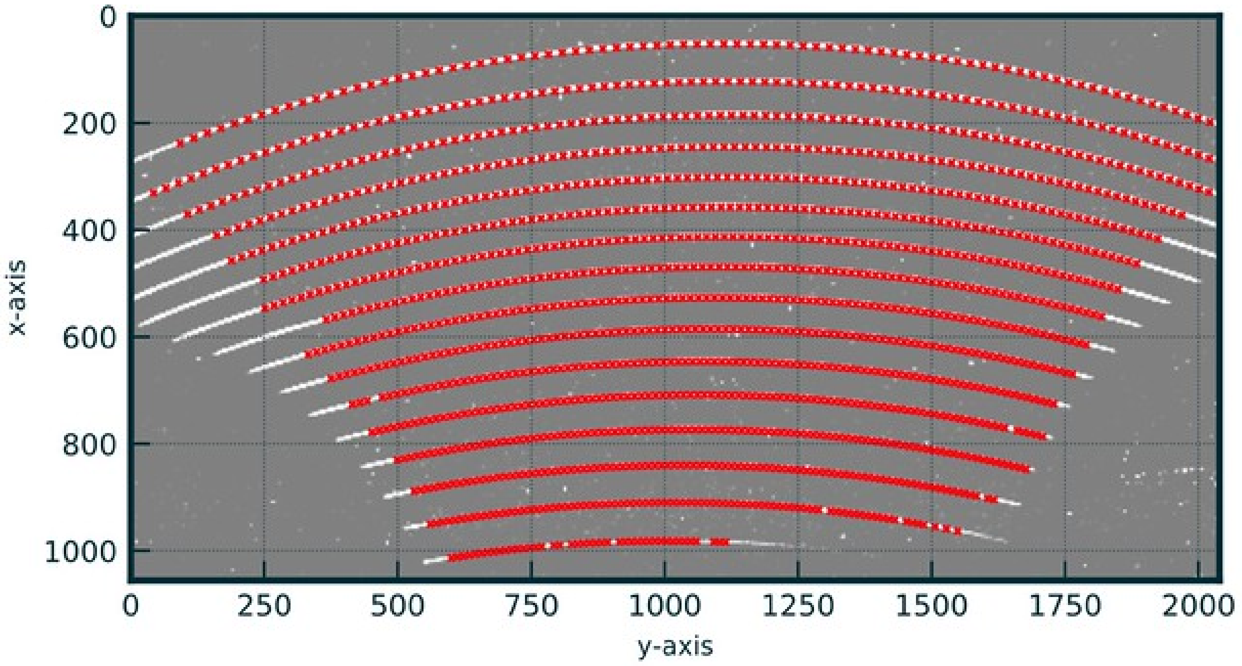
The purpose of the `detect_continuum` utility is to find and fit the order centres with low-level polynomials.


detect_continuum()
calibrated single-pinhole flat-lamp frame
guess dispersion map from soxs_disp_solution
Spectral format table detailing the how the orders typically present themselves on the frame
Select the correct spectral format table for the input frame
For each order in the spectral format table read the minimum and maximum wavelength values and create an array of evenly spaced (spacing size a recipe parameter) wavelength values within this range. One wavelength array per order.
For each wavelength in each array, use the dispersion map from `soxs_disp_solution` to generate its approximate (X,Y) pixel coordinate counterpart. These pixel arrays are the first approximation of the order locations. One pixel array per order.
For each (X,Y) coordinate in each pixel array, cut a 1D N-pixel (N is a recipe parameter) slice centred on the coorinate in the cross-dispersion direction.
For each slice fit a guassian profile to the pixel values and record the peak (X,Y) pixel position. One array of peak (X,Y) coordinates per order.
For each peak pixel array fit a low-order polynomial of Y as a function of X.
Remove outlying pixel positions
The residuals of each fit are calculated by subtracting the peak pixel-positions from the fit positions.
NO
do any of the pixel-position lie outside of the sigma-clipping limits set for the recipe?
YES
has the maximum number of iterations been reached for the sigma-clipping of the global fit?
NO
order table
write out product files
YES

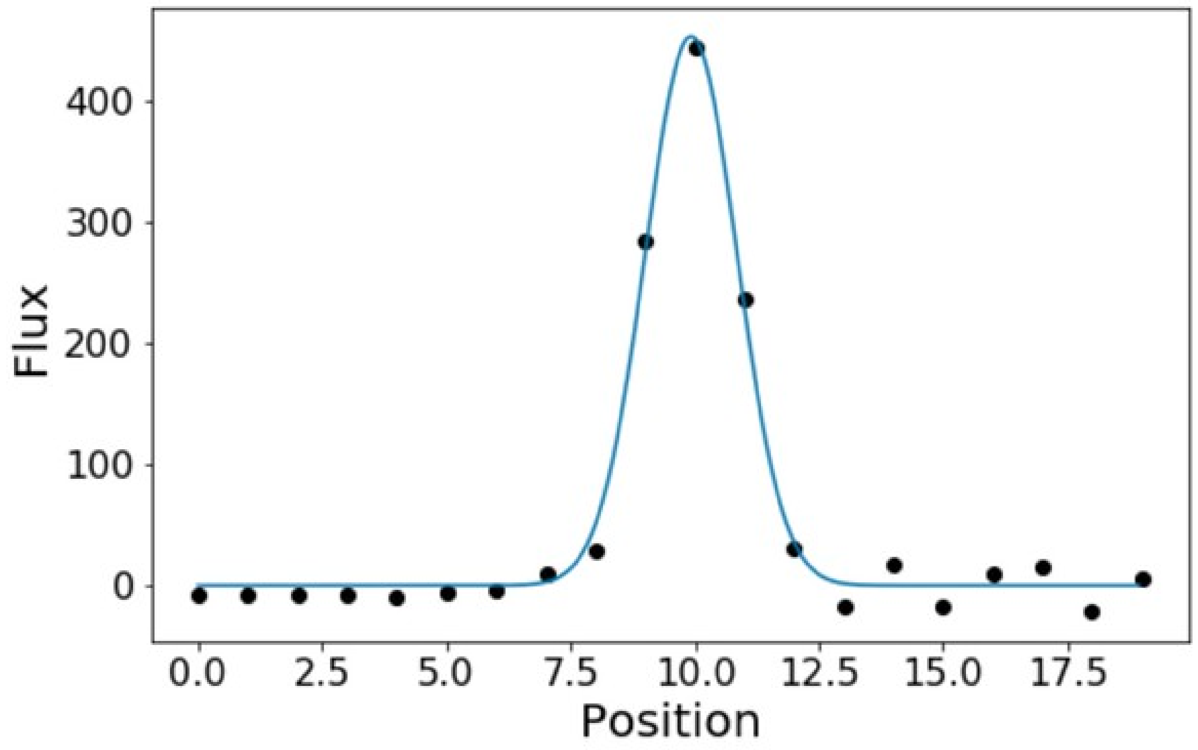
The utility takes as input a calibrated single pinhole flat-frame which displays the traces of the order centre locations:



From a spectral format table (specific to the arm in question) we know the minimum and maximum wavelength values for each order. Then using the first guess dispersion map generated by `soxs_disp_solution` we generate an array of approximate pixel locations on each order centre between these wavelength limits.



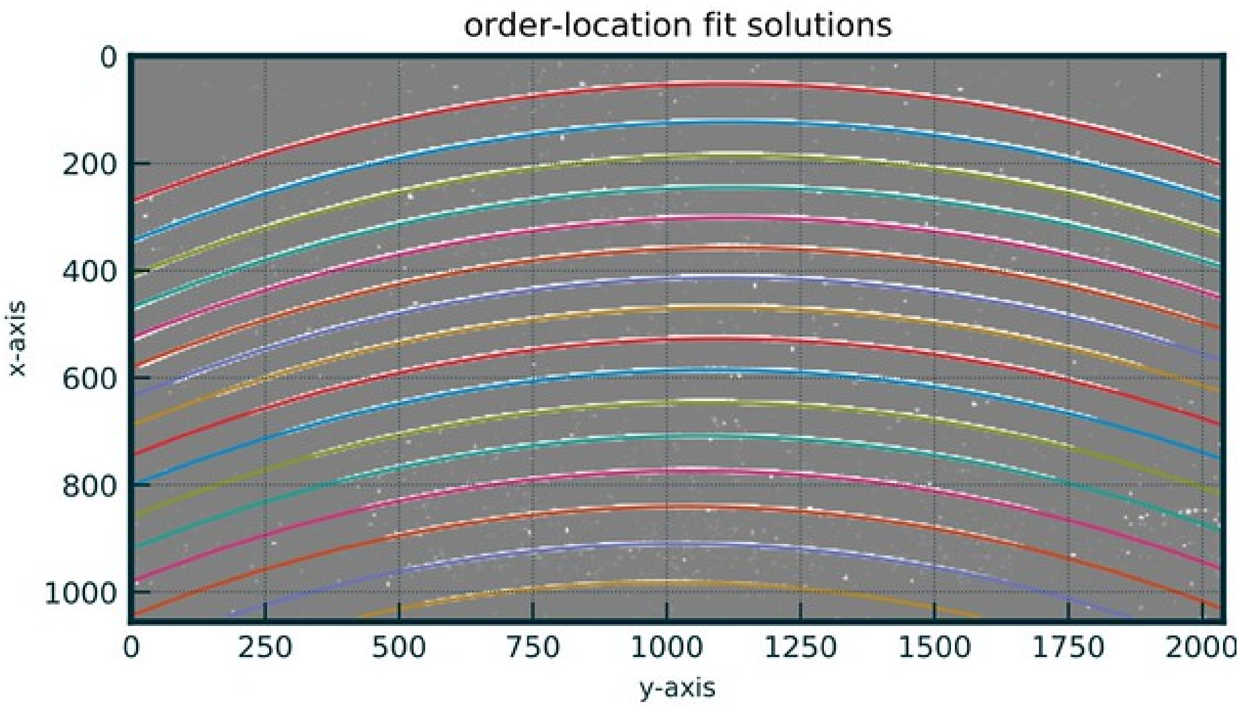
Centred on each pixel position we take a single-pixel wide image slice in the cross-dispersion direction N-pixels long (N is a recipe parameter). A 1D gaussian is fitted against the pixel slice and the peak pixel-position is stored.

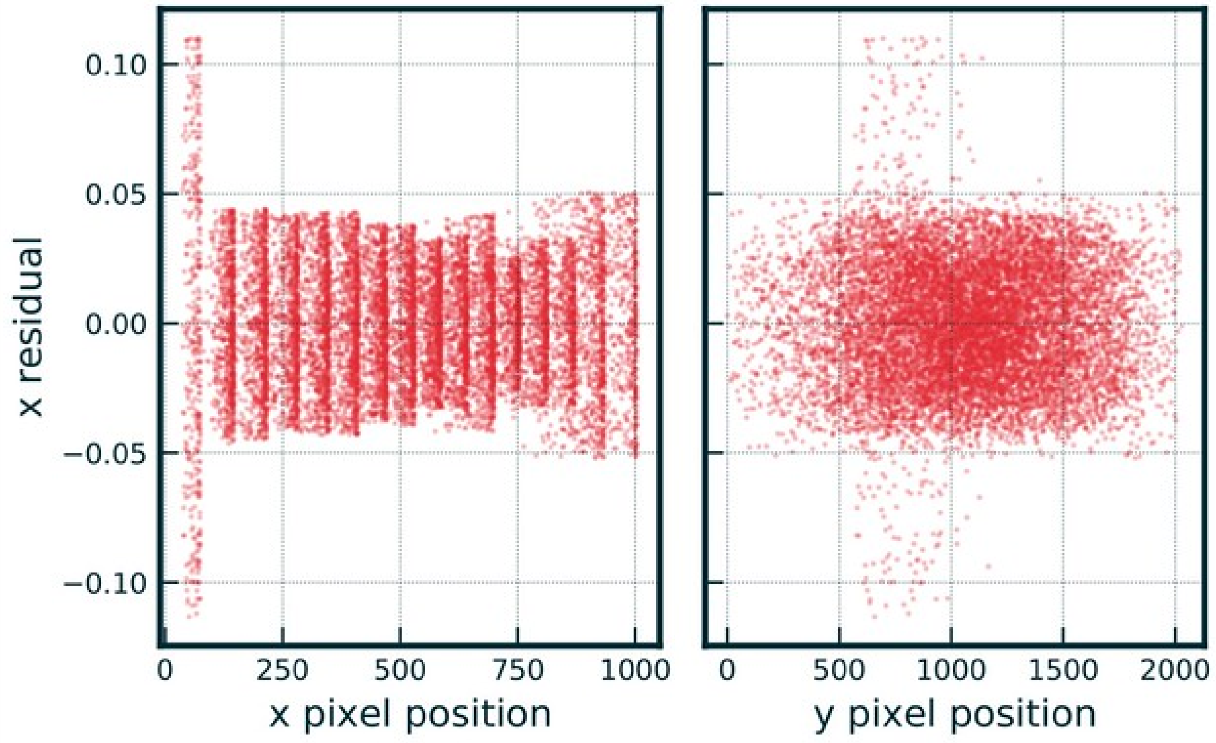


Finally, for each order, the set of gaussian peak pixel-positions is fitted with a low-order polynomial (X as a function of Y).

$$X = \sum_{i=0}^{n} c_i \times Y^i$$

Where n is the degree of the polynomials. Polynomials are iteratively fitted while sigma-clipping pixel-positions with outlying residuals. Results are stored in an order table.


order-location fit solutions
x-axis
y-axis
0
200
400
600
800
1000
0
250
500
750
1000
1250
1500
1750
2000

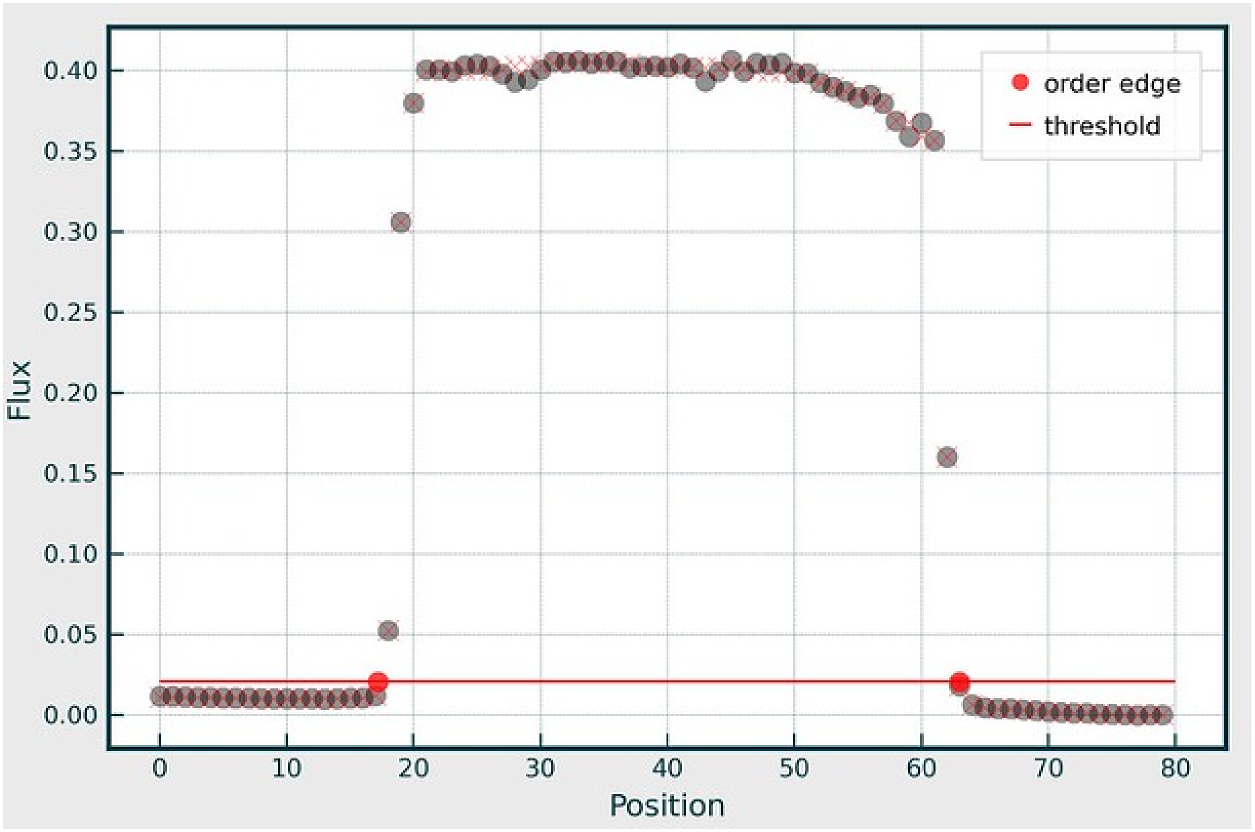

x residual
0.10
0.05
0.00
−0.05
−0.10
x pixel position
0
250
500
750
1000
y pixel position
0
500
1000
1500
2000

2.7.5 `detect_order_edges` – COMPLETED

The purpose of the `detect_order_edges` utility is to detect the edges of each order on the detector.

The utility takes as input a fully illuminated slit image, that is, a calibrated flat-field frame or a stacked flat-field frame (preferable as cosmic ray hits will be removed from the stack frame). The second input is the order table generated by the `soxs_order_centres` recipe which provides an estimation of a. the order locations, and b. the order shapes.

For each order an array of central pixel positions is generated. At each pixel position a N-pixel long, M-pixel wide image slice in the cross-dispersion direction (N and M are recipe parameters) is cut from the flat-frame. The slice is collapsed to a 1D array by taking the median value across its width (ignoring masked pixels). Further median smoothing is applied along the length of the slice to compensate of any rogue pixel values. Below you can see an example of a slice with the grey dot representing the collapsed 1D slice pixel values and the red crosses showing the median smoothed values.

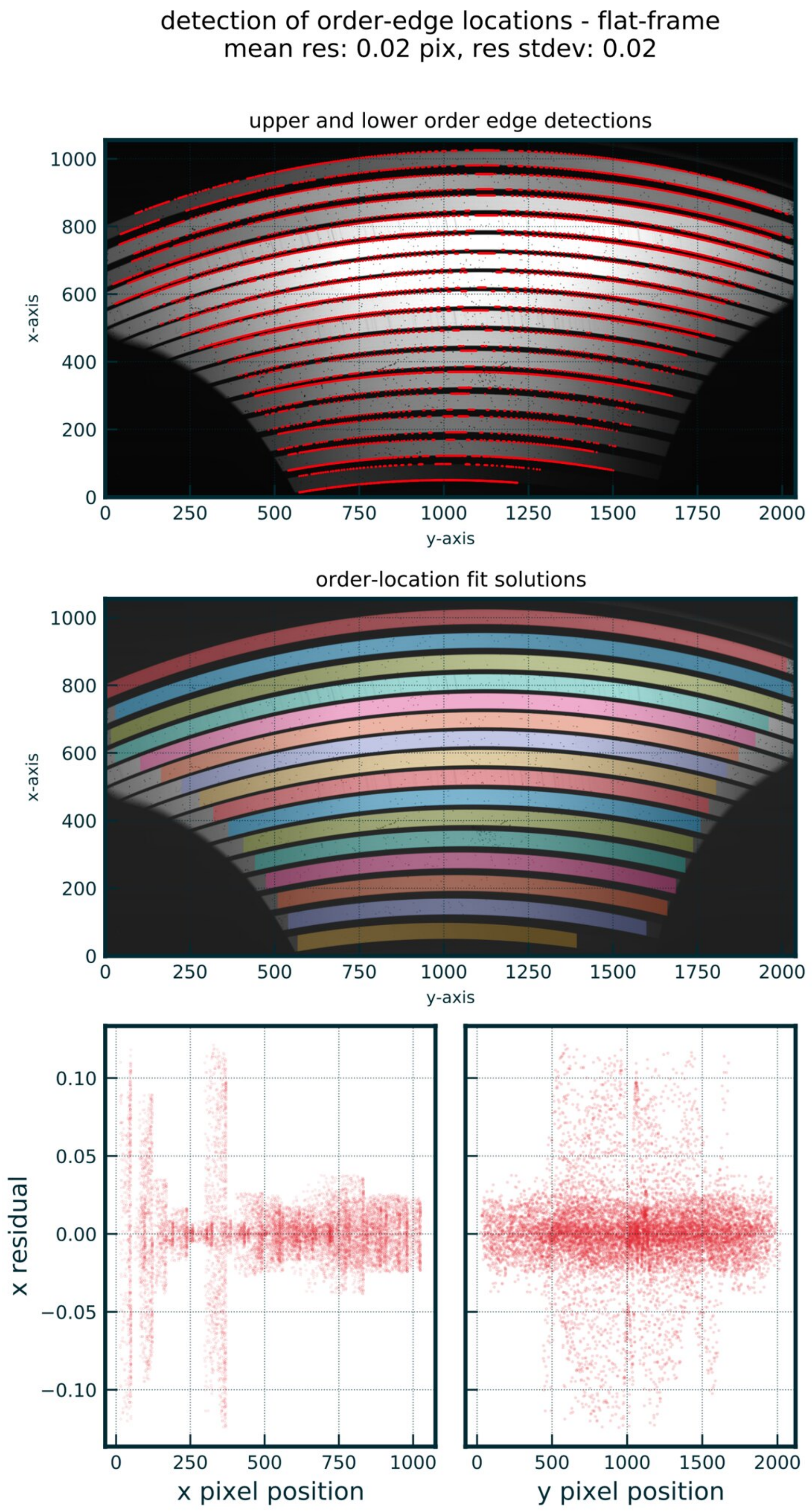


From the central 1D slice in each order the minimum and maximum fluxes are calculated to give a flux range. The absolute minimum and maximum flux thresholds are determined from the percentage thresholds given as recipe parameters. If the minimum threshold was set at 25% then the absolute flux would be:

$$threshold = minvalue + (maxvalue - minvalue) * 0.25$$

For each slice along each order the pixel-locations xmin and xmax along the order edge where the flux reaches this minimum flux threshold are detected and recorded (see red circles in the figure above). If the minimum flux is not recorded at any point along the order edge, the threshold is slowly incremented up to a maximum flux until the threshold is registered. Slices where both edges are not detected before the maximum flux threshold is reached are rejected.

Finally for each order the arrays of xmin and xmax pixel-positions are iteratively fitted with low order polynomials of Y as a function of X. The new order table is written to file and now includes the upper and lower-edge locations alongside the central location for each order.


detection of order-edge locations - flat-frame
mean res: 0.02 pix, res stdev: 0.02
upper and lower order edge detections
x-axis
y-axis
order-location fit solutions
x-axis
y-axis
x residual
x pixel position
y pixel position

2.7.6 `detector_lookup`

The purpose of the `detector_lookup` utility is to act as reference in the code for the various characteristics of the soxs detectors.

When initiated `detector_lookup` reads the detector characteristics from a Detector Parameters file and can later serve these parameters to the soxspipe code when requested.

class detector_lookup(*log*, *settings=False*)

return a dictionary of detector characteristics and parameters

Key Arguments:

- `log` – logger
- `settings` – the settings dictionary

Usage:

To initiate a detector_lookup object, use the following:

```
from soxspipe.commonutils import detector_lookup
detector = detector_lookup(
    log=log,
    settings=settings
).get("NIR")
print(detector["science-pixels"])
```

get(*arm*)

return a dictionary of detector characteristics and parameters

Key Arguments:

- `arm` – the detector parameters to return

2.7.7 `filenamer`

The purpose of the `filenamer` utility is to implement the file-naming scheme as laid out here. This util is usually called to determine a file name before writing a frame/table/product to disk.

2.7.8 `keyword_lookup`

The purpose of the `keyword_lookup` utility is to act as a lookup-reference in the code for specific SOXS FITS Header keywords.

When initiated `keyword_lookup` reads the keywords from a Keyword Dictionary file and can later serve the keywords names to the soxspipe code when requested.

class keyword_lookup(*log*, *instrument=False*, *settings=False*)

The worker class for the keyword_lookup module

Key Arguments:

- `log` – logger
- `settings` – the settings dictionary. Default *False*
- `instrument` – can directly add the instrument if settings file is not avalable. Default *False*

Usage

To initalise the keyword lookup object in your code add the following:

```
from soxspipe.commonutils import keyword_lookup
kw = keyword_lookup(
    log=log,
    settings=settings,
    instrument=False,
).get
```

After this it's possible to either look up a single keyword using it's alias:

```
kw("DET_NDITSKIP")
> "ESO DET NDITSKIP"
```

or return a list of keywords:

```
kw(["PROV", "DET_NDITSKIP"])
> ['PROV', 'ESO DET NDITSKIP']
```

For those keywords that require an index it's possible to also pass the index to the `kw` function:

```
kw("PROV", 9)
> 'PROV09'
```

If a tag is not in the list of FITS Header keyword aliases in the configuration file a `LookupError` will be raised.

get (*tag*, *index=False*)

given a tag, and optional keyword index, return the FITS Header keyword for the selected instrument

Key Arguments:

- `tag` – the keyword tag as set in the yaml keyword dictionary (e.g. 'SDP_KEYWORD_TMID' returns 'TMID'). Can be string or list of sttings.
- `index` – add an index to the keyword if not False (e.g. tag='PROV', index=3 returns 'PROV03') Default *False*

Return:

- `keywords` – the FITS Header keywords. Can be string or list of sttings depending on format of tag argument

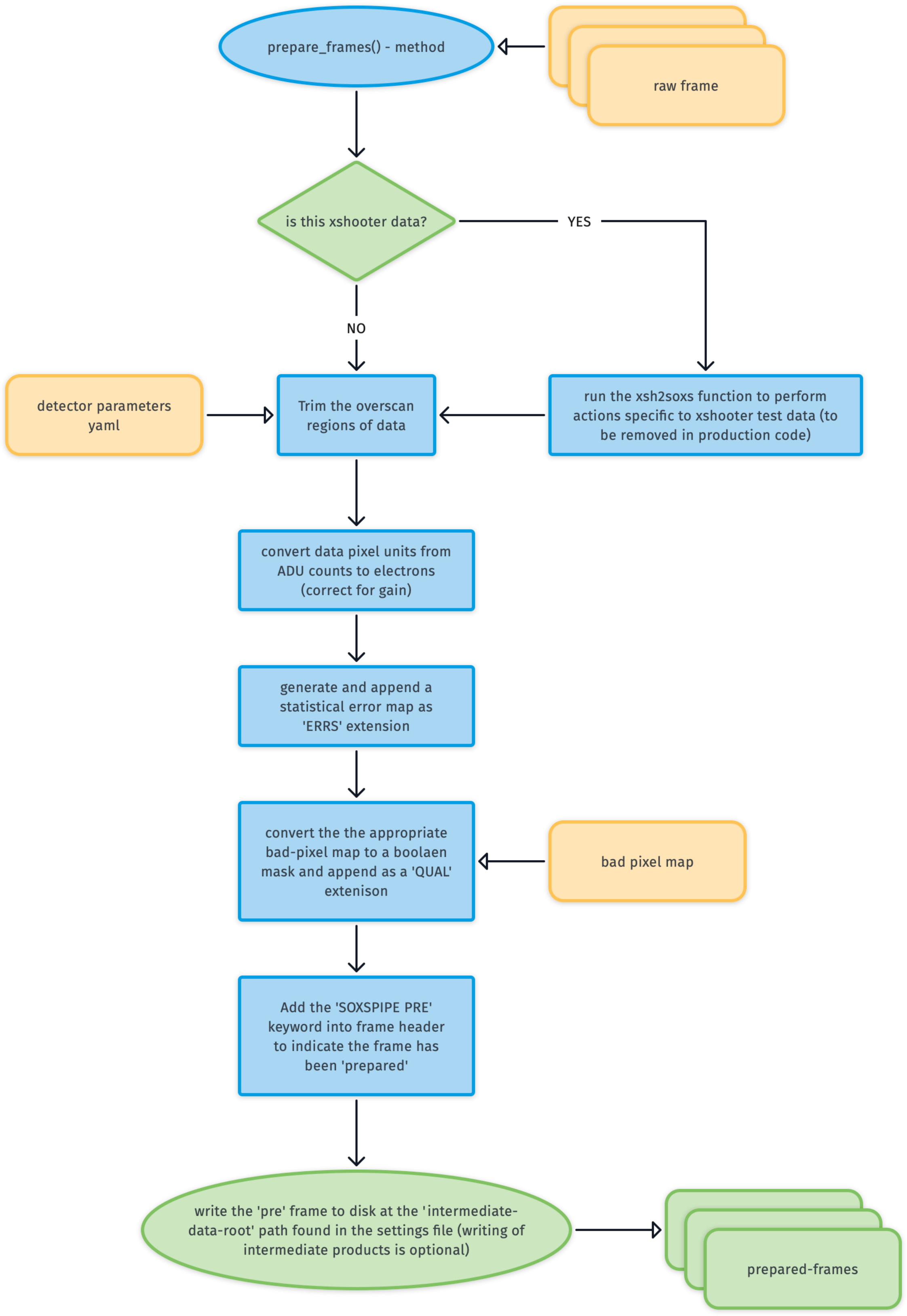
Usage

See docstring for the class

2.7.9 `prepare_frames`

The purpose of `prepare_frames` is to prepare the raw SOXS frames for data reduction.

Here's the typical workflow for preparing the raw frames:


prepare_frames() - method
raw frame
is this xshooter data?
YES
NO
detector parameters yaml
Trim the overscan regions of data
run the xsh2soxs function to perform actions specific to xshooter test data (to be removed in production code)
convert data pixel units from ADU counts to electrons (correct for gain)
generate and append a statistical error map as 'ERRS' extension
convert the the appropriate bad-pixel map to a boolaen mask and append as a 'QUAL' extenison
bad pixel map
Add the 'SOXSPIPE PRE' keyword into frame header to indicate the frame has been 'prepared'
write the 'pre' frame to disk at the 'intermediate-data-root' path found in the settings file (writing of intermediate products is optional)
prepared-frames

1. Trim Overscan

The first thing we need to do is trim off the overscan area of the image. The science-pixel regions for the detectors are read from the Detector Parameters file.

2. ADU to Electrons

Next the pixel data is converted from ADU to electron counts by multiplying each pixel value in the raw frame by the detector gain (the gain is read in units of electrons/ADU).

$$\text{electron count} = \text{adu count} \times \text{gain}$$

3. Generating an Uncertainty Map

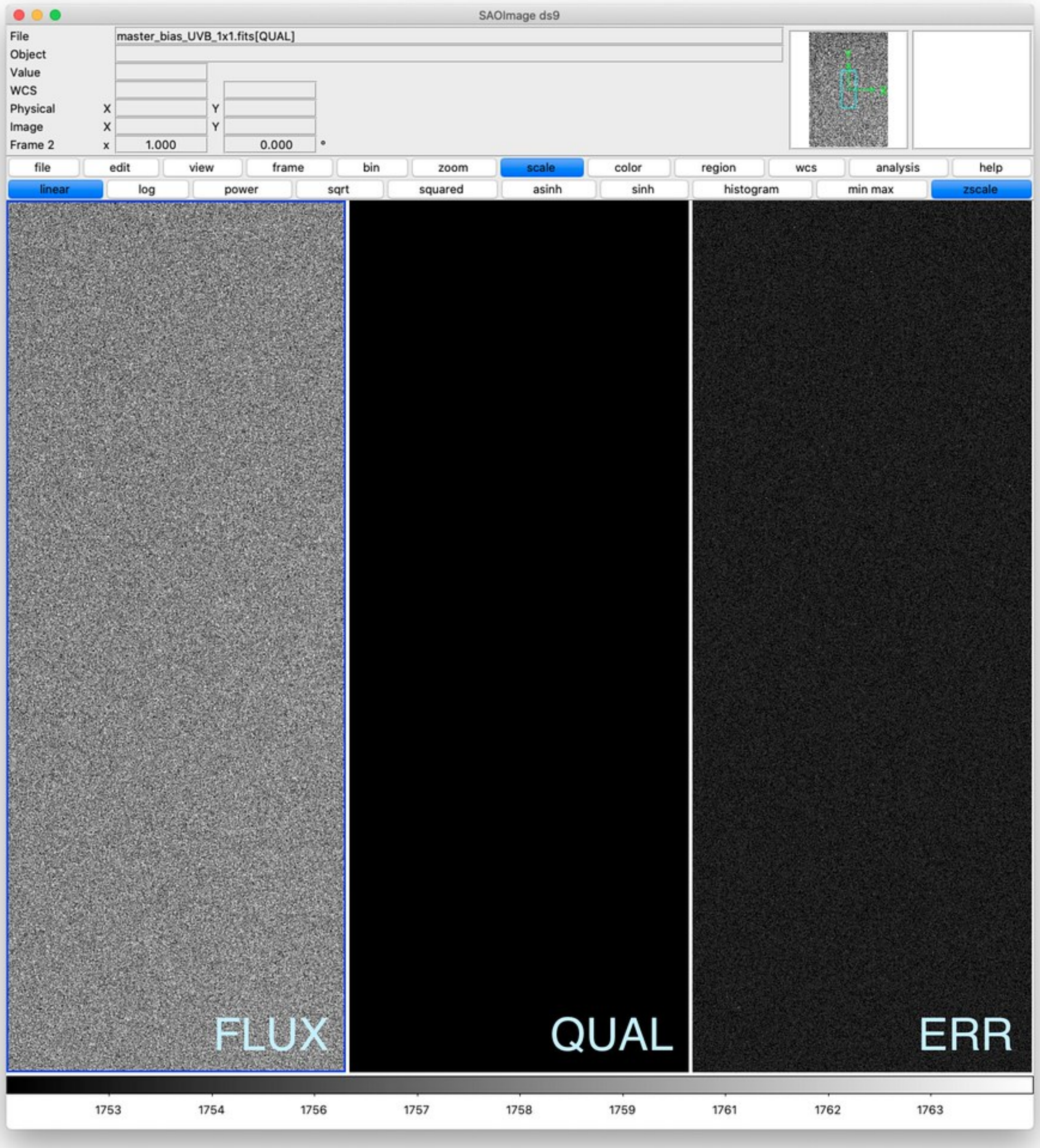
Next an uncertainty map is generated for the raw image and added as the 'ERR' extension of the image.

4. Bad Pixel Mask

The default detector bitmap is read from the static calibration suite and converted to a boolean mask, with values >0 becoming TRUE to indicate these pixels need to be masks. All other values are set to FALSE. This map is add as the 'QUAL' extesion of the image.

Finally the prepared frames are saved out into the intermediate frames location with the prefix `pre_`.

Viewing the image in DS9 (using the command `ds9 -multiframe -tile columns pre_filename.fits` to show all extensions as tiled frames) we can see the 'FLUX', 'QUAL' and 'ERR' extensions are now all present.



`_base_recipe_.`**`prepare_frames`**(*save=False*)

prepare raw frames by converting pixel data from ADU to electrons and adding mask and uncertainty extensions

Key Arguments:

- `save` – save out the prepared frame to the intermediate products directory. Default False.

Return:

- `preframes` – the new image collection containing the prepared frames

Usage

Usually called within a recipe class once the input frames have been selected and verified (see `soxs_mbias` code for example):

```
self.inputFrames = self.prepare_frames(
    save=self.settings["save-intermediate-products"])
```

2.7.10 `set_of_files`

The `set_of_files` utility helps to translate and homogenise various recipe input-frame lists. This allows recipes to accept any of the following inputs:

- an ESORex-like sof file,
- a directory of FITS files
- a list of fits file paths

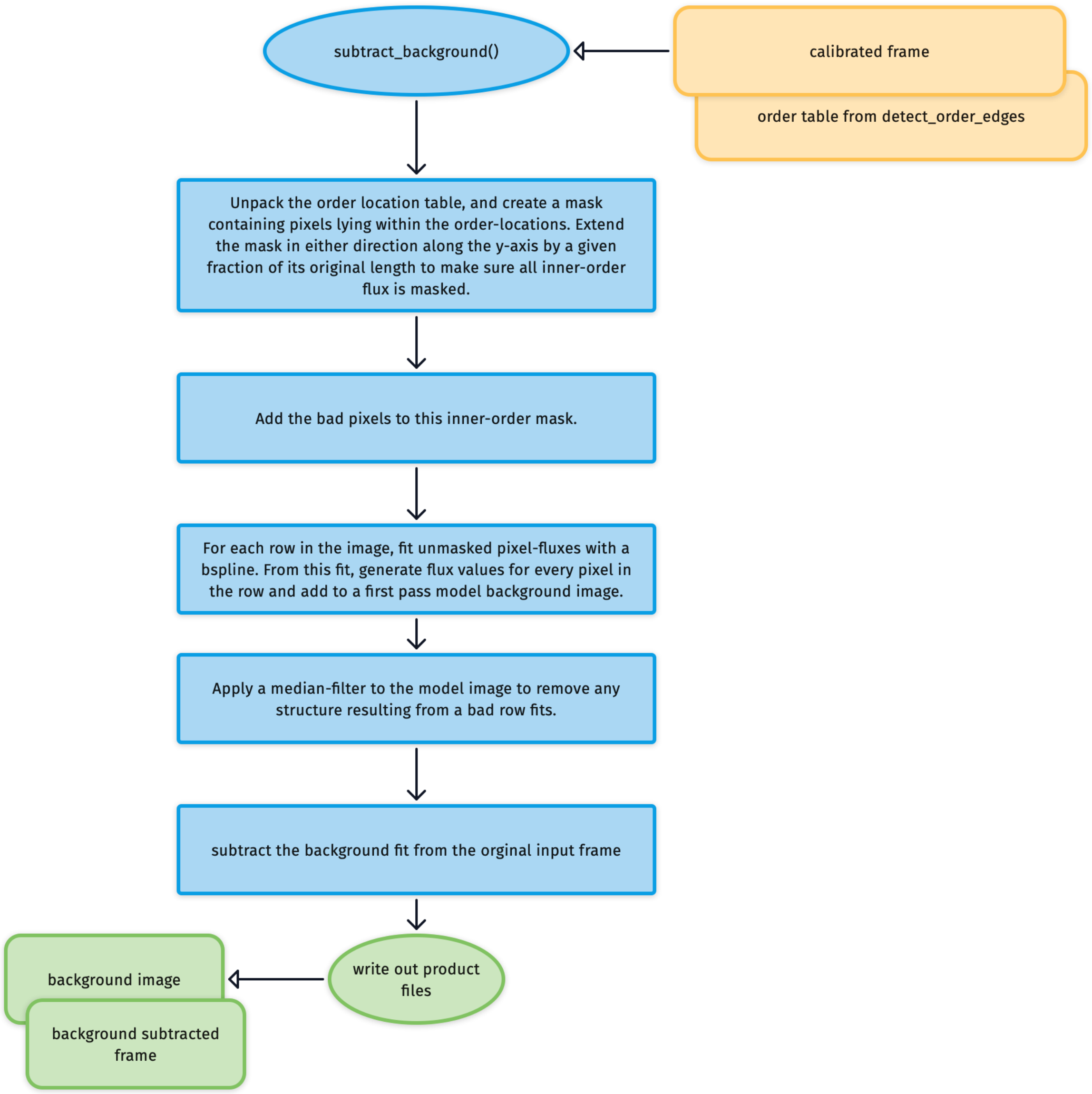
Behind the scenes `set_of_files` converts the lists into a CCDProc ImageFileCollection.

Lines in a sof file beginning with a `#` are considered as comments and therefore ignored by the pipeline.

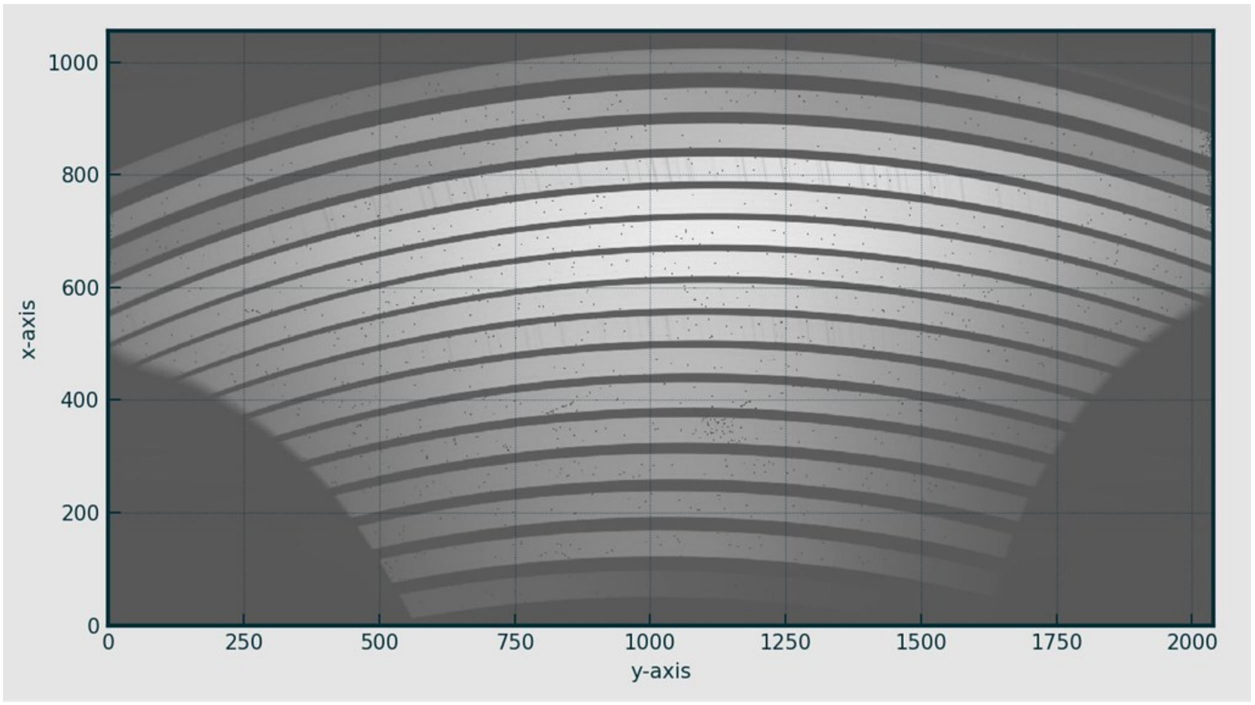
2.7.11 `subtract_background`

The purpose of the `subtract_background` utility is to model the topology of the scattered background light within an image and then remove it.

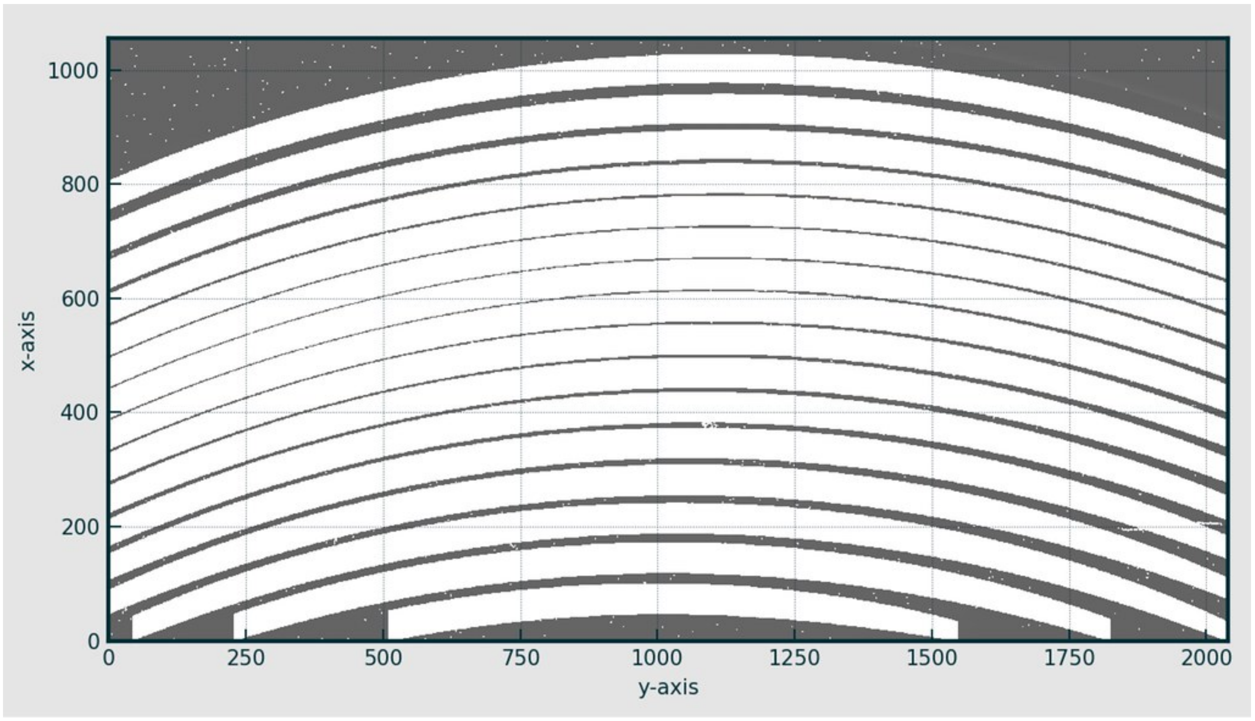
Here's the workflow for subtracting a frame's background:


subtract_background()
calibrated frame
order table from detect_order_edges
Unpack the order location table, and create a mask containing pixels lying within the order-locations. Extend the mask in either direction along the y-axis by a given fraction of its original length to make sure all inner-order flux is masked.
Add the bad pixels to this inner-order mask.
For each row in the image, fit unmasked pixel-fluxes with a bspline. From this fit, generate flux values for every pixel in the row and add to a first pass model background image.
Apply a median-filter to the model image to remove any structure resulting from a bad row fits.
subtract the background fit from the orginal input frame
write out product files
background image
background subtracted frame

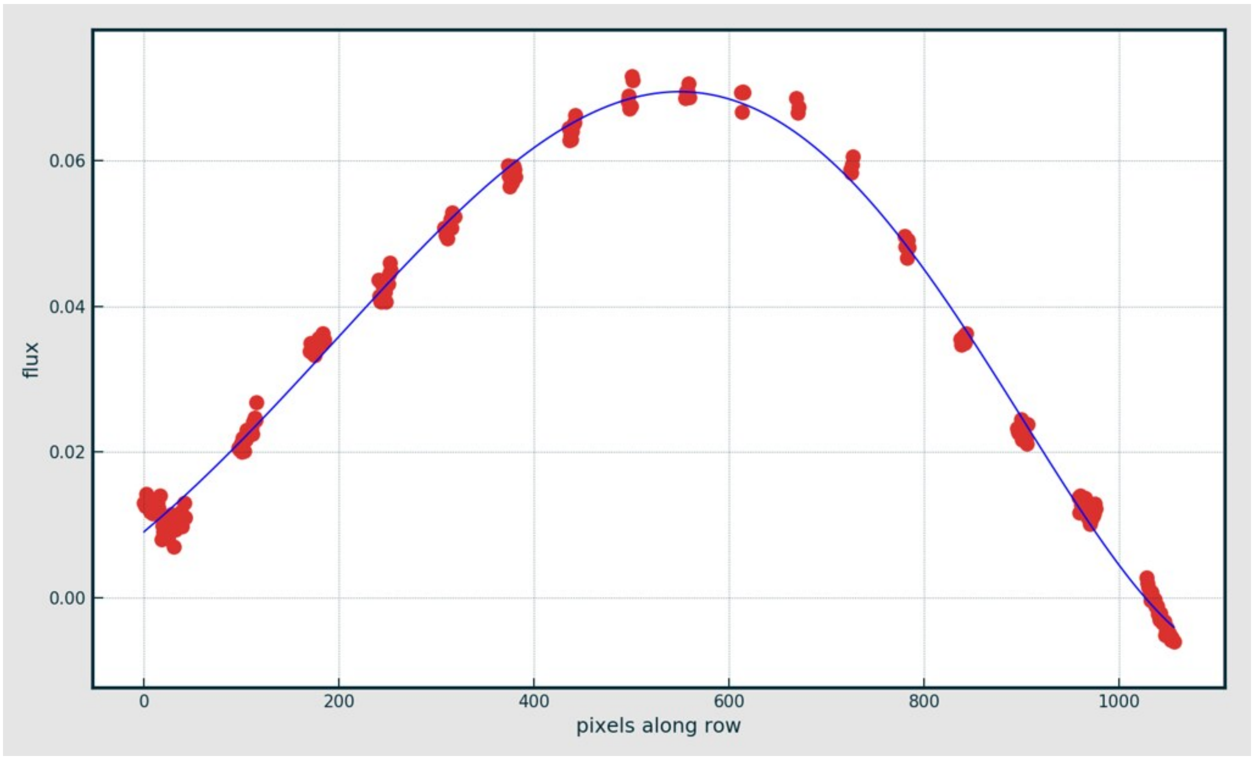
Here's an example frame requiring the background scattered light to be fitted and removed:



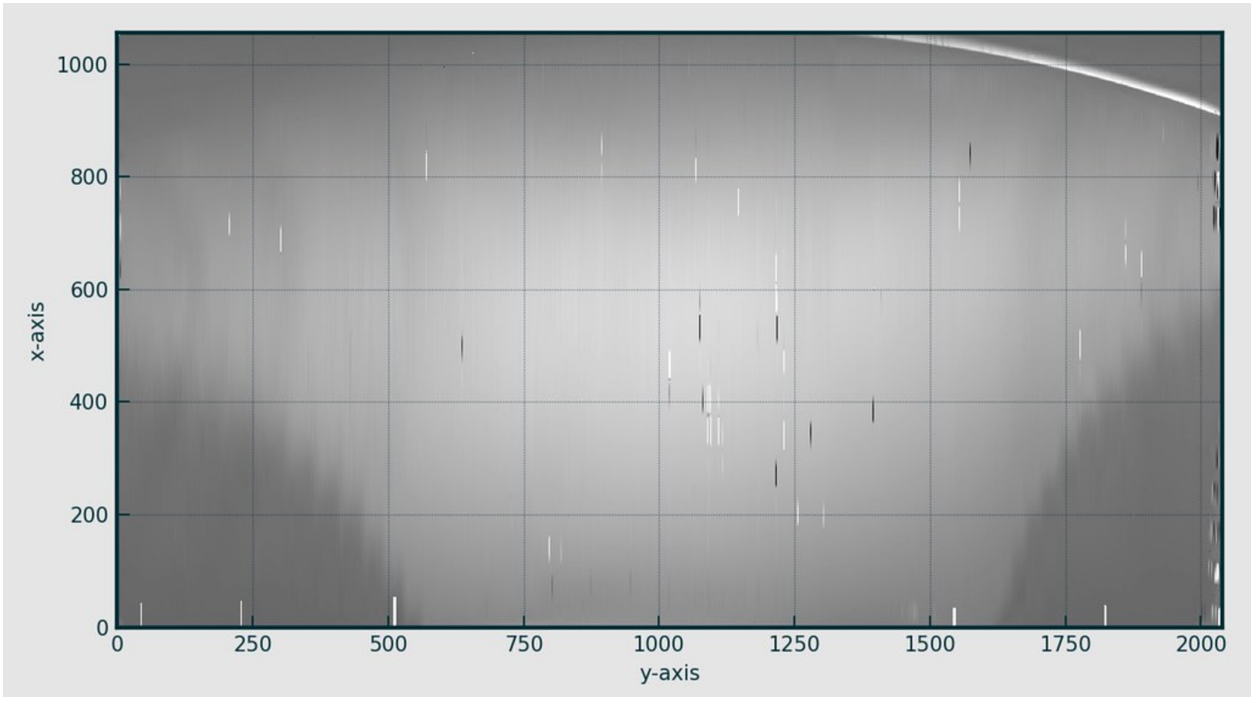
Having unpacked the order location table, a mask is created containing pixels lying within the order-locations. The mask is extended in either direction along the y-axis by a given fraction (recipe parameter) of its original length to make sure all inner-order flux is masked.



The bad-pixel mask is merged with this inner-order mask. For each row in the masked frame, a bspline is fitted to the unmasked pixel fluxes to model the shape of the scattered background light along the row.

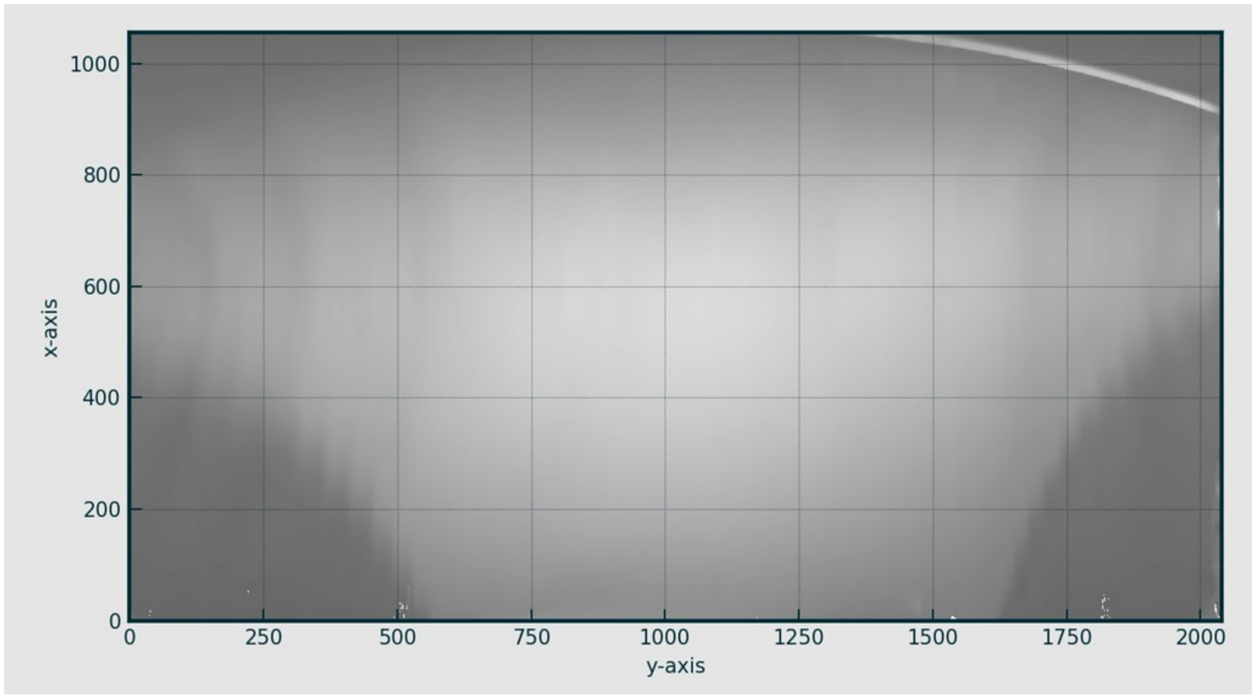


For each row, flux-values are generated for all pixels in the row using the bspline fit and added to a first pass model background image.

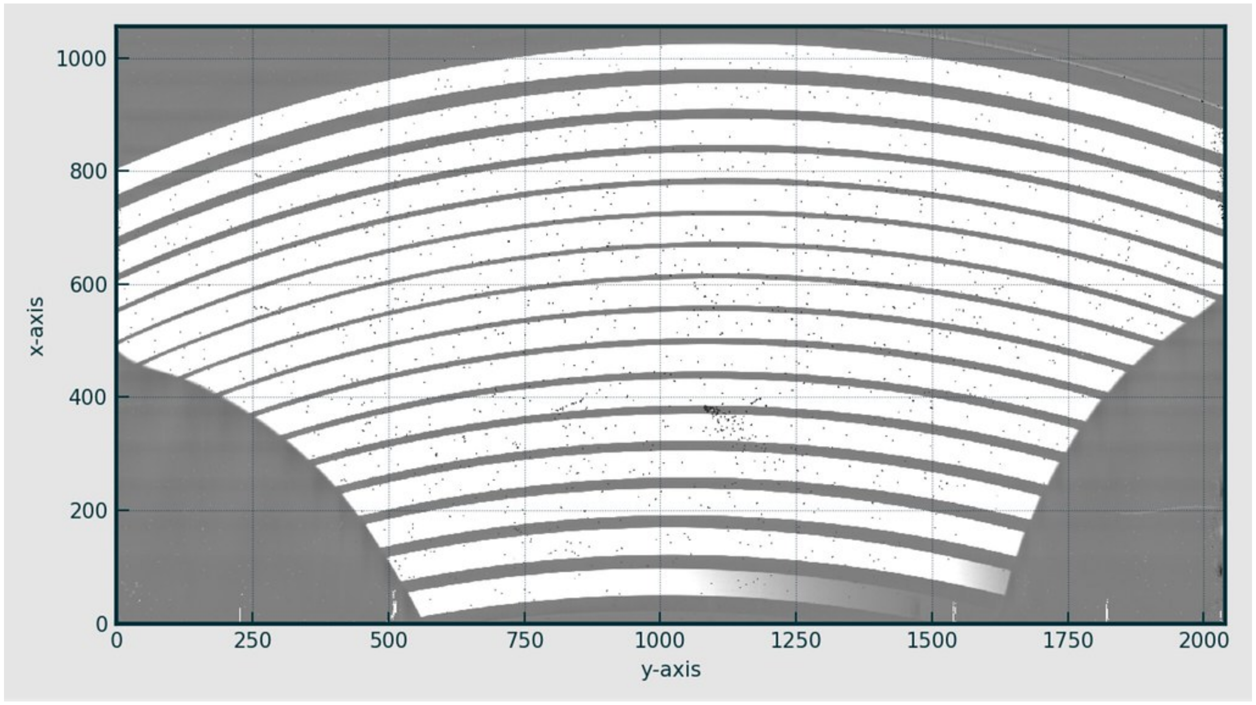


It's essential that background fitting is accurate within the order locations, but not outside of these areas, so there is no cause for concern if the fit is poor at the edges and corners of the image.

A median-filter is applied to the image to remove the structure resulting from a bad row fits (light and dark lines along the x-axis).



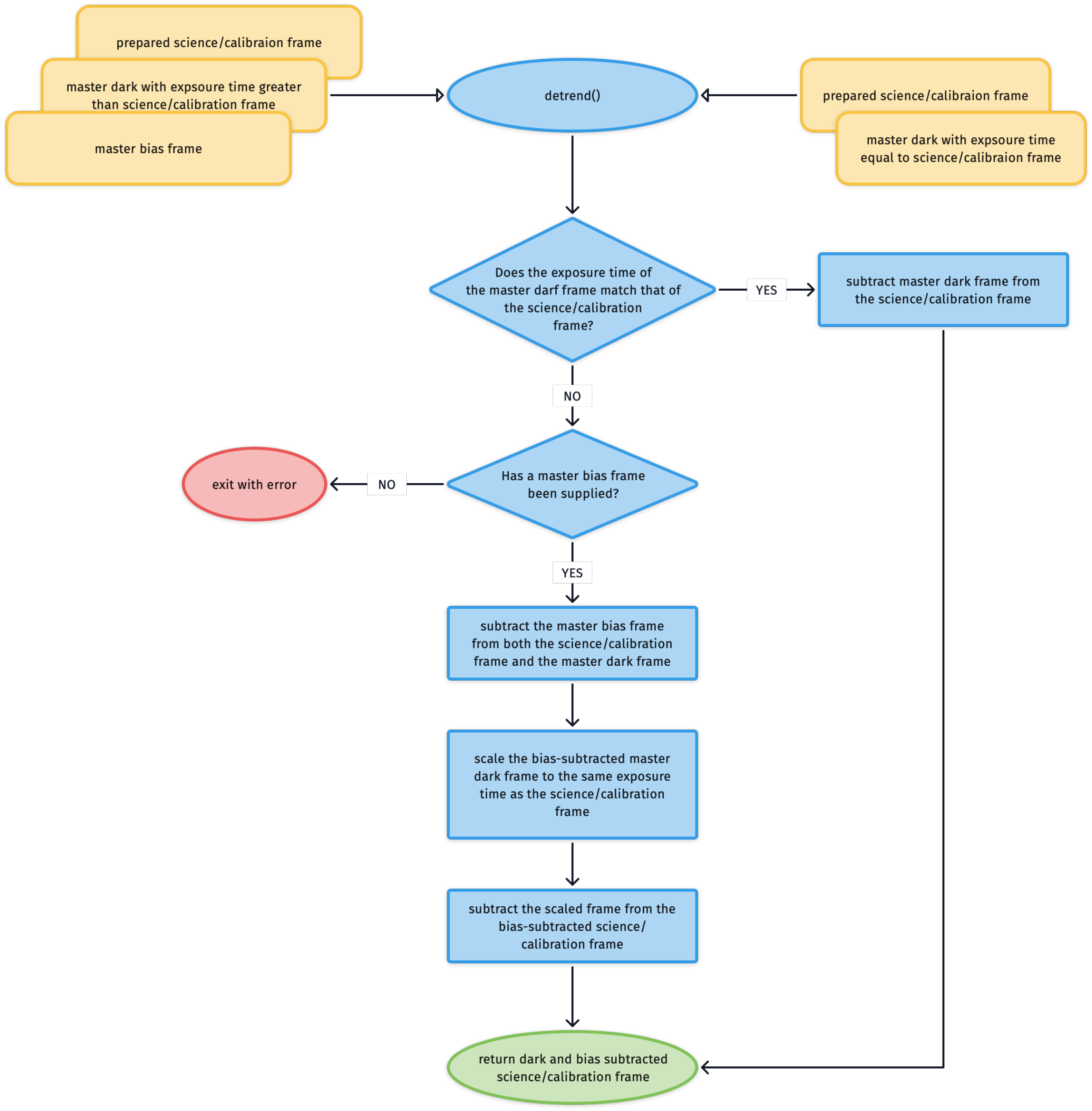
Finally, the modelled background image is subtracted from the original frame:



2.7.12 `detrend` - SEMI-COMPLETED

For the UVB-VIS arm we will often need to scale the master-dark frame to match the exposure time of our science/calibration frame. If this is the case than the master-bias frame needs to be subtracted from the master-dark frame before scaling. However, if the master-dark frame has the same exposure time as your science/calibration frame then it can be subtracted directly from frame as this serves to remove the bias and dark-current contributions simultaneously.

This logic is all housed within the `detrend` method.


prepared science/calibraion frame
master dark with expsoure time greater than science/calibration frame
master bias frame
detrend()
prepared science/calibraion frame
master dark with expsoure time equal to science/calibraion frame
Does the exposure time of the master darf frame match that of the science/calibration frame?
YES
subtract master dark frame from the science/calibration frame
NO
Has a master bias frame been supplied?
NO
exit with error
YES
subtract the master bias frame from both the science/calibration frame and the master dark frame
scale the bias-subtracted master dark frame to the same exposure time as the science/calibration frame
subtract the scaled frame from the bias-subtracted science/ calibration frame
return dark and bias subtracted science/calibration frame

2.8 Files

Files generated by SOXSPIPE follow a strict naming scheme.

2.8.1 Product Files

2.8.2 Static Calibration Files

2.8.2.1 Pinhole Map

The Pinhole Map is a static calibration file that provides a list wavelength λ, order number n and slit position s of the arc-lines from a multi-pinhole exposure alongside a first approximation of their (X, Y) pixel-positions on the detector.

2.8.2.2 Detector Parameters

This is a yaml file hosting the fixed characteristics of the soxs detectors such as gain, science-pixel indexes and image orientations.

Having these characteristics in a plain-text yaml file allows them to be abstracted from the main code and helps maintain a single-source of truth.

2.8.2.3 Spectral Format Table

The spectral format tables are static calibration files that provide some overview information about how each order typically presents itself on the plain of the detector. Useful information includes the minimum and maximum wavelength limits for each order. Here is an example of the XShooter UVB spectral format table contents:

```
ORDER,LAMP,WLMINFUL,WLMIN,WLMAX,WLMAXFUL,DISP_MIN,DISP_MAX,LFSR,UFSR
13,QTH,536.57666,544.63275,590.0,603.15594,19.0,2999.0,547.52716,591.32935
14,QTH,499.57135,505.7841,550.3,558.4777,120.0,2999.0,509.7667,547.52716
...
```

2.8.3 Intermediate Files

2.8.3.1 Master Bias

The master bias frames are the product of the `soxs_mbias` recipe. The master-bias frame is to be subtracted from science/calibration frames to remove the contribution to pixel counts resulting from the bias-voltage.

2.8.3.2 Master Dark

The master dark frames are the product of the `soxs_mdark` recipe. The master-dark frame is to be subtracted from science/calibration frames to remove the contribution to pixel counts resulting from the dark-current.

2.8.3.3 Prepared Frame

2.8.3.4 Dispersion Map

In the case of `soxs_disp_solution` the output dispersion map hosts the coefficients (c_{ij}) to two polynomials that describe the global dispersion solution for the entire detector frame:

$$X = \sum_{ij} c_{ij} \times n^i \times \lambda^j$$

$$Y = \sum_{ij} c_{ij} \times n^i \times \lambda^j$$

`soxs_spatial_solution`, building from this dispersion solution, provides global dispersion *and* spatial solution. The dispersion map output by this recipe hosts the coefficients (c_{ijk}) to two polynomials (note now the inclusion of slit position):

$$X = \sum_{ijk} c_{ijk} \times n^i \times \lambda^j \times s^k$$

$$Y = \sum_{ijk} c_{ijk} \times n^i \times \lambda^j \times s^k$$

2.8.3.5 Order Table

The order table gets built over `soxs_order_centres` and `soxs_mflat` recipes. The final product contains polynomial fits for the order centres and the upper and lower edges of the order locations on the detector.

Here is an example of the content of the NIR order table from the `soxs_order_centres` recipes.

```
order,degy,CENT_c0,CENT_c1,CENT_c2,CENT_c3
11,3,787.9616,0.387875383,-0.000175467653,7.67345562e-10
12,3,714.084224,0.392816612,-0.000177468713,1.24058095e-09
...
```

Todo:

- add order table example after soxs_mflat

2.9 Release Notes

2.9.1 v0.10.2 - April 23, 2024

- **ENHANCEMENT:** the calibration lamp name is now added to the sof filenames, and hence the product file names
- **ENHANCEMENT:** file summary now shows which calibration lamps are used
- **ENHANCEMENT:** adding bad-pixel maps for SOXS detectors (currently blank)
- **ENHANCEMENT:** pipeline can select default settings for either soxs or xsh (previously there was only one default settings file)
- **FIXED**: SOXS VIS darks are now getting split by EXPTIME in data-organiser

2.9.2 v0.10.1 - April 11, 2024

- **FEATURE:** the data-organiser has been 'plumbed' to work with SOXS data (will now work with Xshooter or SOXS data).
- **ENHANCEMENT:** clipping of entire MPH set based on their combined RMS scatter from their predicted locations. MPH sets with large scatter are consider poor and removed before polynomial fitting.
- **ENHANCEMENT:** option added to relevant recipes settings to allow toggling of fitting and subtracting of intra-order scattered background light (`subtract_background`)
- **REFACTOR**: added arm and lamp to QC plot titles.
- **REFACTOR** recipe settings can now be set independently for each arm.
- **REFACTOR:** fitting of the scatter background light is now much more robust.
- **REFACTOR:** The scattered light background images are now saved as QC PDFs instead of FITS frames.
- **FIXED**: fixed issue where logs were getting duplicated.
- **FIXED**: the scaling and stitching together of the UVB D2 and QTH lamp flats.

2.9.3 v0.10.0 - February 20, 2024

- a `bootstrap_dispersion_solution` has been added to the advanced settings. It this is set to True, the pipeline will attempt to bootstrap the initial dispersion solution (using the static line list) with lines from a line-atlas. The line-atlas contains more lines and lines with more precise wavelength measurements.
- **FEATURE**: a new 'reducer' module and terminal command replace the old `_reduce_all/sh` script. This allows the data-organiser to dynamically self-correct if a recipe fails.
- **ENHANCEMENT:** robustness fixes and updates.
- `pinhole_fwhm_px_min` and `pinhole_fwhm_px_max` settings added to `soxs-spatial-solution`. Detected pinholes with a FWHM below/above these values get clipped.
- **FIXED**: The bad-pixel mask from the noise map of the mbias frame is now injected mbias product. The Xshooter UVB electron trap is now clearly visible in the master bias quality extension.
- `mph_line_set_min` setting added to `soxs-spatial-solution`. Full multi-pinholes sets (same arc line) with fewer than mph_line_set_min lines detected get clipped.

2.9.4 v0.9.9 - January 24, 2024

- **FIXED**: bug fix logger

2.9.5 v0.9.8 - January 19, 2024

- **FIXED**: bug fix in collecting settings files from the default location

2.9.6 v0.9.7 - December 7, 2023

- **ENHANCEMENT:** the instrument name is now included in the SOF & product filename.
- **ENHANCEMENT:** setting bad pixels to zero in sky-subtracted product frames.
- **FIXED**: blocking filters now taken into account when building the master-flats and determining the order-edges.
- **FIXED**: master flats taken with blocking filters are no longer matched with multi-pinhole frames fro the spatial solution recipe.
- **FIXED**: master flats with identical slit-widths now matched against science frames within the data-organiser when building SOF files.
- **FIXED**: the order of the columns in the extracted & merged spectrum tables is now WAVE, FLUX (was FLUX, WAVE).
- **FIXED**: specutils dependency added to conda-forge requirements.

2.9.7 v0.9.4 - December 5, 2023

- **REFACTOR:** orders are now clipped so that only the pixels deemed to be within the flux receiving regions of the order are extracted (according to the static calibration spectral format table).
- **REFACTOR:**`soxspipe prep` will now warn user if no FITS files are found in the workspace directory and quit before moving any files (safer).
- **REFACTOR:**`soxspipe session` commands will look for a sessions directory before creating any new files and folders (cleaner).
- **REFACTOR:**`read_spectral_format` function can now return limits to the usable region of each spectral order if a dispersion map is given.
- **FIXED**: fixes to make detect_continuum more robust.

2.9.8 v0.9.2 - November 29, 2023

- **ENHANCEMENT:** intra-order background (scattered light) fits are now being written to FITS image files in the QC directories and reported at the end of a recipe run.
- **ENHANCEMENT:** added a `create_dispersion_solution_grid_lines_for_plot` function to allow adding dispersion solution grid to QC plots. This is extremely useful for quickly diagnosing problems with the fits.
- **REFACTOR:** All product FITS files now pass fitverify without error or warnings. All issues were due to using '-' instead of underscores in FITS binary table column names.
- **REFACTOR:** bad-pixel values set to 0 in data extensions of products
- **REFACTOR:** nans have been replaced by zero in FITS image product
- **FIXED**: a mismatch between daofind results and the original input pixel table was causing dispersion solution to break (a recent bug introduced during code optimisations)
- **FIXED**: the internal soxspipe logger was being interfered with by astropy so that logs were sometimes getting redirected to the wrong place

2.9.9 v0.9.0 - October 11, 2023

- **FEATURE:** added a `predict_product_path` function to determine the product path from a recipe's input sof file
- **FEATURE:** Merging of individual order extracted spectra from object frame into a single spectrum for each arm
- **FEATURE:** Object spectra are now extracted from the sky-subtracted frames using the Horne 86 method
- **FEATURE:** Real SOXS data is now included in the unit-test suite (starting to replace simulated data unit-tests). `soxs-disp-solu` recipe so far.
- **FEATURE:** SOXS NIR Xe line-lists added to static-calibration suite (single and multi pinhole).
- **FEATURE:** when running a recipe, `soxspipe` writes informative logs to stdoutAND to a log file adjacent to the recipe's product file(s). Error logs are also written if a recipe fails (see docs).
- **ENHANCEMENT:** recipe timing added to the end of the logs
- **ENHANCEMENT:** fitted lines from the dispersion solution are written out to file as a QC product
- **ENHANCEMENT:** flux (and other daostarfinder metrics) are now recorded in the detected line-list QC file. This will help measure degradation of arc-lamps over time.
- **ENHANCEMENT:** FWHM and pixel-scale added to fitted lines from the dispersion solution
- **ENHANCEMENT:** legends added to many of the QC plots
- **ENHANCEMENT:** OB ids now getting add to the data-organiser database tables.
- **ENHANCEMENT:** object trace FITS binary table added to stare-mode products (alongside complimentary QC plot)
- **ENHANCEMENT:** products and QC outputs are differentiated in the table reported upon recipe completion (see label column).
- **ENHANCEMENT:** verifying the master flat used to calibrate object/std spectra has the same slit-witdh as used to take the science frames
- **REFACTOR:**`init` command has been subsumed into the prep command. The `prep` command will generate a settings file to live within the prepared workspace.
- **REFACTOR:**`misc/` directory created by data-organiser even if empty
- **REFACTOR:** close matplotlib plot after writing plots to file
- **REFACTOR:** command-line startup speeds improved
- **REFACTOR:** continuum fitting code made more robust against edge cases (orders of the fit are automatically reduced if fit does not converge)
- **REFACTOR:** soxspipe now has a 'full' and a 'lite' test-suite. Using the lite suite will speed up deploying of new releases.
- **DOCS:** updated docs with a more robust SOXSPIPE upgrade path (users having issue with `conda update ...`)
- **FIXED**: sky-subtraction code and data-organiser fixed to work with binned data

2.9.10 v0.8.0 - May 18, 2023

- **FEATURE:** we now have a data-organiser to sort data, prepare the required SOF files and generate reduction scripts.
- **ENHANCEMENT:** '.db', '.yaml', '.sh' and '.log' extensions skipped when moving items to the misc folder
- **ENHANCEMENT:** move information printed to STDOUT when preparing a workspace to inform the user of how the data is organised
- **ENHANCEMENT:** code can automatically adjust polynomial fitting parameters to find a dispersion solution if those provided in the settings file fail.
- **ENHANCEMENT:** uncompression of fits.Z files (if any) occurs before data-organising
- **REFACTOR:** speed & robustness improvements to dispersion solution to 2D image map conversion.
- **REFACTOR:** much fast check for product existence so recipes are quickly skipped if they have already run.
- **REFACTOR:** removed the `intermediate-data-root` setting renamed to a more accurate `workspace-root-dir`
- **REFACTOR:** removed the `reduced-data-root` setting.
- **REFACTOR:** updating all depreciated pandas commands so pipeline is now compatible with 1.X and 2.X versions of pandas
- **FIXED** pandas 1.X and pandas 2.X were doing different things when renaming columns in data-frames. Both 1.X and 2.X now work in the pipeline.

2.9.11 v0.7.2 - March 3, 2023

- **REFACTOR:** Big improvements on sky-subtraction
- **REFACTOR:** UV order-edge detection more robust
- **REFACTOR:** changed quickstart guide compress to gzipped tar
- **REFACTOR:** updated default settings to be more robust

2.9.12 v0.7.1 - November 4, 2022

- **FEATURE:** UV D-Lamp and QTH-Lamp master flats now being stitched together
- **FEATURE:** errors in error maps now being treated correctly and propagating to combined images
- **FEATURE:** Pipeline can now 'remember' where it left off in the reduction cascade. If it has run a recipe before it will exit with a message to the user informing them how to force the recipe to rerun.
- **FEATURE:** added a `twoD_disp_map_image_to_dataframe` function to toolkit
- **ENHANCEMENT:** PRO CATG now written to product FITS header
- **ENHANCEMENT:** Handling of binned images when generating flats and order-locations
- **ENHANCEMENT:** Where possible, product files are given the same name as the SOF file used to generate them (replacing `.sof` extension with `.fits`)
- **ENHANCEMENT:** SOF files can now contain a file 'tag' to allow users to read the SOF file contents and know exactly which files are being passed to the recipe (e.g. `MASTER_BIAS_UVB`, `LAMP,DORDERDEF_UVB` ...)
- **ENHANCEMENT:** dispersion solution now working with simulated NIR SOXS data

- **ENHANCEMENT:** quicklook now renders dispersion solution grid
- **ENHANCEMENT:** ~40% speed gain in combining images.
- **REFACTOR:** 2D Map generation now ~6-8 times faster (seeding solutions with nearest neighbour with cubic spline method)
- **REFACTOR:** SOF filenames reworked to contain the UTC observation date instead of MJD (more in-line with ESO ecosystems)
- **REFACTOR:** updated workflow for master bias combination
- **REFACTOR:** updated workflow for master dark combination
- **REFACTOR:** QC PDF plots now added to their own directory separate from the products
- **REFACTOR:** products now sub-divided into recipe directories (e.g. `./products/soxs-mbias/`)
- **DOCS:** mflat docs brought up-to-date
- **DOCS:** mflat docs brought up-to-date
- **FIXED:** mflat recipe now exits if flat frames are not of a consistent exptime.

2.9.13 v0.6.2 - April 13, 2022

- **ENHANCEMENT:** quickstart guide added for calibration recipes
- **FEATURE:** QCs added for dispersion solution and order centre recipes
- **REFACTOR:** clean up of stdout information

2.9.14 v0.6.1 - April 11, 2022

- **FEATURE:** shipping static calibration files with the code (one less thing for end-users to install and set-up)

2.9.15 v0.6.0 - April 10, 2022

This is only a summary of some of the updates included in this release:

- **ENHANCEMENT:** All CSV files moved to FITS binary tables - metadata very useful for developing data organiser
- **FEATURE:** 2D image map now created by create_dispersion_solution `subtract_calibrations` util renamed to `detrend` and added ability to flat correct
- **FEATURE:** 2D image map of wavelength values, slit-position values and order values written alongside polynomial solutions of full dispersion solution
- **FEATURE:** soxspipe now on conda
- **FEATURE:** QCs now being written to FITS header
- **FEATURE:** adding QC and product collection in mbias recipe
- **ENHANCEMENT** RON and bias structure QCs now reported by mbias
- **ENHANCEMENT** nan ignored when scaling quicklook images
- **ENHANCEMENT** RON and bias structure QCs now reported by mdark
- **ENHANCEMENT:** QCs have an option to *NOT* (`to_header`) write to FITS header (default is to write)

- **REFACTOR:** better treatment of masked pixels when stacking images (e.g. in mbias and mdark)
- **REFACTOR:** removed raw frame reports and neater QC table
- **REFACTOR:** fits header keywords neatly sorted before writing to file
- **FIX:** Correct management of mask when determining RON on bias and darks

2.9.16 v0.5.1 - September 29, 2021

- **FEATURE:** recipes now have a `qc` and `products` attribute. These are pandas data frames used to collect QCs and generated products throughout the life-time of the recipe. They are printed to STDOUT at the end of the recipe (can be used in the future to send post request to health monitor API with JSON content in request body).
- **ENHANCEMENT** added code-base to conda-forge
- **ENHANCEMENT** added bottleneck to the install requirement (makes image combination more efficient)
- **ENHANCEMENT** masked pixel now coloured red in quicklook plots (easier to differentiate from good pixels)
- **ENHANCEMENT** low-sensitivity pixels in lamp-flats now identified and added to bad-pixel mask
- **ENHANCEMENT** add a verbosity flag to the command-line and a verbose parameter to each recipe
- **REFACTOR** inter-order pixel value in flats now set to unity (instead of running background fitting and subtraction)
- **REFACTOR:** recipes now have their recipe name as a `recipeName` attribute

2.9.17 v0.5.0 - June 10, 2021

- **FEATURE** Added a new `filenamer` module that implements a strict intermediate and reduced file-naming scheme
- **FEATURE:**`soxs_mflat` recipe now included
- **FEATURE:**`soxs_spatial_solution` recipe is now included
- **FEATURE:**`subtract_background` utility added
- **FEATURE:** added a `detect_order_edges` object
- **FEATURE:** Added a `dispersion_map_to_pixel_arrays` function to convert from order-based and wavelength arrays to pixel arrays (first guess dispersion map only so far)
- **FEATURE:** added a quicklook function in toolkit to quickly visualise a frame
- **FEATURE:** added a toolkit module for small functions used throughout soxspipe
- **FEATURE:** added function in toolkit to unpack an order table into lists of coordinates, one list per order.
- **FEATURE:** added image slice tool to toolkit
- **ENHANCEMENT** Added a `-o <outputDirectory>` switch to the command-line to optionally override the 'intermediate-data-root' setting in the settings file.
- **ENHANCEMENT:** added a fraction of a second tolerance when matching exptimes between darks and science/calibration frames
- **ENHANCEMENT:** y limits now added to the order table to show limits of order locations on detector
- **REFACTOR:** Change the "SOXSPIPE PRE" date stamp keyword to "SXSPRE" to future-proof for phase III (8 character keyword limit)

- **REFACTOR:** Pandas tables are now used through-out code to pass line-lists between methods
- **REFACTOR:** refactoring of polynomial fitting has made creation of dispersion maps ~50 times faster
- **REFACTOR:** removed OBID from file names and added readout mode. This information is more helpful at the glance.
- **FIXED:** correct binning reported in product file names
- **FIXED:** lines in a sof file beginning with a # are considered as comments and therefore ignored by the pipeline.

2.9.18 v0.4.1 - September 15, 2020

- **FEATURE:** add command-line util for soxs order_centres recipe
- **FEATURE** added the `detect_continuum` utility to fit order centre locations in single pinhole flat frames.
- **ENHANCEMENT:** added a supplementary file list for non-fits input files in set-of-file util
- **ENHANCEMENT:** adding more information residual plots & visualisation of fitting for disp solution
- **ENHANCEMENT:** check that files in the sof files exist before proceeding.
- **ENHANCEMENT:** added spectral format table lookup to detector settings file
- **REFACTOR:** moved chebyshev order/wavelength polynomials into its own class - decoupled from create_dispersion_map class

2.9.19 v0.4.0 - September 3, 2020

- **FEATURE:** added create_dispersion_map class to be used in `soxs_disp_solution` and `soxs_spatial_solution`
- **FEATURE:** added a `detrend` method to subtract calibration frames (bias and dark) from an input frame
- **FEATURE:** added the dispersion solution recipe and unit tests
- **FEATURE:** added the disp_solution command-line tool
- **DOCS:** major docs overhaul
- **ENHANCEMENT:** added predicted lines lists to detector parameter file
- **ENHANCEMENT:** DPR CATG and DPR TECH added to metadata of sof imagefilecollection objects
- **ENHANCEMENT:** wcs copied from a single frame into the combined frames during clip and stack
- **REFACTOR:** bad-pixel map paths abstracted to detector settings files
- **REFACTOR:** renaming of unit-testing test data directories
- **REFACTOR:** only filenames reported by sof summaries when files are found in the same directory (easier to read on terminal)
- **FIXED:** fixed detector science pixels for UVB

2.9.20 v0.3.1 - August 25, 2020

- **FEATURE:** recipe & command-line tool for master dark creation (`mdark`)
- **ENHANCEMENT:** default binning add to detector settings file
- **ENHANCEMENT:** added mixed exposure time unit test for dark-frames
- **ENHANCEMENT:** added default values for gain and ron in the detector settings files. Default values can be overwritten if correct GAIN and RON are found in fits-headers (overwritten for UVB and VIS but not NIR for XShooter)
- **ENHANCEMENT:** can now interrupt "~" as home directory in sof file path
- **FIXED:** binning factor used when trimming frames
- **FIXED:** the trimming dimensions of NIR frames - bad-pixel map now aligns correctly with data frame
- **FIXED:** science pixels for all 3 xshooter detectors in parameters file

2.9.21 v0.3.0 - August 18, 2020

- **FEATURE:** added a `write` method to the `_base_recipe` to write frames to disk (renames extensions to ESO preferred naming scheme)
- **FEATURE:** detector lookup class added alongside yaml files to host detector specific parameters (rotation, science-pixels etc). Code has been updated to remove hard-wired detector values.
- **FEATURE:** added a cleanup method to remove intermediate file once recipe completes
- **ENHANCEMENT:** parameters for clip and stack method added to the settings files
- **ENHANCEMENT:** added strict typing of data and variables with astropy units to avoid silent mistakes in frame arithmetic
- **ENHANCEMENT:** added mixing of readout speeds to input frame verification checks
- **ENHANCEMENT:** added readnoise and gain to the list of keyword values to check during frame verification
- **ENHANCEMENT:** inject a 'SOXSPIPE PRE' keyword with timestamp value into prepared frames
- **ENHANCEMENT:** check frames for 'SOXSPIPE PRE' keyword before preparing - raises exception if found
- **ENHANCEMENT:** ron and gain are added to the recipe's detector lookup dictionary during frame verification (so they don't need read again later)
- **REFACTOR:** moved stacking code to it own `clip_and_stack` method hosted in the `_base_recipe`
- **REFACTOR:** moved basic input frame verifications to the `_base_recipe` - so not to repeat code
- **REFACTOR:** removed python 2.7 support - not feasible with CCDProc
- **DOCS:** added workflow diagrams to the documentation for many of the methods implemented (`prepare_frames()`, `clip_and_stack`)

2.9.22 v0.2.0 - February 27, 2020

- **FEATURE** added keyword lookups - abstracting exact keyword names from code

2.10 Modules

`soxspipe.commonutils`	*common tools used throughout package*
`soxspipe.recipes`	*The pipeline recipes*
`soxspipe.commonutils.polynomials`	*definition of polynomial functions needed throughout code*
`soxspipe.commonutils.toolkit`	*small reusable functions used throughout soxspipe*
`soxspipe.utKit`	*Unit testing tools*

2.10.1 commonutils *(module)*

common tools used throughout package

Classes

`create_dispersion_map`(log, settings, . . . [, . . .])	*detect arc-lines on a pinhole frame to generate a dispersion solution*
`data_organiser`(log, rootDir)	*The worker class for the data_organiser module*
`detect_continuum`(log, pinholeFlat, . . . [, . . .])	*find and fit the continuum in a pinhole flat frame with low-order polynomials.
`detect_order_edges`(log, flatFrame, . . . [, . . .])	*using a fully-illuminated slit flat frame detect and record the order-edges*
`detector_lookup`(log[, settings])	*return a dictionary of detector characteristics and parameters*
`flux_calibration`(log, responseFunction, . . .)	*The worker class for the flux_calibration module*
`horne_extraction`(log, settings, . . . [, . . .])	*perform optimal source extraction using the Horne method (Horne 1986)*
`keyword_lookup`(log[, instrument, settings])	*The worker class for the keyword_lookup module*
`reducer`(log, workspaceDirectory[, settings, . . .])	*reduce all the data in a workspace, or target specific obs and files for reduction*
`response_function`(log, stdExtractionPath[, . . .])	*The worker class for the response_function module*
`subtract_background`(log, frame, orderTable)	*fit and subtract background flux from scattered light from frame*
`subtract_sky`(log, settings, recipeSettings, . . .)	*Subtract the sky background from a science image using the Kelson Method*

Functions

dispersion_map_to_pixel_arrays(log, . . . [, . . .])	*use a first-guess dispersion map to append x,y fits to line-list data frame.*
filenamer(log, frame[, keywordLookup, . . .])	Given a FITS object, use the SOXS file-naming scheme to return a filename to be used to save the FITS object to disk
getpackagepath()	*Get the root path for this python package*
uncompress(log, directory)	uncompress ESO fits.Z frames

2.10.2 recipes *(module)*

The pipeline recipes

Classes

soxs_disp_solution(log[, settings, . . .])	*generate a first approximation of the dispersion solution from single pinhole frames*
soxs_mbias(log[, settings, inputFrames, . . .])	*The* soxs_mbias *recipe is used to generate a master-bias frame from a set of input raw bias frames.
soxs_mdark(log[, settings, inputFrames, . . .])	*The soxs_mdark recipe*
soxs_mflat(log[, settings, inputFrames, . . .])	*The soxs_mflat recipe*
soxs_order_centres(log[, settings, . . .])	*The soxs_order_centres recipe*
soxs_spatial_solution(log[, settings, . . .])	*The soxs_spatial_solution recipe*
soxs_stare(log[, settings, inputFrames, . . .])	*The soxs_stare recipe*
soxs_straighten(log[, settings, . . .])	*The soxs_straighten recipe*

2.10.3 polynomials *(module)*

definition of polynomial functions needed throughout code

Author David Young

Date Created September 10, 2020

Classes

chebyshev_order_wavelength_polynomials(log, . . .)	*the chebyshev polynomial fits for the single frames; to be iteratively fitted to minimise errors*
chebyshev_order_xy_polynomials(log, . . . [, . . .])	*the chebyshev polynomial fits FIX ME*
chebyshev_xy_polynomial(log, y_deg[, yCol, . . .])	*the chebyshev polynomial fits for the pinhole flat frame order tracing; to be iteratively fitted to minimise errors*
object()	The base class of the class hierarchy.

continues on next page

Table 5 – continued from previous page

`tools`(arguments, docString[, logLevel, . . .])	*common setup methods & attributes of the main function in cl-util*

2.10.4 toolkit *(module)*

small reusable functions used throughout soxspipe

Author David Young

Date Created September 18, 2020

Classes

`MaxFilter`(max_level)	
`chebyshev_order_xy_polynomials`(log, . . . [, . . .])	*the chebyshev polynomial fits FIX ME*
`chebyshev_xy_polynomial`(log, y_deg[, yCol, . . .])	*the chebyshev polynomial fits for the pinhole flat frame order tracing; to be iteratively fitted to minimise errors*
`datetime`(year, month, day[, hour[, minute[, . . .)	The year, month and day arguments are required.
`detector_lookup`(log[, settings])	*return a dictionary of detector characteristics and parameters*
`keyword_lookup`(log[, instrument, settings])	*The worker class for the keyword_lookup module*
`object`()	The base class of the class hierarchy.
`tools`(arguments, docString[, logLevel, . . .])	*common setup methods & attributes of the main function in cl-util*

Functions

`add_recipe_logger`(log, productPath)	*add a recipe-specific handler to the default logger that writes the recipe's logs adjacent to the recipe project*
`create_dispersion_solution_grid_lines_for_plot`(. . .)	*give a dispersion solution and accompanying 2D dispersion map image, generate the grid lines to add to QC plots*
`cut_image_slice`(log, frame, width, length, x, y)	*cut and return an N-pixel wide and M-pixels long slice, centred on a given coordinate from an image frame*
`dispersion_map_to_pixel_arrays`(log, . . . [, . . .])	*use a first-guess dispersion map to append x,y fits to line-list data frame.*
`expanduser`(path)	Expand :sub:` and `user constructions. If user or $HOME is unknown, do nothing.
`generic_quality_checks`(log, frame, settings, . . .)	*measure very basic quality checks on a frame and return the QC table with results appended*
`get_calibration_lamp`(log, frame, kw)	*given a frame, determine which calibration lamp is being used*
`get_calibrations_path`(log, settings)	*return the root path to the static calibrations*
`predict_product_path`(sofName[, recipeName])	*predict the path of the recipe product from a given SOF name*

continues on next page

Table 7 – continued from previous page

`quicklook_image`(log, CCDObject[, show, ext, ...])	*generate a quicklook image of a CCDObject - useful for development/debugging*
`read_spectral_format`(log, settings, arm[, ...])	*read the spectral format table to get some key parameters*
`spectroscopic_image_quality_checks`(log, ...)	*measure and record spectroscopic image quailty checks*
`twoD_disp_map_image_to_dataframe`(log, ...[, ...])	*convert the 2D dispersion image map to a pandas dataframe*
`unpack_order_table`(log, orderTablePath[, ...])	*unpack an order table and return a top-level `orderPolyTable` data-frame and a second `orderPixelTable` data-frame with the central-trace coordinates of each order given

2.10.5 utKit *(module)*

Unit testing tools

Classes

`utKit`(moduleDirectory[, dbConn])	*Override dryx utKit*

2.11 Classes

`soxspipe.commonutils.create_dispersion_map`	*detect arc-lines on a pinhole frame to generate a dispersion solution*
`soxspipe.commonutils.data_organiser`	*The worker class for the data_organiser module*
`soxspipe.commonutils.detect_continuum`	*find and fit the continuum in a pinhole flat frame with low-order polynomials.
`soxspipe.commonutils.detect_order_edges`	*using a fully-illuminated slit flat frame detect and record the order-edges*
`soxspipe.commonutils.detector_lookup`	*return a dictionary of detector characteristics and parameters*
`soxspipe.commonutils.flux_calibration`	*The worker class for the flux_calibration module*
`soxspipe.commonutils.horne_extraction`	*perform optimal source extraction using the Horne method (Horne 1986)*
`soxspipe.commonutils.keyword_lookup`	*The worker class for the keyword_lookup module*
`soxspipe.commonutils.polynomials.chebyshev_order_wavelength_polynomials`	*the chebyshev polynomial fits for the single frames; to be iteratively fitted to minimise errors*
`soxspipe.commonutils.polynomials.chebyshev_order_xy_polynomials`	*the chebyshev polynomial fits FIX ME*
`soxspipe.commonutils.polynomials.chebyshev_xy_polynomial`	*the chebyshev polynomial fits for the pinhole flat frame order tracing; to be iteratively fitted to minimise errors*
`soxspipe.commonutils.reducer`	*reduce all the data in a workspace, or target specific obs and files for reduction*
`soxspipe.commonutils.response_function`	*The worker class for the response_function module*

continues on next page

Table 9 – continued from previous page

soxspipe.commonutils.subtract_background	*fit and subtract background flux from scattered light from frame*
soxspipe.commonutils.subtract_sky	*Subtract the sky background from a science image using the Kelson Method*
soxspipe.commonutils.toolkit.MaxFilter	
soxspipe.recipes.soxs_disp_solution	*generate a first approximation of the dispersion solution from single pinhole frames*
soxspipe.recipes.soxs_mbias	*The* soxs_mbias *recipe is used to generate a master-bias frame from a set of input raw bias frames.
soxspipe.recipes.soxs_mdark	*The soxs_mdark recipe*
soxspipe.recipes.soxs_mflat	*The soxs_mflat recipe*
soxspipe.recipes.soxs_order_centres	*The soxs_order_centres recipe*
soxspipe.recipes.soxs_spatial_solution	*The soxs_spatial_solution recipe*
soxspipe.recipes.soxs_stare	*The soxs_stare recipe*
soxspipe.recipes.soxs_straighten	*The soxs_straighten recipe*

2.11.1 create_dispersion_map *(class)*

class create_dispersion_map(*log*, *settings*, *recipeSettings*, *pinholeFrame*, *firstGuessMap=False*, *orderTable=False*, *qcTable=False*, *productsTable=False*, *sofName=False*, *create2DMap=True*)

Bases: object

detect arc-lines on a pinhole frame to generate a dispersion solution

Key Arguments:

- log – logger
- settings – the settings dictionary
- recipeSettings – the recipe specific settings
- pinholeFrame – the calibrated pinhole frame (single or multi)
- firstGuessMap – the first guess dispersion map from the soxs_disp_solution recipe (needed in soxs_spat_solution recipe). Default *False*.
- orderTable – the order geometry table
- qcTable – the data frame to collect measured QC metrics
- productsTable – the data frame to collect output products
- sofName – name of the originating SOF file
- create2DMap – create the 2D image map of wavelength, slit-position and order from disp solution.

Usage:

```
from soxspipe.commonutils import create_dispersion_map
mapPath, mapImagePath, res_plots, qcTable, productsTable = create_dispersion_map(
    log=log,
    settings=settings,
    pinholeFrame=frame,
    firstGuessMap=False,
```

(continues on next page)

(continued from previous page)

```
    qcTable=self.qc,
    productsTable=self.products
).get()
```

Methods

`calculate_residuals`(orderPixelTable, xcoeff, ...)	*calculate residuals of the polynomial fits against the observed line positions*
`convert_and_fit`(order, bigWlArray, ...[, plots])	*convert wavelength and slit position grids to pixels*
`create_new_static_line_list`(dispersionMapPath)	*using a first pass dispersion solution, use a line atlas to generate a more accurate and more complete static line list*
`create_placeholder_images`([order, plot, reverse])	*create CCDData objects as placeholders to host the 2D images of the wavelength and spatial solutions from dispersion solution map*
`detect_pinhole_arc_line`(predictedLine[, iraf])	*detect the observed position of an arc-line given the predicted pixel positions*
`fit_polynomials`(orderPixelTable, ...[, ...])	*iteratively fit the dispersion map polynomials to the data, clipping residuals with each iteration*
`get`()	*generate the dispersion map*
`get_predicted_line_list`()	*lift the predicted line list from the static calibrations*
`map_to_image`(dispersionMapPath)	*convert the dispersion map to images in the detector format showing pixel wavelength values and slit positions*
`order_to_image`(orderInfo)	*convert a single order in the dispersion map to wavelength and slit position images*
`write_map_to_file`(xcoeff, ycoeff, orderDeg, ...)	*write out the fitted polynomial solution coefficients to file*

calculate_residuals(*orderPixelTable*, *xcoeff*, *ycoeff*, *orderDeg*, *wavelengthDeg*, *slitDeg*, *writeQCs=False*, *pixelRange=False*)

calculate residuals of the polynomial fits against the observed line positions

Key Arguments:

- `orderPixelTable` – the predicted line list as a data frame
- `xcoeff` – the x-coefficients
- `ycoeff` – the y-coefficients
- `orderDeg` – degree of the order fitting
- `wavelengthDeg` – degree of wavelength fitting
- `slitDeg` – degree of the slit fitting (False for single pinhole)
- `writeQCs` – write the QCs to dataframe? Default *False*
- `pixelRange` – return centre pixel *and* +- 2nm from the centre pixel (to measure the pixel scale)

Return:

- `residuals` – combined x-y residuals

- `mean` – the mean of the combine residuals
- `std` – the stdev of the combine residuals
- `median` – the median of the combine residuals

convert_and_fit(*order*, *bigWlArray*, *bigSlitArray*, *slitMap*, *wlMap*, *iteration*, *plots=False*)

convert wavelength and slit position grids to pixels

Key Arguments:

- `order` – the order being considered
- `bigWlArray` – 1D array of all wavelengths to be converted
- `bigSlitArray` – 1D array of all split-positions to be converted (same length as `bigWlArray`)
- `slitMap` – place-holder image hosting fitted pixel slit-position values
- `wlMap` – place-holder image hosting fitted pixel wavelength values
- `iteration` – the iteration index (used for CL reporting)
- `plots` – show plot of the slit-map

Return:

- `orderPixelTable` – dataframe containing unfitted pixel info
- `remainingCount` – number of remaining pixels in orderTable

Usage:

```
orderPixelTable = self.convert_and_fit(
        order=order, bigWlArray=bigWlArray, bigSlitArray=bigSlitArray,␣
↪slitMap=slitMap, wlMap=wlMap)
```

create_new_static_line_list(*dispersionMapPath*)

using a first pass dispersion solution, use a line atlas to generate a more accurate and more complete static line list

Key Arguments:

```
- `dispersionMapPath` -- path to the first pass dispersion solution
```

Return:

```
- `newPredictedLineList` -- a new predicted line list (to replace the static␣
↪calibration line-list)
```

create_placeholder_images(*order=False*, *plot=False*, *reverse=False*)

create CCDData objects as placeholders to host the 2D images of the wavelength and spatial solutions from dispersion solution map

Key Arguments:

- `order` – specific order to generate the placeholder pixels for. Inner-order pixels set to NaN, else set to 0. Default *False* (generate all orders)
- `plot` – generate plots of placeholder images (for debugging). Default *False*.
- `reverse` – Inner-order pixels set to 0, else set to NaN (reverse of default output).

Return:

- `slitMap` – placeholder image to add pixel slit positions to

- `wlMap` – placeholder image to add pixel wavelength values to

Usage:

```
slitMap, wlMap, orderMap = self._create_placeholder_images(order=order)
```

detect_pinhole_arc_line(*predictedLine*, *iraf=True*)

detect the observed position of an arc-line given the predicted pixel positions

Key Arguments:

- `predictedLine` – single predicted line coordinates from predicted line-list
- `iraf` – use IRAF star finder to generate a FWHM

Return:

- `predictedLine` – the line with the observed pixel coordinates appended (if detected, otherwise nan)

fit_polynomials(*orderPixelTable*, *wavelengthDeg*, *orderDeg*, *slitDeg*, *missingLines=False*)

iteratively fit the dispersion map polynomials to the data, clipping residuals with each iteration

Key Arguments:

- `orderPixelTable` – data frame containing order, wavelengths, slit positions and observed pixel positions
- `wavelengthDeg` – degree of wavelength fitting
- `orderDeg` – degree of the order fitting
- `slitDeg` – degree of the slit fitting (0 for single pinhole)
- `missingLines` – lines not detected on the image

Return:

- `xcoeff` – the x-coefficients post clipping
- `ycoeff` – the y-coefficients post clipping
- `goodLinesTable` – the fitted line-list with metrics
- `clippedLinesTable` – the lines that were sigma-clipped during polynomial fitting

get()

generate the dispersion map

Return:

- `mapPath` – path to the file containing the coefficients of the x,y polynomials of the global dispersion map fit

get_predicted_line_list()

lift the predicted line list from the static calibrations

Return:

- `orderPixelTable` – a panda's data-frame containing wavelength,order,slit_index,slit_position,detector_x,detector_y

map_to_image(*dispersionMapPath*)

convert the dispersion map to images in the detector format showing pixel wavelength values and slit positions

Key Arguments:

- `dispersionMapPath` – path to the full dispersion map to convert to images

Return:

- `dispersion_image_filePath` – path to the FITS image with an extension for wavelength values and another for slit positions

Usage:

```
mapImagePath = self.map_to_image(dispersionMapPath=mapPath)
```

order_to_image (*orderInfo*)
convert a single order in the dispersion map to wavelength and slit position images

Key Arguments:

- `orderInfo` – tuple containing the order number to generate the images for, the minimum wavelength to consider (from format table) and maximum wavelength to consider (from format table).

Return:

- `slitMap` – the slit map with order values filled
- `wlMap` – the wavelengths map with order values filled

Usage:

```
slitMap, wlMap = self.order_to_image(order=order,minWl=minWl, maxWl=maxWl)
```

write_map_to_file (*xcoeff*, *ycoeff*, *orderDeg*, *wavelengthDeg*, *slitDeg*)
write out the fitted polynomial solution coefficients to file

Key Arguments:

- `xcoeff` – the x-coefficients
- `ycoeff` – the y-coefficients
- `orderDeg` – degree of the order fitting
- `wavelengthDeg` – degree of wavelength fitting
- `slitDeg` – degree of the slit fitting (False for single pinhole)

Return:

- `disp_map_path` – path to the saved file

2.11.2 data_organiser *(class)*

class data_organiser (*log*, *rootDir*)
Bases: `object`

The worker class for the data_organiser module

Key Arguments:

- `log` – logger
- `rootDir` – the root directory of the data to process

Usage:

To setup your logger, settings and database connections, please use the `fundamentals` package (see tutorial here).

To initiate a data_organiser object, use the following:

Todo:

- create cl-util for this class
- add a tutorial about `data_organiser` to documentation
- create a blog post about what `data_organiser` does

```
from soxspipe.commonutils import data_organiser
do = data_organiser(
    log=log,
    rootDir="/path/to/workspace/root/"
)
do.prepare()
```

Methods

`categorise_frames`(filteredFrames[, verbose])	*given a dataframe of frame, categorise frames into raw, reduced pixels, reduced tables*
`create_directory_table`(pathToDirectory, ...)	*create an astropy table based on the contents of a directory*
`generate_sof_and_product_names`(series, ...)	*add a recipe name and SOF filename to all rows in the raw_frame_sets DB table*
`populate_products_table`(series, reductionOrder)	*determine what the products should be for a given recipe and SOF file and ppulate the products table*
`prepare`()	*Prepare the workspace for data reduction by generating all SOF files and reduction scripts.*
`session_create`([sessionId])	*create a data-reduction session with accompanying settings file and required directories*
`session_list`([silent])	*list the sessions available to the user*
`session_refresh`()	*refresh a session's SOF file (needed if a recipe fails)*
`session_switch`(sessionId)	*switch to an existing workspace data-reduction session*
`symlink_session_assets_to_workspace_root`()	*symlink session QC, product, SOF directories, database and scripts to workspace root*
`sync_sql_table_to_directory`(directory, tableName)	*sync sql table content to files in a directory (add and delete from table as appropriate)*

categorise_frames (*filteredFrames*, *verbose=False*)

given a dataframe of frame, categorise frames into raw, reduced pixels, reduced tables

Key Arguments:

- `filteredFrames` – the dataframe from which to split frames into categorise.
- `verbose` – print results to stdout.

Return:

- `rawFrames` – dataframe of raw frames only
- `reducedFramesPixels` – dataframe of reduced images only

- `reducedFramesTables` – dataframe of reduced tables only

Usage:

```
rawFrames, reducedFramesPixels, reducedFramesTables = self.categorise_
↪frames(filteredFrames)
```

create_directory_table(*pathToDirectory*, *filterKeys*)

create an astropy table based on the contents of a directory

Key Arguments:

- `log` – logger
- `pathToDirectory` – path to the directory containing the FITS frames
- `filterKeys` – these are the keywords we want to filter on later

Return

- `masterTable` – the primary dataframe table listing all FITS files in the directory (including indexes on `filterKeys` columns)
- `fitsPaths` – a simple list of all FITS file paths
- `fitsNames` – a simple list of all FITS file name

Usage:

```
# GENERATE AN ASTROPY TABLES OF FITS FRAMES WITH ALL INDEXES NEEDED
masterTable, fitsPaths, fitsNames = create_directory_table(
    log=log,
    pathToDirectory="/my/directory/path",
    keys=["file","mjd-obs", "exptime","cdelt1", "cdelt2"],
    filterKeys=["mjd-obs","exptime"]
)
```

generate_sof_and_product_names(*series*, *reductionOrder*, *rawFrames*, *calibrationFrames*, *calibrationTables*)

add a recipe name and SOF filename to all rows in the raw_frame_sets DB table

Key Arguments:

- `series` – the dataframe row/series to apply work on

Usage:

Todo:

- add usage info

```
usage code
```

populate_products_table(*series*, *reductionOrder*)

determine what the products should be for a given recipe and SOF file and ppulate the products table

Key Arguments:

- `recipeName` – the name of the recipe.
- `sofName` – the name of the sof file.

Return:

```
- None
```

Usage:

```
usage code
```

Todo:

- add usage info
- create a sublime snippet for usage
- write a command-line tool for this method
- update package tutorial with command-line tool info if needed

prepare()

Prepare the workspace for data reduction by generating all SOF files and reduction scripts.

session_create(*sessionId=False*)

create a data-reduction session with accompanying settings file and required directories

Key Arguments:

- `sessionId` – optionally provide a sessionId (A-Z, a-z 0-9 and/or _- allowed, 16 character limit)

Return:

- `sessionId` – the unique ID of the data-reduction session

Usage:

```
do = data_organiser(
    log=log,
    rootDir="/path/to/workspace/root/"
)
sessionId = do.session_create(sessionId="my_supernova")
```

session_list(*silent=False*)

list the sessions available to the user

Key Arguments:

- `silent` – don't print listings if True

Return:

- `currentSession` – the single ID of the currently used session
- `allSessions` – the IDs of the other sessions

Usage:

```
from soxspipe.commonutils import data_organiser
do = data_organiser(
    log=log,
    rootDir="."
)
currentSession, allSessions = do.session_list()
```

session_refresh()

refresh a session's SOF file (needed if a recipe fails)

Key Arguments:

```
# -
```

Return:

```
- None
```

Usage:

```
usage code
```

Todo:

- add usage info
- create a sublime snippet for usage
- write a command-line tool for this method
- update package tutorial with command-line tool info if needed

session_switch(*sessionId*)

switch to an existing workspace data-reduction session

Key Arguments:

- `sessionId` – the sessionId to switch to

Usage:

```
from soxspipe.commonutils import data_organiser
do = data_organiser(
    log=log,
    rootDir="."
)
do.session_switch(mySessionId)
```

symlink_session_assets_to_workspace_root()

symlink session QC, product, SOF directories, database and scripts to workspace root

Key Arguments:

```
# -
```

Return:

```
- None
```

Usage:

```
usage code
```

Todo:

- add usage info
- create a sublime snippet for usage
- write a command-line tool for this method
- update package tutorial with command-line tool info if needed

sync_sql_table_to_directory(*directory*, *tableName*, *recursive=False*)

sync sql table content to files in a directory (add and delete from table as appropriate)

Key Arguments:

- `directory` – the directory of fits file to inspect.
- `tableName` – the sqlite table to sync.
- `recursive` – recursively dig into the directory to find FITS files? Default *False*.

Return:

```
- None
```

Usage:

```
do.sync_sql_table_to_directory('/raw/directory/', 'raw_frames',␣
↪recursive=False)
```

2.11.3 detect_continuum *(class)*

class detect_continuum(*log*, *pinholeFlat*, *dispersion_map*, *settings=False*, *recipeSettings=False*, *recipeName=False*, *qcTable=False*, *productsTable=False*, *sofName=False*, *binx=1*, *biny=1*, *lampTag=False*)

Bases: `soxspipe.commonutils.detect_continuum._base_detect`

find and fit the continuum in a pinhole flat frame with low-order polynomials. These polynominals are the central loctions of the orders

Key Arguments:

- `log` – logger
- `pinholeFlat` – calibrationed pinhole flat frame (CCDObject)
- `dispersion_map` – path to dispersion map csv file containing polynomial fits of the dispersion solution for the frame
- `settings` – the settings dictionary
- `recipeSettings` – the recipe specific settings
- `recipeName` – the recipe name as given in the settings dictionary
- `qcTable` – the data frame to collect measured QC metrics
- `productsTable` – the data frame to collect output products
- `sofName` —- name of the originating SOF file
- `binx` – binning in x-axis
- `biny` – binning in y-axis

- `lampTag` – add this tag to the end of the product filename (Default *False*)

Usage:

To use the `detect_continuum` object, use the following:

```
from soxspipe.commonutils import detect_continuum
detector = detect_continuum(
    log=log,
    pinholeFlat=pinholeFlat,
    dispersion_map=dispersion_map,
    settings=settings,
    recipeName="soxs-order-centre"
)
order_table_path = detector.get()
```

Methods

`calculate_residuals`(orderPixelTable, coeff, ...)	*calculate residuals of the polynomial fits against the observed line postions*
`create_pixel_arrays`()	*create a pixel array for the approximate centre of each order*
`fit_1d_gaussian_to_slice`(pixelPostion)	*cut a slice from the pinhole flat along the cross-dispersion direction centred on pixel position, fit 1D gaussian and return the peak pixel position*
`fit_global_polynomial`(pixelList[, axisACol, ...])	*iteratively fit the global polynomial to the data, fitting axisA as a function of axisB, clipping residuals with each iteration*
`fit_order_polynomial`(pixelList, order, ...)	*iteratively fit the dispersion map polynomials to the data, clipping residuals with each iteration*
`get`()	*return the order centre table filepath*
`plot_results`(orderPixelTable, ...)	*generate a plot of the polynomial fits and residuals*
`write_order_table_to_file`(frame, ...)	*write out the fitted polynomial solution coefficients to file*

calculate_residuals(*orderPixelTable*, *coeff*, *axisACol*, *axisBCol*, *orderCol=False*, *writeQCs=False*)

calculate residuals of the polynomial fits against the observed line postions

Key Arguments:

- `orderPixelTable` – data-frame containing pixel list for given order
- `coeff` – the coefficients of the fitted polynomial
- `axisACol` – name of x-pixel column
- `axisBCol` – name of y-pixel column
- `orderCol` – name of the order column (global fits only)
- `writeQCs` – write the QCs to dataframe? Default *False*

Return:

- `res` – x residuals
- `mean` – the mean of the residuals

- `std` – the stdev of the residuals
- `median` – the median of the residuals
- `xfit` – fitted x values

`create_pixel_arrays`()

create a pixel array for the approximate centre of each order

Return:

- `orderPixelTable` – a data-frame containing lines and associated pixel locations

`fit_1d_gaussian_to_slice`(*pixelPostion*)

cut a slice from the pinhole flat along the cross-dispersion direction centred on pixel position, fit 1D gaussian and return the peak pixel position

Key Arguments:

- `pixelPostion` – the x,y pixel coordinate from orderPixelTable data-frame (series)

Return:

- `pixelPostion` – now including gaussian fit peak xy position

`fit_global_polynomial`(*pixelList*, *axisACol='cont_x'*, *axisBCol='cont_y'*, *orderCol='order'*, *exponentsIncluded=False*, *writeQCs=False*)

iteratively fit the global polynomial to the data, fitting axisA as a function of axisB, clipping residuals with each iteration

Key Arguments:

- `pixelList` – data-frame group containing x,y pixel array
- `exponentsIncluded` – the exponents have already been calculated in the dataframe so no need to regenerate. Default *False*

Return:

- `coeffs` – the coefficients of the polynomial fit
- `pixelList` – the pixel list but now with fits and residuals included
- `allClipped` – data that was sigma-clipped

`fit_order_polynomial`(*pixelList*, *order*, *axisBDeg*, *axisACol*, *axisBCol*, *exponentsIncluded=False*)

iteratively fit the dispersion map polynomials to the data, clipping residuals with each iteration

Key Arguments:

- `pixelList` – data-frame group containing x,y pixel array
- `order` – the order to fit
- `axisBDeg` – degree for polynomial to fit
- `axisACol` – name of columns containing axis to be fitted
- `axisBCol` – name of columns containing free axis (values known)
- `exponentsIncluded` – the exponents have already been calculated in the dataframe so no need to regenerate. Default *False*

Return:

- `coeffs` – the coefficients of the polynomial fit
- `pixelList` – the pixel list but now with fits and residuals included

get()

return the order centre table filepath

Return:

- `order_table_path` – file path to the order centre table giving polynomial coeffs to each order fit

plot_results(*orderPixelTable*, *orderPolyTable*, *clippedData*)

generate a plot of the polynomial fits and residuals

Key Arguments:

- `orderPixelTable` – the pixel table with residuals of fits
- `orderPolyTable` – data-frame of order-location polynomial coeff
- `clippedData` – the sigma-clipped data

Return:

- `filePath` – path to the plot pdf
- `orderMetaTable` – dataframe of useful order fit metadata

write_order_table_to_file(*frame*, *orderPolyTable*, *orderMetaTable*)

write out the fitted polynomial solution coefficients to file

Key Arguments:

- `frame` – the calibration frame used to generate order location data
- `orderPolyTable` – data-frames containing centre location coefficients (and possibly also order edge coeffs)
- `orderMetaTable` – extra order meta data to be added in an extra FITS extension

Return:

- `order_table_path` – path to the order table file

2.11.4 detect_order_edges *(class)*

class detect_order_edges(*log*, *flatFrame*, *orderCentreTable*, *settings=False*, *recipeSettings=False*, *recipeName='soxs-mflat'*, *verbose=False*, *qcTable=False*, *productsTable=False*, *tag=''*, *sofName=False*, *binx=1*, *biny=1*, *extendToEdges=True*, *lampTag=False*)

Bases: `soxspipe.commonutils.detect_continuum._base_detect`

using a fully-illuminated slit flat frame detect and record the order-edges

Key Arguments:

- `log` – logger
- `settings` – the settings dictionary
- `recipeSettings` – the recipe specific settings
- `flatFrame` – the flat frame to detect the order edges on
- `orderCentreTable` – the order centre table
- `recipeName` – name of the recipe as it appears in the settings dictionary
- `verbose` – verbose. True or False. Default *False*

- `qcTable` – the data frame to collect measured QC metrics
- `productsTable` – the data frame to collect output products
- `tag` – e.g. '_DLAMP' to differentiate between UV-VIS lamps
- `sofName` – name of the originating SOF file
- `binx` – binning in x-axis
- `biny` – binning in y-axis
- `extendToEdges` – if true, extend the order edge tracing to the edges of the frame (Default *True*)
- `lampTag` – add this tag to the end of the product filename (Default *False*)

Usage:

Todo:

- add a tutorial about `detect_order_edges` to documentation

```
from soxspipe.commonutils import detect_order_edges
edges = detect_order_edges(
    log=log,
    flatFrame=flatFrame,
    orderCentreTable=orderCentreTable,
    settings=settings,
    recipeSettings=recipeSettings,
    recipeName="soxs-mflat",
    verbose=False,
    qcTable=False,
    productsTable=False,
    extendToEdges=True,
    lampTag=False
)
productsTable, qcTable, orderDetectionCounts = edges.get()
```

Methods

`calculate_residuals`(orderPixelTable, coeff, . . .)	*calculate residuals of the polynomial fits against the observed line postions*
`determine_lower_upper_edge_pixel_positions`(. . .)	*from a pixel postion somewhere on the trace of the order centre, return the lower and upper edges of the order*
`determine_order_flux_threshold`(orderData, . . .)	*determine the flux threshold at the central column of each order*
`fit_global_polynomial`(pixelList[, axisACol, . . .])	*iteratively fit the global polynomial to the data, fitting axisA as a function of axisB, clipping residuals with each iteration*
`fit_order_polynomial`(pixelList, order, . . .)	*iteratively fit the dispersion map polynomials to the data, clipping residuals with each iteration*
`get`()	*get the detect_order_edges object*
`plot_results`(orderPixelTableUpper, . . .)	*generate a plot of the polynomial fits and residuals*

continues on next page

Table 13 – continued from previous page

`write_order_table_to_file`(frame, ...)	*write out the fitted polynomial solution coefficients to file*

calculate_residuals(*orderPixelTable*, *coeff*, *axisACol*, *axisBCol*, *orderCol=False*, *writeQCs=False*)

calculate residuals of the polynomial fits against the observed line postions

Key Arguments:

- `orderPixelTable` – data-frame containing pixel list for given order
- `coeff` – the coefficients of the fitted polynomial
- `axisACol` – name of x-pixel column
- `axisBCol` – name of y-pixel column
- `orderCol` – name of the order column (global fits only)
- `writeQCs` – write the QCs to dataframe? Default *False*

Return:

- `res` – x residuals
- `mean` – the mean of the residuals
- `std` – the stdev of the residuals
- `median` – the median of the residuals
- `xfit` – fitted x values

determine_lower_upper_edge_pixel_positions(*orderData*)

from a pixel postion somewhere on the trace of the order centre, return the lower and upper edges of the order

Key Arguments:

- `orderData` – one row in the orderTable

Return:

- `orderData` – orderData with upper and lower edge xcoord arrays added

determine_order_flux_threshold(*orderData*, *orderPixelTable*)

determine the flux threshold at the central column of each order

Key Arguments:

- `orderData` – one row in the orderTable

```
- ``orderPixelTable`` the order table containing pixel arrays
```

Return:

- `orderData` – orderData with min and max flux thresholds added

fit_global_polynomial(*pixelList*, *axisACol='cont_x'*, *axisBCol='cont_y'*, *orderCol='order'*, *exponentsIncluded=False*, *writeQCs=False*)

iteratively fit the global polynomial to the data, fitting axisA as a function of axisB, clipping residuals with each iteration

Key Arguments:

- `pixelList` – data-frame group containing x,y pixel array
- `exponentsIncluded` – the exponents have already been calculated in the dataframe so no need to regenerate. Default *False*

Return:

- `coeffs` – the coefficients of the polynomial fit
- `pixelList` – the pixel list but now with fits and residuals included
- `allClipped` – data that was sigma-clipped

`fit_order_polynomial`(*pixelList*, *order*, *axisBDeg*, *axisACol*, *axisBCol*, *exponentsIncluded=False*)
iteratively fit the dispersion map polynomials to the data, clipping residuals with each iteration

Key Arguments:

- `pixelList` – data-frame group containing x,y pixel array
- `order` – the order to fit
- `axisBDeg` – degree for polynomial to fit
- `axisACol` – name of columns containing axis to be fitted
- `axisBCol` – name of columns containing free axis (values known)
- `exponentsIncluded` – the exponents have already been calculated in the dataframe so no need to regenerate. Default *False*

Return:

- `coeffs` – the coefficients of the polynomial fit
- `pixelList` – the pixel list but now with fits and residuals included

`get`()
get the detect_order_edges object

Return:

- `orderTablePath` – path to the new order table

`plot_results`(*orderPixelTableUpper*, *orderPixelTableLower*, *orderPolyTable*, *orderMetaTable*, *clippedDataUpper*, *clippedDataLower*)
generate a plot of the polynomial fits and residuals

Key Arguments:

- `orderPixelTableUpper` – the pixel table with residuals of fits for the upper edges
- `orderPixelTableLower` – the pixel table with residuals of fits for the lower edges
- `orderPolyTable` – data-frame of order-location polynomial coeff
- `orderMetaTable` – data-frame containing the limits of the fit
- `clippedDataUpper` – the sigma-clipped data from upper edge
- `clippedDataLower` – the sigma-clipped data from lower edge

Return:

- `filePath` – path to the plot pdf

`write_order_table_to_file`(*frame*, *orderPolyTable*, *orderMetaTable*)
write out the fitted polynomial solution coefficients to file

Key Arguments:

- `frame` – the calibration frame used to generate order location data
- `orderPolyTable` – data-frames containing centre location coefficients (and possibly also order edge coeffs)
- `orderMetaTable` – extra order meta data to be added in an extra FITS extension

Return:

- `order_table_path` – path to the order table file

2.11.5 detector_lookup *(class)*

class detector_lookup(*log*, *settings=False*)

Bases: `object`

return a dictionary of detector characteristics and parameters

Key Arguments:

- `log` – logger
- `settings` – the settings dictionary

Usage:

To initiate a detector_lookup object, use the following:

```
from soxspipe.commonutils import detector_lookup
detector = detector_lookup(
    log=log,
    settings=settings
).get("NIR")
print(detector["science-pixels"])
```

Methods

`get`(arm)	*return a dictionary of detector characteristics and parameters*

get(*arm*)

return a dictionary of detector characteristics and parameters

Key Arguments:

- `arm` – the detector parameters to return

2.11.6 flux_calibration *(class)*

class flux_calibration(*log*, *responseFunction*, *extractedSpectrum*, *settings=False*)

Bases: `object`

The worker class for the flux_calibration module

Key Arguments:

- `log` – logger
- `responseFunction` – the instrument response function.
- `extractedSpectrum` – the extracted science spectrum
- `settings` – the settings dictionary

Usage:

To setup your logger, settings and database connections, please use the `fundamentals` package (see tutorial here).

To initiate a flux_calibration object, use the following:

Todo:

- add usage info
- create a sublime snippet for usage
- create cl-util for this class
- add a tutorial about `flux_calibration` to documentation
- create a blog post about what `flux_calibration` does

```
usage code
```

Methods

`calibrate()`	*flux calibrate the science spectrum*

calibrate()

flux calibrate the science spectrum

Return:

```
- ``flux_calibration``
```

Usage:

Todo:

- add usage info
- create a sublime snippet for usage
- create cl-util for this method
- update the package tutorial if needed

```
usage code
```

2.11.7 horne_extraction *(class)*

class horne_extraction(*log*, *settings*, *recipeSettings*, *skyModelFrame*, *skySubtractedFrame*, *twoDMapPath*, *recipeName=False*, *qcTable=False*, *productsTable=False*, *dispersionMap=False*, *sofName=False*)

Bases: `object`

perform optimal source extraction using the Horne method (Horne 1986)

Key Arguments:

- `log` – logger
- `settings` – the settings dictionary
- `recipeSettings` – the recipe specific settings
- `skyModelFrame` – path to sky model frame
- `skySubtractedFrame` – path to sky subtracted frame
- `twoDMapPath` – path to 2D dispersion map image path
- `recipeName` – name of the recipe as it appears in the settings dictionary
- `qcTable` – the data frame to collect measured QC metrics
- `productsTable` – the data frame to collect output products
- `dispersionMap` – the FITS binary table containing dispersion map polynomial
- `sofName` – the set-of-files filename

Usage:

To setup your logger, settings and database connections, please use the `fundamentals` package (see tutorial here).

To initiate a horne_extraction object, use the following:

Todo:

- add usage info
- create a sublime snippet for usage
- create cl-util for this class
- add a tutorial about `horne_extraction` to documentation
- create a blog post about what `horne_extraction` does

```python
from soxspipe.commonutils import horne_extraction
optimalExtractor = horne_extraction(
    log=log,
    skyModelFrame=skyModelFrame,
    skySubtractedFrame=skySubtractedFrame,
    twoDMapPath=twoDMap,
```

(continues on next page)

(continued from previous page)

```
    settings=settings,
    recipeName="soxs-stare",
    qcTable=qc,
    productsTable=products,
    dispersionMap=dispMap,
    sofName=sofName
)
qc, products = optimalExtractor.extract()
```

Methods

`extract`()	*extract the full spectrum order-by-order and return FITS Binary table containing order-merged spectrum*
`merge_extracted_orders`(extractedOrdersDF)	*merge the extracted order spectra in one continuous spectrum*
`residual_merge`(group)	
`weighted_average`(group)	

extract()

extract the full spectrum order-by-order and return FITS Binary table containing order-merged spectrum

Return:

```
- None
```

merge_extracted_orders(*extractedOrdersDF*)

merge the extracted order spectra in one continuous spectrum

Key Arguments:

- `extractedOrdersDF` – a data-frame containing the extracted orders

Return:

```
- None
```

2.11.8 keyword_lookup *(class)*

class keyword_lookup(*log*, *instrument=False*, *settings=False*)

Bases: `object`

The worker class for the keyword_lookup module

Key Arguments:

- `log` – logger
- `settings` – the settings dictionary. Default *False*
- `instrument` – can directly add the instrument if settings file is not avalable. Default *False*

Usage

To initalise the keyword lookup object in your code add the following:

```
from soxspipe.commonutils import keyword_lookup
kw = keyword_lookup(
    log=log,
    settings=settings,
    instrument=False,
).get
```

After this it's possible to either look up a single keyword using it's alias:

```
kw("DET_NDITSKIP")
> "ESO DET NDITSKIP"
```

or return a list of keywords:

```
kw(["PROV", "DET_NDITSKIP"])
> ['PROV', 'ESO DET NDITSKIP']
```

For those keywords that require an index it's possible to also pass the index to the `kw` function:

```
kw("PROV", 9)
> 'PROV09'
```

If a tag is not in the list of FITS Header keyword aliases in the configuration file a `LookupError` will be raised.

Methods

`get`(tag[, index])	*given a tag, and optional keyword index, return the FITS Header keyword for the selected instrument*

get (*tag*, *index=False*)
given a tag, and optional keyword index, return the FITS Header keyword for the selected instrument

Key Arguments:

- `tag` – the keyword tag as set in the yaml keyword dictionary (e.g. 'SDP_KEYWORD_TMID' returns 'TMID'). Can be string or list of sttings.
- `index` – add an index to the keyword if not False (e.g. tag='PROV', index=3 returns 'PROV03') Default *False*

Return:

- `keywords` – the FITS Header keywords. Can be string or list of sttings depending on format of tag argument

Usage

See docstring for the class

2.11.9 chebyshev_order_wavelength_polynomials *(class)*

class chebyshev_order_wavelength_polynomials(*log*, *orderDeg*, *wavelengthDeg*, *slitDeg*, *exponentsIncluded=False*, *axis=False*)

Bases: `object`

the chebyshev polynomial fits for the single frames; to be iteratively fitted to minimise errors

Key Arguments:

- `log` – logger
- `orderDeg` – degree of the order polynomial components
- `wavelengthDeg` – degree of wavelength polynomial components
- `slitDeg` – degree of the slit polynomial components
- `exponentsIncluded` – the exponents have already been calculated in the dataframe so no need to regenerate. Default *False*
- `axis` – x, y or False. Default *False*.

Usage:

```
from soxspipe.commonutils.polynomials import chebyshev_order_wavelength_
↪polynomials
poly = chebyshev_order_wavelength_polynomials(
        log=self.log, orderDeg=orderDeg, wavelengthDeg=wavelengthDeg,␣
↪slitDeg=slitDeg).poly
```

Methods

`poly`(orderPixelTable, *coeff)	the polynomial definition

poly(*orderPixelTable*, **coeff*)

the polynomial definition

Key Arguments:

- `orderPixelTable` – a pandas dataframe containing wavelengths, orders and slit positions
- `*coeff` – a list of the initial coefficients

Return:

- `lhsVals` – the left-hand-side vals of the fitted polynomials

2.11.10 chebyshev_order_xy_polynomials *(class)*

class chebyshev_order_xy_polynomials(*log*, *orderDeg*, *axisBDeg*, *axisB='y'*, *axisBCol=False*, *orderCol=False*, *exponentsIncluded=False*)

Bases: `object`

the chebyshev polynomial fits FIX ME

Key Arguments:

- `log` – logger
- `orderDeg` – degree of the order polynomial components

- `axisBDeg` – degree for polynomial to fit free axis-values
- `axisB` – the free axis related to `axisBDeg`. Default *'y'*. ['x'|'y']
- `axisBCol` – name of the free axis column (if needed). Default *False*
- `orderCol` – name of the order column (if needed). Default *False*
- `exponentsIncluded` – the exponents have already been calculated in the dataframe so no need to regenerate. Default *False*

Usage:

```
from soxspipe.commonutils.polynomials import chebyshev_order_wavelength_
↪polynomials
poly = chebyshev_order_wavelength_polynomials(
        log=self.log, orderDeg=orderDeg, wavelengthDeg=wavelengthDeg,␣
↪slitDeg=slitDeg).poly
```

Methods

poly(orderPixelTable, *coeff)	the polynomial definition

poly (*orderPixelTable*, **coeff*)
the polynomial definition

Key Arguments:

- `orderPixelTable` – a pandas dataframe containing x, y, order
- `*coeff` – a list of the initial coefficients

Return:

- `lhsVals` – the left-hand-side vals of the fitted polynomials

2.11.11 chebyshev_xy_polynomial *(class)*

class chebyshev_xy_polynomial (*log*, *y_deg*, *yCol=False*, *exponentsIncluded=False*)
Bases: `object`

the chebyshev polynomial fits for the pinhole flat frame order tracing; to be iteratively fitted to minimise errors

Key Arguments:

- `log` – logger
- `yCol` – name of the yCol
- `y_deg` – y degree of the polynomial components
- `exponentsIncluded` – the exponents have already been calculated in the dataframe so no need to regenerate. Default *False*

Usage:

```
from soxspipe.commonutils.polynomials import chebyshev_xy_polynomial
poly = chebyshev_xy_polynomial(
        log=self.log, deg=deg).poly
```

Methods

poly(orderPixelTable, *coeff)	the polynomial definition

poly (*orderPixelTable*, **coeff*)

the polynomial definition

Key Arguments:

- `orderPixelTable` – data frame with all pixel data arrays
- `*coeff` – a list of the initial coefficients

Return:

- `xvals` – the x values of the fitted polynomial

2.11.12 reducer *(class)*

class reducer (*log*, *workspaceDirectory*, *settings=False*, *pathToSettings=False*, *quitOnFail=False*, *overwrite=False*)

Bases: `object`

reduce all the data in a workspace, or target specific obs and files for reduction

Key Arguments:

- `log` – logger
- `workspaceDirectory` – path to the root of the workspace
- `settings` – the settings dictionary
- `pathToSettings` – path to the settings file.
- `quitOnFail` – quit the pipeline on any recipe failure
- `overwrite` – overwrite existing reductions. Default *False*.

Usage:

```
from soxspipe.commonutils import reducer
collection = reducer(
    log=log,
    workspaceDirectory="/path/to/workspace/root/",
    settings=settings,
    pathToSettings="/path/to/settings.yaml"
)
collection.reduce()
```

Methods

`reduce()`	*reduce the selected data*
`run_recipe(recipe, sof)`	*execute a pipeline recipe*
`select_sof_files_to_process()`	*select all of the SOF files still requiring processing*

reduce()

reduce the selected data

Return:

```
- ``reducer``
```

Usage:

Todo:

- add usage info
- create a sublime snippet for usage
- create cl-util for this method
- update the package tutorial if needed

```
usage code
```

run_recipe(*recipe*, *sof*)

execute a pipeline recipe

Key Arguments:

- `recipe` – the name of the recipe tp execute
- `sof` – path to the sof file containing the files the recipe requires

Usage:

```
reducer.run_recipe("mbias", "/path/to/sofs/my_bias_files.sof")
```

select_sof_files_to_process()

select all of the SOF files still requiring processing

Key Arguments:

```
# -
```

Return:

```
- `rawGroups` -- a dataframe of the containing a list of recipes and sof file␣
↪paths
```

Usage:

```
rawGroups = reducer.select_sof_files_to_process()
```

2.11.13 response_function *(class)*

class response_function(*log*, *stdExtractionPath*, *settings=False*)

Bases: `object`

The worker class for the response_function module

Key Arguments:

- `log` – logger
- `settings` – the settings dictionary
- `stdExtractionPath` – fits binary table containing the extracted standard spectrum

Usage:

To setup your logger, settings and database connections, please use the `fundamentals` package (see tutorial here).

To initiate a response_function object, use the following:

Todo:

- add usage info
- create a sublime snippet for usage
- create cl-util for this class
- add a tutorial about `response_function` to documentation
- create a blog post about what `response_function` does

```
usage code
```

Methods

`get()`	*get the response_function object*

get()

get the response_function object

Return:

```
- ``response_function``
```

Usage:

Todo:

- add usage info
- create a sublime snippet for usage
- create cl-util for this method
- update the package tutorial if needed

```
usage code
```

2.11.14 subtract_background *(class)*

class subtract_background(*log*, *frame*, *orderTable*, *sofName=False*, *recipeName=False*, *settings=False*, *qcTable=False*, *productsTable=False*, *lamp=''*)

Bases: `object`

fit and subtract background flux from scattered light from frame

Key Arguments:

- `log` – logger
- `settings` – the settings dictionary
- `frame` – the frame to subtract background light from
- `recipeName` – name of the parent recipe
- `sofName` – the sof file name given to the parent recipe
- `orderTable` – the order geometry table
- `qcTable` – the data frame to collect measured QC metrics
- `productsTable` – the data frame to collect output products
- `lamp` – needed for UVB flats

Usage:

To setup your logger, settings and database connections, please use the `fundamentals` package (see tutorial here).

To fit and subtract the background from an image:

```
from soxspipe.commonutils import subtract_background
background = subtract_background(
    log=log,
    frame=myCCDDataObject,
    orderTable="/path/to/orderTable",
    settings=settings
)
backgroundFrame, backgroundSubtractedFrame = background.subtract()
```

Methods

`create_background_image`(rowFitOrder, ...)	*model the background image from intra-order flux detected*
`mask_order_locations`(orderPixelTable)	*mask the order locations and return the masked frame*
`subtract`()	*fit and subtract background light from frame*

create_background_image(*rowFitOrder*, *gaussianSigma*)

model the background image from intra-order flux detected

Key Arguments:

- `rowFitOrder` – order of the polynomial fit to flux in each row
- `gaussianSigma` – the sigma of the gaussian used to blur the final image

mask_order_locations(*orderPixelTable*)
mask the order locations and return the masked frame

Key Arguments:

- `orderPixelTable` – the order location in a pandas datafrmae.

subtract()
fit and subtract background light from frame

Return:

- `backgroundSubtractedFrame` – a CCDData object of the original input frame with fitted background light subtracted

2.11.15 subtract_sky *(class)*

class subtract_sky(*log*, *settings*, *recipeSettings*, *objectFrame*, *twoDMap*, *qcTable*, *productsTable*, *dispMap=False*, *sofName=False*, *recipeName='soxs-stare'*)
Bases: `object`

Subtract the sky background from a science image using the Kelson Method

A model of the sky-background is created using a method similar to that described in Kelson, D. (2003), *Optimal Techniques in Two-dimensional Spectroscopy: Background Subtraction for the 21st Century* (http://dx.doi.org/10.1086/375502). This model-background is then subtracted from the original science image to leave only non-sky flux.

Key Arguments:

- `log` – logger
- `settings` – the soxspipe settings dictionary
- `recipeSettings` – the recipe specific settings
- `objectFrame` – the image frame in need of sky subtraction
- `twoDMap` – 2D dispersion map image path
- `qcTable` – the data frame to collect measured QC metrics
- `productsTable` – the data frame to collect output products
- `dispMap` – path to dispersion map. Default *False*
- `sofName` – name of the originating SOF file. Default *False*
- `recipeName` – name of the recipe as it appears in the settings dictionary. Default *soxs-stare*

Usage:

To setup your logger and settings, please use the `fundamentals` package (see tutorial here).

To initiate a `subtract_sky` object, use the following:

Todo:

- add a tutorial about `subtract_sky` to documentation

```
from soxspipe.commonutils import subtract_sky
skymodel = subtract_sky(
    log=log,
    settings=settings,
    recipeSettings=recipeSettings,
    objectFrame=objectFrame,
    twoDMap=twoDMap,
    qcTable=qc,
    productsTable=products,
    dispMap=dispMap
)
skymodelCCDData, skySubtractedCCDData, qcTable, productsTable = skymodel.
↪subtract()
```

Methods

`add_data_to_placeholder_images`(. . .)	*add sky-model and sky-subtracted data to placeholder images*
`calculate_residuals`(skyPixelsDF, fluxcoeff, . . .)	*calculate residuals of the polynomial fits against the observed line positions*
`clip_object_slit_positions`(order_dataframes)	*clip out pixels flagged as an object*
`create_placeholder_images`()	*create placeholder images for the sky model and sky-subtracted frame*
`cross_dispersion_flux_normaliser`(orderDF)	*measure and normalise the flux in the cross-dispersion direction*
`fit_bspline_curve_to_sky`(imageMapOrder, . . .)	*fit a single-order univariate bspline to the unclipped sky pixels (wavelength vs flux)*
`get_over_sampled_sky_from_order`(imageMapOrder)	*unpack the over sampled sky from an order*
`plot_image_comparison`(objectFrame, . . .)	*generate a plot of original image, sky-model and sky-subtraction image*
`plot_sky_sampling`(order, . . . [, tck, . . .])	*generate a plot of sky sampling*
`rectify_order`(order, imageMapOrder[, . . .])	*rectify order on a fine slit-postion, wavelength grid*
`rolling_window_clipping`(imageMapOrderDF, . . .)	*clip pixels in a rolling wavelength window*
`subtract`()	*generate and subtract a sky-model from the input frame*

add_data_to_placeholder_images(*imageMapOrderDF*, *skymodelCCDData*, *skySubtractedCCDData*)

add sky-model and sky-subtracted data to placeholder images

Key Arguments:

- `imageMapOrderDF` – single order dataframe from object image and 2D map
- `skymodelCCDData` – the sky model
- `skySubtractedCCDData` – the sky-subtracted data

Usage:

```
self.add_data_to_placeholder_images(imageMapOrderSkyOnly, skymodelCCDData,␣
↪skySubtractedCCDData)
```

calculate_residuals(*skyPixelsDF*, *fluxcoeff*, *orderDeg*, *wavelengthDeg*, *slitDeg*, *writeQCs=False*)

calculate residuals of the polynomial fits against the observed line positions

Key Arguments:

- `skyPixelsDF` – the predicted line list as a data frame
- `fluxcoeff` – the flux-coefficients
- `orderDeg` – degree of the order fitting
- `wavelengthDeg` – degree of wavelength fitting
- `slitDeg` – degree of the slit fitting (False for single pinhole)
- `writeQCs` – write the QCs to dataframe? Default *False*

Return:

- `residuals` – combined x-y residuals
- `mean` – the mean of the combine residuals
- `std` – the stdev of the combine residuals
- `median` – the median of the combine residuals

clip_object_slit_positions(*order_dataframes*, *aggressive=False*)

clip out pixels flagged as an object

Key Arguments:

- `order_dataframes` – a list of order data-frames with pixels potentially containing the object flagged.

Return:

- `order_dataframes` – the order dataframes with the object(s) slit-ranges clipped
- `aggressive` – mask entire slit range where an object is expected to lie. Default *False*

Usage:

```
allimageMapOrder = self.clip_object_slit_positions(
    allimageMapOrder,
    aggressive=True
)
```

create_placeholder_images()

create placeholder images for the sky model and sky-subtracted frame

Return:

- `skymodelCCDData` – placeholder for sky model image
- `skySubtractedCCDData` – placeholder for sky-subtracted image

Usage:

```
skymodelCCDData, skySubtractedCCDData = self.create_placeholder_images()
```

cross_dispersion_flux_normaliser(*orderDF*)

measure and normalise the flux in the cross-dispersion direction

Key Arguments:

- `orderDF` – a single order dataframe containing sky-subtraction flux residuals used to determine and remove a slit-illumination correction

Return:

```
- `correctedOrderDF` -- dataframe with slit-illumination correction factor␣
↪added (flux-normaliser)
```

Usage:

```
correctedOrderDF = self.cross_dispersion_flux_normaliser(orderDF)
```

fit_bspline_curve_to_sky(*imageMapOrder*, *bspline_order*)
fit a single-order univariate bspline to the unclipped sky pixels (wavelength vs flux)

Key Arguments:

- `imageMapOrder` – single order dataframe, containing sky flux with object(s) and CRHs removed
- `order` – the order number
- `bspline_order` – the order of the bspline to fit

Return:

- `imageMapOrder` – same `imageMapOrder` as input but now with `sky_model` (bspline fit of the sky) and `sky_subtracted_flux` columns
- `tck` – the fitted bspline components. t for knots, c of coefficients, k for order

Usage:

```
imageMapOrder, tck = self.fit_bspline_curve_to_sky(
    imageMapOrder,
    bspline_order
)
```

get_over_sampled_sky_from_order(*imageMapOrder*, *clipBPs=True*, *clipSlitEdge=False*)
unpack the over sampled sky from an order

Key Arguments:

- `imageMapOrder` – single order dataframe from object image and 2D map
- `clipBPs` – clip bad-pixels? Deafult *True*
- `clipSlitEdge` – clip the slit edges. Percentage of slit width to clip. Default *False*

Return:

```
- `imageMapOrderWithObject` -- input order dataframe with outlying pixels␣
↪masked
- `imageMapOrder` -- input order dataframe with outlying pixels masked AND␣
↪object pixels masked
```

Usage:

```
imageMapOrderWithObject, imageMapOrderSkyOnly = skymodel.get_over_sampled_sky_
↪from_order(imageMapOrder, o, ignoreBP=False, clipSlitEdge=0.00)
```

plot_image_comparison(*objectFrame*, *skyModelFrame*, *skySubFrame*)
generate a plot of original image, sky-model and sky-subtraction image

Key Arguments:

- `objectFrame` – object frame
- `skyModelFrame` – sky model frame
- `skySubFrame` – sky subtracted frame

Return:

- `filePath` – path to the plot pdf

plot_sky_sampling(*order*, *imageMapOrderWithObjectDF*, *imageMapOrderDF*, *tck=False*, *knotLocations=False*)

generate a plot of sky sampling

Key Arguments:

- `order` – the order number.
- `imageMapOrderWithObjectDF` – dataframe with various processed data without object clipped
- `imageMapOrderDF` – dataframe with various processed data for order
- `tck` – spline parameters to replot
- `knotLocations` – wavelength locations of all knots used in the fit

Return:

- `filePath` – path to the generated QC plots PDF

Usage:

```
self.plot_sky_sampling(
    order=myOrder,
    imageMapOrderWithObjectDF=imageMapOrderWithObject,
    imageMapOrderDF=imageMapOrder
)
```

rectify_order(*order*, *imageMapOrder*, *remove_clipped=False*, *conserve_flux=False*)

rectify order on a fine slit-postion, wavelength grid

Key Arguments:

- `order` – order to be rectified
- `imageMapOrder` – the image map for this order (wavelength, slit-position and flux for each physical pixel
- `conserve_flux` – conserve the flux budget across the entire image

Return:

```
- None
```

Usage:

```
usage code
```

Todo:

- add usage info

- create a sublime snippet for usage
- write a command-line tool for this method
- update package tutorial with command-line tool info if needed

rolling_window_clipping(*imageMapOrderDF*, *windowSize*, *sigma_clip_limit=5*, *max_iterations=10*, *median_centre_func=False*)

clip pixels in a rolling wavelength window

Key Arguments:

- `imageMapOrderDF` – dataframe with various processed data for a given order
- `windowSize` – the window-size used to perform rolling window clipping (number of data-points)
- `sigma_clip_limit` – clip data values straying beyond this sigma limit. Default *5*
- `max_iterations` – maximum number of iterations when clipping
- `median_centre_func` – use a median centre function for rolling window instead of quantile (use to clip most deviate pixels only). Default *False*

Return:

- `imageMapOrderDF` – image order dataframe with 'clipped' == True for those pixels that have been clipped via rolling window clipping

Usage:

```
imageMapOrder = self.rolling_window_clipping(
    imageMapOrderDF=imageMapOrder,
    windowSize=23,
    sigma_clip_limit=4,
    max_iterations=10,
    median_centre_func=True
)
```

subtract()

generate and subtract a sky-model from the input frame

Return:

- `skymodelCCDData` – CCDData object containing the model sky frame
- `skySubtractedCCDData` – CCDData object containing the sky-subtacted frame
- `qcTable` – the data frame containing measured QC metrics
- `productsTable` – the data frame containing collected output products

2.11.16 MaxFilter *(class)*

class MaxFilter(*max_level*)

Bases: `object`

Methods

`filter`(record)	

2.11.17 soxs_disp_solution *(class)*

class soxs_disp_solution(*log*, *settings=False*, *inputFrames=[]*, *verbose=False*, *overwrite=False*)

Bases: `soxspipe.recipes._base_recipe_._base_recipe_`

generate a first approximation of the dispersion solution from single pinhole frames

Key Arguments

- `log` – logger
- `settings` – the settings dictionary
- `inputFrames` – input fits frames. Can be a directory, a set-of-files (SOF) file or a list of fits frame paths.
- `verbose` – verbose. True or False. Default *False*
- `overwrite` – overwrite the prodcut file if it already exists. Default *False*

Usage

```
from soxspipe.recipes import soxs_disp_solution
disp_map_path = soxs_disp_solution(
    log=log,
    settings=settings,
    inputFrames=sofPath
).produce_product()
```

Todo:

- add a tutorial about `soxs_disp_solution` to documentation

Methods

`clean_up`()	*update product status in DB and remove intermediate files once recipe is complete*
`clip_and_stack`(frames, recipe[, ...])	*mean combine input frames after sigma-clipping outlying pixels using a median value with median absolute deviation (mad) as the deviation function*
`detrend`(inputFrame[, master_bias, dark, ...])	*subtract calibration frames from an input frame*
`get_recipe_settings`()	*get the recipe and arm specific settings*

continues on next page

Table 26 – continued from previous page

`prepare_frames`([save])	*prepare raw frames by converting pixel data from ADU to electrons and adding mask and uncertainty extensions*
`produce_product`()	*generate a fisrt guess of the dispersion solution*
`qc_median_flux_level`(frame[, frameType, . . .])	*calculate the median flux level in the frame, excluding masked pixels*
`qc_ron`([frameType, frameName, masterFrame, . . .])	*calculate the read-out-noise from bias/dark frames*
`report_output`([rformat])	*a method to report QC values alongside intermediate and final products*
`subtract_mean_flux_level`(rawFrame)	*iteratively median sigma-clip raw bias data frames before calculating and removing the mean bias level*
`update_fits_keywords`(frame)	*update fits keywords to comply with ESO Phase 3 standards*
`verify_input_frames`()	*verify input frames match those required by the ``soxs_disp_solution`` recipe*
`xsh2soxs`(frame)	*perform some massaging of the xshooter data so it more closely resembles soxs data - this function can be removed once code is production ready*

clean_up()

update product status in DB and remove intermediate files once recipe is complete

Usage

```
recipe.clean_up()
```

clip_and_stack(*frames*, *recipe*, *ignore_input_masks=False*, *post_stack_clipping=True*)

mean combine input frames after sigma-clipping outlying pixels using a median value with median absolute deviation (mad) as the deviation function

Key Arguments:

- `frames` – an ImageFileCollection of the frames to stack or a list of CCDData objects
- `recipe` – the name of recipe needed to read the correct settings from the yaml files
- `ignore_input_masks` – ignore the input masks during clip and stacking?
- `post_stack_clipping` – allow cross-plane clipping on combined frame. Clipping settings in setting file. Default *True*.

Return:

- `combined_frame` – the combined master frame (with updated bad-pixel and uncertainty maps)

Usage:

This snippet can be used within the recipe code to combine individual (using bias frames as an example):

```
combined_bias_mean = self.clip_and_stack(
    frames=self.inputFrames, recipe="soxs_mbias", ignore_input_masks=False,␣
↪post_stack_clipping=True)
```

detrend(*inputFrame*, *master_bias=False*, *dark=False*, *master_flat=False*, *order_table=False*)

subtract calibration frames from an input frame

Key Arguments:

- `inputFrame` – the input frame to have calibrations subtracted. CCDData object.
- `master_bias` – the master bias frame to be subtracted. CCDData object. Default *False*.
- `dark` – a dark frame to be subtracted. CCDData object. Default *False*.
- `master_flat` – divided input frame by this master flat frame. CCDData object. Default *False*.
- `order_table` – order table with order edges defined. Used to subtract scattered light background from frames. Default *False*.

Return:

- `calibration_subtracted_frame` – the input frame with the calibration frame(s) subtracted. CCDData object.

Usage:

Within a soxspipe recipe use `detrend` like so:

```
myCalibratedFrame = self.detrend(
    inputFrame=inputFrameCCDObject, master_bias=masterBiasCCDObject,␣
↪dark=darkCCDObject)
```

Todo:

- code needs written to scale dark frame to exposure time of science/calibration frame

get_recipe_settings()

get the recipe and arm specific settings

Return:

- `recipeSettings` – the recipe specific settings

Usage:

```
usage code
```

prepare_frames(*save=False*)

prepare raw frames by converting pixel data from ADU to electrons and adding mask and uncertainty extensions

Key Arguments:

- `save` – save out the prepared frame to the intermediate products directory. Default False.

Return:

- `preframes` – the new image collection containing the prepared frames

Usage

Usually called within a recipe class once the input frames have been selected and verified (see `soxs_mbias` code for example):

```
self.inputFrames = self.prepare_frames(
    save=self.settings["save-intermediate-products"])
```

produce_product()

generate a fisrt guess of the dispersion solution

Return:

- `productPath` – the path to the first guess dispersion map

qc_median_flux_level (*frame*, *frameType='MBIAS'*, *frameName='master bias'*, *medianFlux=False*)

calculate the median flux level in the frame, excluding masked pixels

Key Arguments:

- `frame` – the frame (CCDData object) to determine the median level.
- `frameType` – the type of the frame for reporting QC values Default "MBIAS"
- `frameName` – the name of the frame in human readable words. Default "master bias"
- `medianFlux` – if serendipitously calculated elsewhere don't recalculate. Default *False*

Return:

- `medianFlux` – median flux level in electrons

Usage:

```
medianFlux = self.qc_median_flux_level(
    frame=myFrame,
    frameType="MBIAS",
    frameName="master bias")
```

qc_ron (*frameType=False*, *frameName=False*, *masterFrame=False*, *rawRon=False*, *masterRon=False*)

calculate the read-out-noise from bias/dark frames

Key Arguments:

- `frameType` – the type of the frame for reporting QC values. Default *False*
- `frameName` – the name of the frame in human readable words. Default *False*
- `masterFrame` – the master frame (only makes sense to measure RON on master bias). Default *False*
- `rawRon` – if serendipitously calculated elsewhere don't recalculate. Default *False*
- `masterRon` – if serendipitously calculated elsewhere don't recalculate. Default *False*

Return:

- `rawRon` – raw read-out-noise in electrons
- `masterRon` – combined read-out-noise in mbias

Usage:

```
rawRon, mbiasRon = self.qc_ron(
    frameType="MBIAS",
    frameName="master bias",
    masterFrame=masterFrame
)
```

report_output (*rformat='stdout'*)

a method to report QC values alongside intermediate and final products

Key Arguments:

- `rformat` – the format to outout reports as. Default *stdout*. [stdout|. . .]

Usage:

```
self.report_output(rformat="stdout")
```

subtract_mean_flux_level(*rawFrame*)
iteratively median sigma-clip raw bias data frames before calculating and removing the mean bias level

Key Arguments:

- `rawFrame` – the raw bias frame

Return:

```
- `meanFluxLevel` -- the frame mean bias level
- `fluxStd` -- the standard deviation of the flux distribution (RON)
- `noiseFrame` -- the raw bias frame with mean bias level removed
```

Usage:

```
meanFluxLevel, fluxStd, noiseFrame = self.subtract_mean_flux_level(rawFrame)
```

update_fits_keywords(*frame*)
update fits keywords to comply with ESO Phase 3 standards

Key Arguments:

- `frame` – the frame to update

Return:

```
- None
```

Usage:

```
usage code
```

Todo:

- add usage info
- create a sublime snippet for usage
- write a command-line tool for this method
- update package tutorial with command-line tool info if needed

verify_input_frames()
verify input frames match those required by the ``soxs_disp_solution`` recipe

If the fits files conform to required input for the recipe everything will pass silently, otherwise an exception shall be raised.

xsh2soxs(*frame*)
perform some massaging of the xshooter data so it more closely resembles soxs data - this function can be removed once code is production ready

Key Arguments:

- `frame` – the CCDDate frame to manipulate

Return:

- `frame` – the manipulated soxspipe-ready frame

Usage:

```
frame = self.xsh2soxs(frame)
```

2.11.18 soxs_mbias *(class)*

class soxs_mbias(*log*, *settings=False*, *inputFrames=[]*, *verbose=False*, *overwrite=False*)

Bases: `soxspipe.recipes._base_recipe_._base_recipe_`

The `soxs_mbias` *recipe is used to generate a master-bias frame from a set of input raw bias frames. The recipe is used only for the UV-VIS arm as NIR frames have bias (and dark current) removed by subtracting an off-frame of equal expsoure length.*

Key Arguments

- `log` – logger
- `settings` – the settings dictionary
- `inputFrames` – input fits frames. Can be a directory, a set-of-files (SOF) file or a list of fits frame paths.
- `verbose` – verbose. True or False. Default *False*
- `overwrite` – overwrite the prodcut file if it already exists. Default *False*

Usage

```
from soxspipe.recipes import soxs_mbias
mbiasFrame = soxs_mbias(
    log=log,
    settings=settings,
    inputFrames=fileList
).produce_product()
```

Todo:

- add a tutorial about `soxs_mbias` to documentation

Methods

`clean_up()`	*update product status in DB and remove intermediate files once recipe is complete*
`clip_and_stack(frames, recipe[, ...])`	*mean combine input frames after sigma-clipping outlying pixels using a median value with median absolute deviation (mad) as the deviation function*
`detrend(inputFrame[, master_bias, dark, ...])`	*subtract calibration frames from an input frame*
`get_recipe_settings()`	*get the recipe and arm specific settings*
`prepare_frames([save])`	*prepare raw frames by converting pixel data from ADU to electrons and adding mask and uncertainty extensions*

continues on next page

Table 27 – continued from previous page

`produce_product`()	*generate a master bias frame*
`qc_bias_structure`(combined_bias_mean)	*calculate the structure of the bias*
`qc_median_flux_level`(frame[, frameType, ...])	*calculate the median flux level in the frame, excluding masked pixels*
`qc_periodic_pattern_noise`(frames)	*calculate the periodic pattern noise based on the raw input bias frames*
`qc_ron`([frameType, frameName, masterFrame, ...])	*calculate the read-out-noise from bias/dark frames*
`report_output`([rformat])	*a method to report QC values alongside intermediate and final products*
`subtract_mean_flux_level`(rawFrame)	*iteratively median sigma-clip raw bias data frames before calculating and removing the mean bias level*
`update_fits_keywords`(frame)	*update fits keywords to comply with ESO Phase 3 standards*
`verify_input_frames`()	*verify the input frame match those required by the soxs_mbias recipe*
`xsh2soxs`(frame)	*perform some massaging of the xshooter data so it more closely resembles soxs data - this function can be removed once code is production ready*

clean_up()

update product status in DB and remove intermediate files once recipe is complete

Usage

```
recipe.clean_up()
```

clip_and_stack(*frames*, *recipe*, *ignore_input_masks=False*, *post_stack_clipping=True*)

mean combine input frames after sigma-clipping outlying pixels using a median value with median absolute deviation (mad) as the deviation function

Key Arguments:

- `frames` – an ImageFileCollection of the frames to stack or a list of CCDData objects
- `recipe` – the name of recipe needed to read the correct settings from the yaml files
- `ignore_input_masks` – ignore the input masks during clip and stacking?
- `post_stack_clipping` – allow cross-plane clipping on combined frame. Clipping settings in setting file. Default *True*.

Return:

- `combined_frame` – the combined master frame (with updated bad-pixel and uncertainty maps)

Usage:

This snippet can be used within the recipe code to combine individual (using bias frames as an example):

```
combined_bias_mean = self.clip_and_stack(
    frames=self.inputFrames, recipe="soxs_mbias", ignore_input_masks=False,␣
↪post_stack_clipping=True)
```

detrend(*inputFrame*, *master_bias=False*, *dark=False*, *master_flat=False*, *order_table=False*)

subtract calibration frames from an input frame

Key Arguments:

- `inputFrame` – the input frame to have calibrations subtracted. CCDData object.
- `master_bias` – the master bias frame to be subtracted. CCDData object. Default *False*.
- `dark` – a dark frame to be subtracted. CCDData object. Default *False*.
- `master_flat` – divided input frame by this master flat frame. CCDData object. Default *False*.
- `order_table` – order table with order edges defined. Used to subtract scattered light background from frames. Default *False*.

Return:

- `calibration_subtracted_frame` – the input frame with the calibration frame(s) subtracted. CCDData object.

Usage:

Within a soxspipe recipe use `detrend` like so:

```
myCalibratedFrame = self.detrend(
    inputFrame=inputFrameCCDObject, master_bias=masterBiasCCDObject,␣
↪dark=darkCCDObject)
```

Todo:

- code needs written to scale dark frame to exposure time of science/calibration frame

get_recipe_settings()

get the recipe and arm specific settings

Return:

- `recipeSettings` – the recipe specific settings

Usage:

```
usage code
```

prepare_frames(*save=False*)

prepare raw frames by converting pixel data from ADU to electrons and adding mask and uncertainty extensions

Key Arguments:

- `save` – save out the prepared frame to the intermediate products directory. Default False.

Return:

- `preframes` – the new image collection containing the prepared frames

Usage

Usually called within a recipe class once the input frames have been selected and verified (see `soxs_mbias` code for example):

```
self.inputFrames = self.prepare_frames(
    save=self.settings["save-intermediate-products"])
```

produce_product()

generate a master bias frame

Return:

- `productPath` – the path to the master bias frame

qc_bias_structure(*combined_bias_mean*)
calculate the structure of the bias

Key Arguments:

- `combined_bias_mean` – the mbias frame

Return:

- `structx` – slope of BIAS in X direction
- `structx` – slope of BIAS in Y direction

Usage:

```
structx, structy = self.qc_bias_structure(combined_bias_mean)
```

qc_median_flux_level(*frame, frameType='MBIAS', frameName='master bias', medianFlux=False*)
calculate the median flux level in the frame, excluding masked pixels

Key Arguments:

- `frame` – the frame (CCDData object) to determine the median level.
- `frameType` – the type of the frame for reporting QC values Default "MBIAS"
- `frameName` – the name of the frame in human readable words. Default "master bias"
- `medianFlux` – if serendipitously calculated elsewhere don't recalculate. Default *False*

Return:

- `medianFlux` – median flux level in electrons

Usage:

```
medianFlux = self.qc_median_flux_level(
    frame=myFrame,
    frameType="MBIAS",
    frameName="master bias")
```

qc_periodic_pattern_noise(*frames*)
calculate the periodic pattern noise based on the raw input bias frames

A 2D FFT is applied to each of the raw bias frames and the standard deviation and median absolute deviation calcualted for each result. The maximum std/mad is then added as the ppnmax QC in the master bias frame header.

Key Arguments:

- `frames` – the raw bias frames (imageFileCollection)

Return:

```
- ``ppnmax``
```

Usage:

```
self.qc_periodic_pattern_noise(frames=self.inputFrames)
```

qc_ron (*frameType=False, frameName=False, masterFrame=False, rawRon=False, masterRon=False*)
calculate the read-out-noise from bias/dark frames

Key Arguments:

- `frameType` – the type of the frame for reporting QC values. Default *False*
- `frameName` – the name of the frame in human readable words. Default *False*
- `masterFrame` – the master frame (only makes sense to measure RON on master bias). Default *False*
- `rawRon` – if serendipitously calculated elsewhere don't recalculate. Default *False*
- `masterRon` – if serendipitously calculated elsewhere don't recalculate. Default *False*

Return:

- `rawRon` – raw read-out-noise in electrons
- `masterRon` – combined read-out-noise in mbias

Usage:

```
rawRon, mbiasRon = self.qc_ron(
    frameType="MBIAS",
    frameName="master bias",
    masterFrame=masterFrame
)
```

report_output (*rformat='stdout'*)
a method to report QC values alongside intermediate and final products

Key Arguments:

- `rformat` – the format to outout reports as. Default *stdout*. [stdout|. . .]

Usage:

```
self.report_output(rformat="stdout")
```

subtract_mean_flux_level (*rawFrame*)
iteratively median sigma-clip raw bias data frames before calculating and removing the mean bias level

Key Arguments:

- `rawFrame` – the raw bias frame

Return:

```
- `meanFluxLevel` -- the frame mean bias level
- `fluxStd` -- the standard deviation of the flux distribution (RON)
- `noiseFrame` -- the raw bias frame with mean bias level removed
```

Usage:

```
meanFluxLevel, fluxStd, noiseFrame = self.subtract_mean_flux_level(rawFrame)
```

update_fits_keywords (*frame*)
update fits keywords to comply with ESO Phase 3 standards

Key Arguments:

- `frame` – the frame to update

Return:

```
- None
```

Usage:

```
usage code
```

Todo:

- add usage info
- create a sublime snippet for usage
- write a command-line tool for this method
- update package tutorial with command-line tool info if needed

---

verify_input_frames()

verify the input frame match those required by the soxs_mbias recipe

If the fits files conform to required input for the recipe everything will pass silently, otherwise an exception shall be raised.

xsh2soxs(*frame*)

perform some massaging of the xshooter data so it more closely resembles soxs data - this function can be removed once code is production ready

Key Arguments:

- `frame` – the CCDDate frame to manipulate

Return:

- `frame` – the manipulated soxspipe-ready frame

Usage:

```
frame = self.xsh2soxs(frame)
```

2.11.19 soxs_mdark *(class)*

class soxs_mdark(*log*, *settings=False*, *inputFrames=[]*, *verbose=False*, *overwrite=False*)

Bases: `soxspipe.recipes._base_recipe_._base_recipe_`

The soxs_mdark recipe

Key Arguments

- `log` – logger
- `settings` – the settings dictionary
- `inputFrames` – input fits frames. Can be a directory, a set-of-files (SOF) file or a list of fits frame paths.
- `verbose` – verbose. True or False. Default *False*
- `overwrite` – overwrite the prodcut file if it already exists. Default *False*

Usage

```
from soxspipe.recipes import soxs_mdark
mdarkFrame = soxs_mdark(
    log=log,
    settings=settings,
    inputFrames=fileList
)..produce_product()
```

Todo:

- add a tutorial about `soxs_mdark` to documentation

Methods

`clean_up()`	*update product status in DB and remove intermediate files once recipe is complete*
`clip_and_stack(frames, recipe[, …])`	*mean combine input frames after sigma-clipping outlying pixels using a median value with median absolute deviation (mad) as the deviation function*
`detrend(inputFrame[, master_bias, dark, …])`	*subtract calibration frames from an input frame*
`get_recipe_settings()`	*get the recipe and arm specific settings*
`prepare_frames([save])`	*prepare raw frames by converting pixel data from ADU to electrons and adding mask and uncertainty extensions*
`produce_product()`	*generate a master dark frame*
`qc_median_flux_level(frame[, frameType, …])`	*calculate the median flux level in the frame, excluding masked pixels*
`qc_ron([frameType, frameName, masterFrame, …])`	*calculate the read-out-noise from bias/dark frames*
`report_output([rformat])`	*a method to report QC values alongside intermediate and final products*
`subtract_mean_flux_level(rawFrame)`	*iteratively median sigma-clip raw bias data frames before calculating and removing the mean bias level*
`update_fits_keywords(frame)`	*update fits keywords to comply with ESO Phase 3 standards*
`verify_input_frames()`	*verify input frame match those required by the soxs_mdark recipe*
`xsh2soxs(frame)`	*perform some massaging of the xshooter data so it more closely resembles soxs data - this function can be removed once code is production ready*

clean_up()

update product status in DB and remove intermediate files once recipe is complete

Usage

```
recipe.clean_up()
```

clip_and_stack(*frames, recipe, ignore_input_masks=False, post_stack_clipping=True*)

mean combine input frames after sigma-clipping outlying pixels using a median value with median absolute deviation (mad) as the deviation function

Key Arguments:

- `frames` – an ImageFileCollection of the frames to stack or a list of CCDData objects
- `recipe` – the name of recipe needed to read the correct settings from the yaml files
- `ignore_input_masks` – ignore the input masks during clip and stacking?
- `post_stack_clipping` – allow cross-plane clipping on combined frame. Clipping settings in setting file. Default *True*.

Return:

- `combined_frame` – the combined master frame (with updated bad-pixel and uncertainty maps)

Usage:

This snippet can be used within the recipe code to combine individual (using bias frames as an example):

```
combined_bias_mean = self.clip_and_stack(
    frames=self.inputFrames, recipe="soxs_mbias", ignore_input_masks=False,␣
↪post_stack_clipping=True)
```

detrend(*inputFrame*, *master_bias=False*, *dark=False*, *master_flat=False*, *order_table=False*)
subtract calibration frames from an input frame

Key Arguments:

- `inputFrame` – the input frame to have calibrations subtracted. CCDData object.
- `master_bias` – the master bias frame to be subtracted. CCDData object. Default *False*.
- `dark` – a dark frame to be subtracted. CCDData object. Default *False*.
- `master_flat` – divided input frame by this master flat frame. CCDData object. Default *False*.
- `order_table` – order table with order edges defined. Used to subtract scattered light background from frames. Default *False*.

Return:

- `calibration_subtracted_frame` – the input frame with the calibration frame(s) subtracted. CCDData object.

Usage:

Within a soxspipe recipe use `detrend` like so:

```
myCalibratedFrame = self.detrend(
    inputFrame=inputFrameCCDObject, master_bias=masterBiasCCDObject,␣
↪dark=darkCCDObject)
```

Todo:

- code needs written to scale dark frame to exposure time of science/calibration frame

get_recipe_settings()
get the recipe and arm specific settings

Return:

- `recipeSettings` – the recipe specific settings

Usage:

```
usage code
```

prepare_frames(*save=False*)

prepare raw frames by converting pixel data from ADU to electrons and adding mask and uncertainty extensions

Key Arguments:

- `save` – save out the prepared frame to the intermediate products directory. Default False.

Return:

- `preframes` – the new image collection containing the prepared frames

Usage

Usually called within a recipe class once the input frames have been selected and verified (see `soxs_mbias` code for example):

```
self.inputFrames = self.prepare_frames(
    save=self.settings["save-intermediate-products"])
```

produce_product()

generate a master dark frame

Return:

- `productPath` – the path to master dark frame

qc_median_flux_level(*frame, frameType='MBIAS', frameName='master bias', medianFlux=False*)

calculate the median flux level in the frame, excluding masked pixels

Key Arguments:

- `frame` – the frame (CCDData object) to determine the median level.
- `frameType` – the type of the frame for reporting QC values Default "MBIAS"
- `frameName` – the name of the frame in human readable words. Default "master bias"
- `medianFlux` – if serendipitously calculated elsewhere don't recalculate. Default *False*

Return:

- `medianFlux` – median flux level in electrons

Usage:

```
medianFlux = self.qc_median_flux_level(
    frame=myFrame,
    frameType="MBIAS",
    frameName="master bias")
```

qc_ron(*frameType=False, frameName=False, masterFrame=False, rawRon=False, masterRon=False*)

calculate the read-out-noise from bias/dark frames

Key Arguments:

- `frameType` – the type of the frame for reporting QC values. Default *False*
- `frameName` – the name of the frame in human readable words. Default *False*

- `masterFrame` – the master frame (only makes sense to measure RON on master bias). Default *False*
- `rawRon` – if serendipitously calculated elsewhere don't recalculate. Default *False*
- `masterRon` – if serendipitously calculated elsewhere don't recalculate. Default *False*

Return:

- `rawRon` – raw read-out-noise in electrons
- `masterRon` – combined read-out-noise in mbias

Usage:

```
rawRon, mbiasRon = self.qc_ron(
    frameType="MBIAS",
    frameName="master bias",
    masterFrame=masterFrame
)
```

`report_output` (*rformat='stdout'*)

a method to report QC values alongside intermediate and final products

Key Arguments:

- `rformat` – the format to outout reports as. Default *stdout*. [stdout|. . .]

Usage:

```
self.report_output(rformat="stdout")
```

`subtract_mean_flux_level` (*rawFrame*)

iteratively median sigma-clip raw bias data frames before calculating and removing the mean bias level

Key Arguments:

- `rawFrame` – the raw bias frame

Return:

```
- `meanFluxLevel` -- the frame mean bias level
- `fluxStd` -- the standard deviation of the flux distribution (RON)
- `noiseFrame` -- the raw bias frame with mean bias level removed
```

Usage:

```
meanFluxLevel, fluxStd, noiseFrame = self.subtract_mean_flux_level(rawFrame)
```

`update_fits_keywords` (*frame*)

update fits keywords to comply with ESO Phase 3 standards

Key Arguments:

- `frame` – the frame to update

Return:

```
- None
```

Usage:

```
usage code
```

Todo:

- add usage info
- create a sublime snippet for usage
- write a command-line tool for this method
- update package tutorial with command-line tool info if needed

verify_input_frames()

verify input frame match those required by the soxs_mdark recipe

If the fits files conform to required input for the recipe everything will pass silently, otherwise an exception shall be raised.

xsh2soxs(*frame*)

perform some massaging of the xshooter data so it more closely resembles soxs data - this function can be removed once code is production ready

Key Arguments:

- `frame` – the CCDDate frame to manipulate

Return:

- `frame` – the manipulated soxspipe-ready frame

Usage:

```
frame = self.xsh2soxs(frame)
```

2.11.20 soxs_mflat *(class)*

class soxs_mflat(*log*, *settings=False*, *inputFrames=[]*, *verbose=False*, *overwrite=False*)

Bases: `soxspipe.recipes._base_recipe_._base_recipe_`

The soxs_mflat recipe

Key Arguments

- `log` – logger
- `settings` – the settings dictionary
- `inputFrames` – input fits frames. Can be a directory, a set-of-files (SOF) file or a list of fits frame paths.
- `verbose` – verbose. True or False. Default *False*
- `overwrite` – overwrite the prodcut file if it already exists. Default *False*

Usage

```
from soxspipe.recipes import soxs_mflat
recipe = soxs_mflat(
    log=log,
    settings=settings,
```

(continues on next page)

(continued from previous page)

```
    inputFrames=fileList
)
mflatFrame = recipe.produce_product()
```

Todo:

- add a tutorial about `soxs_mflat` to documentation

Methods

`calibrate_frame_set()`	*given all of the input data calibrate the frames by subtracting bias and/or dark*
`clean_up()`	*update product status in DB and remove intermediate files once recipe is complete*
`clip_and_stack(frames, recipe[, ...])`	*mean combine input frames after sigma-clipping outlying pixels using a median value with median absolute deviation (mad) as the deviation function*
`detrend(inputFrame[, master_bias, dark, ...])`	*subtract calibration frames from an input frame*
`find_uvb_overlap_order_and_scale(...)`	*find uvb order where both lamps produce a similar flux.
`get_recipe_settings()`	*get the recipe and arm specific settings*
`mask_low_sens_pixels(frame, orderTablePath)`	*add low-sensitivity pixels to bad-pixel mask*
`normalise_flats(inputFlats, orderTablePath)`	*determine the median exposure for each flat frame and normalise the flux to that level*
`prepare_frames([save])`	*prepare raw frames by converting pixel data from ADU to electrons and adding mask and uncertainty extensions*
`produce_product()`	*generate the master flat frames updated order location table (with egde detection)*
`qc_median_flux_level(frame[, frameType, ...])`	*calculate the median flux level in the frame, excluding masked pixels*
`qc_ron([frameType, frameName, masterFrame, ...])`	*calculate the read-out-noise from bias/dark frames*
`report_output([rformat])`	*a method to report QC values alongside intermediate and final products*
`stitch_uv_mflats(medianOrderFluxDF, ...)`	*return a master UV-VIS flat frame after slicing and stitch the UV-VIS D-Lamp and QTH-Lamp flat frames*
`subtract_mean_flux_level(rawFrame)`	*iteratively median sigma-clip raw bias data frames before calculating and removing the mean bias level*
`update_fits_keywords(frame)`	*update fits keywords to comply with ESO Phase 3 standards*
`verify_input_frames()`	*verify the input frames match those required by the soxs_mflat recipe*
`xsh2soxs(frame)`	*perform some massaging of the xshooter data so it more closely resembles soxs data - this function can be removed once code is production ready*

calibrate_frame_set()

given all of the input data calibrate the frames by subtracting bias and/or dark

Return:

- `calibratedFlats` – the calibrated frames

clean_up()

update product status in DB and remove intermediate files once recipe is complete

Usage

```
recipe.clean_up()
```

clip_and_stack(*frames*, *recipe*, *ignore_input_masks=False*, *post_stack_clipping=True*)

mean combine input frames after sigma-clipping outlying pixels using a median value with median absolute deviation (mad) as the deviation function

Key Arguments:

- `frames` – an ImageFileCollection of the frames to stack or a list of CCDData objects
- `recipe` – the name of recipe needed to read the correct settings from the yaml files
- `ignore_input_masks` – ignore the input masks during clip and stacking?
- `post_stack_clipping` – allow cross-plane clipping on combined frame. Clipping settings in setting file. Default *True*.

Return:

- `combined_frame` – the combined master frame (with updated bad-pixel and uncertainty maps)

Usage:

This snippet can be used within the recipe code to combine individual (using bias frames as an example):

```
combined_bias_mean = self.clip_and_stack(
    frames=self.inputFrames, recipe="soxs_mbias", ignore_input_masks=False,␣
↪post_stack_clipping=True)
```

detrend(*inputFrame*, *master_bias=False*, *dark=False*, *master_flat=False*, *order_table=False*)

subtract calibration frames from an input frame

Key Arguments:

- `inputFrame` – the input frame to have calibrations subtracted. CCDData object.
- `master_bias` – the master bias frame to be subtracted. CCDData object. Default *False*.
- `dark` – a dark frame to be subtracted. CCDData object. Default *False*.
- `master_flat` – divided input frame by this master flat frame. CCDData object. Default *False*.
- `order_table` – order table with order edges defined. Used to subtract scattered light background from frames. Default *False*.

Return:

- `calibration_subtracted_frame` – the input frame with the calibration frame(s) subtracted. CCDData object.

Usage:

Within a soxspipe recipe use `detrend` like so:

```
myCalibratedFrame = self.detrend(
    inputFrame=inputFrameCCDObject, master_bias=masterBiasCCDObject,␣
↪dark=darkCCDObject)
```

Todo:

- code needs written to scale dark frame to exposure time of science/calibration frame

find_uvb_overlap_order_and_scale(*dcalibratedFlats*, *qcalibratedFlats*)

find uvb order where both lamps produce a similar flux. This is the order at which the 2 lamp flats will be scaled and stitched together

Key Arguments:

- `qcalibratedFlats` – the QTH lamp calibration flats.
- `dcalibratedFlats` – D2 lamp calibration flats

Return:

- `order` – the order number where the lamp fluxes are similar

Usage:

```
overlapOrder = self.find_uvb_overlap_order_and_
↪scale(dcalibratedFlats=dcalibratedFlats, qcalibratedFlats=qcalibratedFlats)
```

get_recipe_settings()

get the recipe and arm specific settings

Return:

- `recipeSettings` – the recipe specific settings

Usage:

```
usage code
```

mask_low_sens_pixels(*frame*, *orderTablePath*, *returnMedianOrderFlux=False*, *writeQC=True*)

add low-sensitivity pixels to bad-pixel mask

Key Arguments:

- `frame` – the frame to work on
- `orderTablePath` – path to the order table
- `returnMedianOrderFlux` – return a table of the median order fluxes. Default *False*.
- `writeQC` – add the QCs to the QC table?

Return:

- `frame` – with BPM updated with low-sensitivity pixels
- `medianOrderFluxDF` – data-frame of the median order fluxes (if `returnMedianOrderFlux` is True)

normalise_flats(*inputFlats*, *orderTablePath*, *firstPassMasterFlat=False*, *lamp=''*)
determine the median exposure for each flat frame and normalise the flux to that level

Key Arguments:

- `inputFlats` – the input flat field frames
- `orderTablePath` – path to the order table
- `firstPassMasterFlat` – the first pass of the master flat. Default *False*

```
- `lamp` -- a lamp tag for QL plots
```

Return:

- `normalisedFrames` – the normalised flat-field frames (CCDData array)

prepare_frames(*save=False*)
prepare raw frames by converting pixel data from ADU to electrons and adding mask and uncertainty extensions

Key Arguments:

- `save` – save out the prepared frame to the intermediate products directory. Default False.

Return:

- `preframes` – the new image collection containing the prepared frames

Usage

Usually called within a recipe class once the input frames have been selected and verified (see `soxs_mbias` code for example):

```
self.inputFrames = self.prepare_frames(
    save=self.settings["save-intermediate-products"])
```

produce_product()
generate the master flat frames updated order location table (with egde detection)

Return:

- `productPath` – the path to the master flat frame

qc_median_flux_level(*frame*, *frameType='MBIAS'*, *frameName='master bias'*, *medianFlux=False*)
calculate the median flux level in the frame, excluding masked pixels

Key Arguments:

- `frame` – the frame (CCDData object) to determine the median level.
- `frameType` – the type of the frame for reporting QC values Default "MBIAS"
- `frameName` – the name of the frame in human readable words. Default "master bias"
- `medianFlux` – if serendipitously calculated elsewhere don't recalculate. Default *False*

Return:

- `medianFlux` – median flux level in electrons

Usage:

```
medianFlux = self.qc_median_flux_level(
    frame=myFrame,
    frameType="MBIAS",
    frameName="master bias")
```

qc_ron(*frameType=False*, *frameName=False*, *masterFrame=False*, *rawRon=False*, *masterRon=False*)

calculate the read-out-noise from bias/dark frames

Key Arguments:

- `frameType` – the type of the frame for reporting QC values. Default *False*
- `frameName` – the name of the frame in human readable words. Default *False*
- `masterFrame` – the master frame (only makes sense to measure RON on master bias). Default *False*
- `rawRon` – if serendipitously calculated elsewhere don't recalculate. Default *False*
- `masterRon` – if serendipitously calculated elsewhere don't recalculate. Default *False*

Return:

- `rawRon` – raw read-out-noise in electrons
- `masterRon` – combined read-out-noise in mbias

Usage:

```
rawRon, mbiasRon = self.qc_ron(
    frameType="MBIAS",
    frameName="master bias",
    masterFrame=masterFrame
)
```

report_output(*rformat='stdout'*)

a method to report QC values alongside intermediate and final products

Key Arguments:

- `rformat` – the format to outout reports as. Default *stdout*. [stdout|. . .]

Usage:

```
self.report_output(rformat="stdout")
```

stitch_uv_mflats(*medianOrderFluxDF*, *orderTablePath*)

return a master UV-VIS flat frame after slicing and stitch the UV-VIS D-Lamp and QTH-Lamp flat frames

Key Arguments:

- `medianOrderFluxDF` – data frame containing median order fluxes for D and QTH frames
- `orderTablePath` – the original order table paths from order-centre tracing

Return:

- `stitchedFlat` – the stitch D and QTH-Lamp master flat frame

Usage:

```
mflat = self.stitch_uv_mflats(medianOrderFluxDF)
```

subtract_mean_flux_level(*rawFrame*)

iteratively median sigma-clip raw bias data frames before calculating and removing the mean bias level

Key Arguments:

- `rawFrame` – the raw bias frame

Return:

```
- `meanFluxLevel` -- the frame mean bias level
- `fluxStd` -- the standard deviation of the flux distribution (RON)
- `noiseFrame` -- the raw bias frame with mean bias level removed
```

Usage:

```
meanFluxLevel, fluxStd, noiseFrame = self.subtract_mean_flux_level(rawFrame)
```

update_fits_keywords(*frame*)

update fits keywords to comply with ESO Phase 3 standards

Key Arguments:

- `frame` – the frame to update

Return:

```
- None
```

Usage:

```
usage code
```

Todo:

- add usage info
- create a sublime snippet for usage
- write a command-line tool for this method
- update package tutorial with command-line tool info if needed

verify_input_frames()

verify the input frames match those required by the soxs_mflat recipe

If the fits files conform to required input for the recipe everything will pass silently, otherwise an exception will be raised.

xsh2soxs(*frame*)

perform some massaging of the xshooter data so it more closely resembles soxs data - this function can be removed once code is production ready

Key Arguments:

- `frame` – the CCDDate frame to manipulate

Return:

- `frame` – the manipulated soxspipe-ready frame

Usage:

```
frame = self.xsh2soxs(frame)
```

2.11.21 soxs_order_centres *(class)*

class soxs_order_centres(*log*, *settings=False*, *inputFrames=[]*, *verbose=False*, *overwrite=False*)

Bases: `soxspipe.recipes._base_recipe_._base_recipe_`

The soxs_order_centres recipe

Key Arguments

- `log` – logger
- `settings` – the settings dictionary
- `inputFrames` – input fits frames. Can be a directory, a set-of-files (SOF) file or a list of fits frame paths.
- `verbose` – verbose. True or False. Default *False*
- `overwrite` – overwrite the prodcut file if it already exists. Default *False*

Usage

```
from soxspipe.recipes import soxs_order_centres
order_table = soxs_order_centres(
    log=log,
    settings=settings,
    inputFrames=a["inputFrames"]
).produce_product()
```

Todo:

- add a tutorial about `soxs_order_centres` to documentation

Methods

`clean_up()`	*update product status in DB and remove intermediate files once recipe is complete*
`clip_and_stack(frames, recipe[, ...])`	*mean combine input frames after sigma-clipping outlying pixels using a median value with median absolute deviation (mad) as the deviation function*
`detrend(inputFrame[, master_bias, dark, ...])`	*subtract calibration frames from an input frame*
`get_recipe_settings()`	*get the recipe and arm specific settings*
`prepare_frames([save])`	*prepare raw frames by converting pixel data from ADU to electrons and adding mask and uncertainty extensions*
`produce_product()`	*generate the order-table with polynomal fits of order-centres*
`qc_median_flux_level(frame[, frameType, ...])`	*calculate the median flux level in the frame, excluding masked pixels*
`qc_ron([frameType, frameName, masterFrame, ...])`	*calculate the read-out-noise from bias/dark frames*

continues on next page

Table 30 – continued from previous page

`report_output`([rformat])	*a method to report QC values alongside intermediate and final products*
`subtract_mean_flux_level`(rawFrame)	*iteratively median sigma-clip raw bias data frames before calculating and removing the mean bias level*
`update_fits_keywords`(frame)	*update fits keywords to comply with ESO Phase 3 standards*
`verify_input_frames`()	*verify input frames match those required by the soxs_order_centres recipe*
`xsh2soxs`(frame)	*perform some massaging of the xshooter data so it more closely resembles soxs data - this function can be removed once code is production ready*

`clean_up`()

update product status in DB and remove intermediate files once recipe is complete

Usage

```
recipe.clean_up()
```

`clip_and_stack`(*frames*, *recipe*, *ignore_input_masks=False*, *post_stack_clipping=True*)

mean combine input frames after sigma-clipping outlying pixels using a median value with median absolute deviation (mad) as the deviation function

Key Arguments:

- `frames` – an ImageFileCollection of the frames to stack or a list of CCDData objects
- `recipe` – the name of recipe needed to read the correct settings from the yaml files
- `ignore_input_masks` – ignore the input masks during clip and stacking?
- `post_stack_clipping` – allow cross-plane clipping on combined frame. Clipping settings in setting file. Default *True*.

Return:

- `combined_frame` – the combined master frame (with updated bad-pixel and uncertainty maps)

Usage:

This snippet can be used within the recipe code to combine individual (using bias frames as an example):

```
combined_bias_mean = self.clip_and_stack(
    frames=self.inputFrames, recipe="soxs_mbias", ignore_input_masks=False,␣
↪post_stack_clipping=True)
```

`detrend`(*inputFrame*, *master_bias=False*, *dark=False*, *master_flat=False*, *order_table=False*)

subtract calibration frames from an input frame

Key Arguments:

- `inputFrame` – the input frame to have calibrations subtracted. CCDData object.
- `master_bias` – the master bias frame to be subtracted. CCDData object. Default *False*.
- `dark` – a dark frame to be subtracted. CCDData object. Default *False*.
- `master_flat` – divided input frame by this master flat frame. CCDData object. Default *False*.
- `order_table` – order table with order edges defined. Used to subtract scattered light background from frames. Default *False*.

Return:

- `calibration_subtracted_frame` – the input frame with the calibration frame(s) subtracted. CCDData object.

Usage:

Within a soxspipe recipe use `detrend` like so:

```
myCalibratedFrame = self.detrend(
    inputFrame=inputFrameCCDObject, master_bias=masterBiasCCDObject,␣
↪dark=darkCCDObject)
```

Todo:

- code needs written to scale dark frame to exposure time of science/calibration frame

get_recipe_settings()

get the recipe and arm specific settings

Return:

- `recipeSettings` – the recipe specific settings

Usage:

```
usage code
```

prepare_frames(*save=False*)

prepare raw frames by converting pixel data from ADU to electrons and adding mask and uncertainty extensions

Key Arguments:

- `save` – save out the prepared frame to the intermediate products directory. Default False.

Return:

- `preframes` – the new image collection containing the prepared frames

Usage

Usually called within a recipe class once the input frames have been selected and verified (see `soxs_mbias` code for example):

```
self.inputFrames = self.prepare_frames(
    save=self.settings["save-intermediate-products"])
```

produce_product()

generate the order-table with polynomal fits of order-centres

Return:

- `productPath` – the path to the order-table

qc_median_flux_level(*frame, frameType='MBIAS', frameName='master bias', medianFlux=False*)

calculate the median flux level in the frame, excluding masked pixels

Key Arguments:

- `frame` – the frame (CCDData object) to determine the median level.

- `frameType` – the type of the frame for reporting QC values Default "MBIAS"
- `frameName` – the name of the frame in human readable words. Default "master bias"
- `medianFlux` – if serendipitously calculated elsewhere don't recalculate. Default *False*

Return:

- `medianFlux` – median flux level in electrons

Usage:

```
medianFlux = self.qc_median_flux_level(
    frame=myFrame,
    frameType="MBIAS",
    frameName="master bias")
```

qc_ron (*frameType=False, frameName=False, masterFrame=False, rawRon=False, masterRon=False*)
calculate the read-out-noise from bias/dark frames

Key Arguments:

- `frameType` – the type of the frame for reporting QC values. Default *False*
- `frameName` – the name of the frame in human readable words. Default *False*
- `masterFrame` – the master frame (only makes sense to measure RON on master bias). Default *False*
- `rawRon` – if serendipitously calculated elsewhere don't recalculate. Default *False*
- `masterRon` – if serendipitously calculated elsewhere don't recalculate. Default *False*

Return:

- `rawRon` – raw read-out-noise in electrons
- `masterRon` – combined read-out-noise in mbias

Usage:

```
rawRon, mbiasRon = self.qc_ron(
    frameType="MBIAS",
    frameName="master bias",
    masterFrame=masterFrame
)
```

report_output (*rformat='stdout'*)
a method to report QC values alongside intermediate and final products

Key Arguments:

- `rformat` – the format to outout reports as. Default *stdout*. [stdout|. . .]

Usage:

```
self.report_output(rformat="stdout")
```

subtract_mean_flux_level (*rawFrame*)
iteratively median sigma-clip raw bias data frames before calculating and removing the mean bias level

Key Arguments:

- `rawFrame` – the raw bias frame

Return:

```
- `meanFluxLevel` -- the frame mean bias level
- `fluxStd` -- the standard deviation of the flux distribution (RON)
- `noiseFrame` -- the raw bias frame with mean bias level removed
```

Usage:

```
meanFluxLevel, fluxStd, noiseFrame = self.subtract_mean_flux_level(rawFrame)
```

update_fits_keywords (*frame*)

update fits keywords to comply with ESO Phase 3 standards

Key Arguments:

- `frame` – the frame to update

Return:

```
- None
```

Usage:

```
usage code
```

Todo:

- add usage info
- create a sublime snippet for usage
- write a command-line tool for this method
- update package tutorial with command-line tool info if needed

verify_input_frames ()

verify input frames match those required by the soxs_order_centres recipe

Return:

```
- ``None``
```

If the fits files conform to required input for the recipe everything will pass silently, otherwise an exception shall be raised.

xsh2soxs (*frame*)

perform some massaging of the xshooter data so it more closely resembles soxs data - this function can be removed once code is production ready

Key Arguments:

- `frame` – the CCDDate frame to manipulate

Return:

- `frame` – the manipulated soxspipe-ready frame

Usage:

```
frame = self.xsh2soxs(frame)
```

2.11.22 soxs_spatial_solution *(class)*

class soxs_spatial_solution(*log*, *settings=False*, *inputFrames=[]*, *verbose=False*, *overwrite=False*, *create2DMap=True*, *polyOrders=False*)

Bases: `soxspipe.recipes._base_recipe_._base_recipe_`

The soxs_spatial_solution recipe

Key Arguments

- `log` – logger
- `settings` – the settings dictionary
- `inputFrames` – input fits frames. Can be a directory, a set-of-files (SOF) file or a list of fits frame paths
- `verbose` – verbose. True or False. Default *False*
- `overwrite` – overwrite the prodcut file if it already exists. Default *False*
- `create2DMap` – create the 2D image map of wavelength, slit-position and order from disp solution.
- `polyOrders` – the orders of the x-y polynomials used to fit the dispersion solution. Overrides parameters found in the yaml settings file. e.g 345435 is order_x=3, order_y=4 ,wavelength_x=5 ,wavelength_y=4, slit_x=3 ,slit_y=5. Default *False*.

See `produce_product` method for usage.

Todo:

- add a tutorial about `soxs_spatial_solution` to documentation

Methods

`clean_up`()	*update product status in DB and remove intermediate files once recipe is complete*
`clip_and_stack`(frames, recipe[, ...])	*mean combine input frames after sigma-clipping outlying pixels using a median value with median absolute deviation (mad) as the deviation function*
`detrend`(inputFrame[, master_bias, dark, ...])	*subtract calibration frames from an input frame*
`get_recipe_settings`()	*get the recipe and arm specific settings*
`prepare_frames`([save])	*prepare raw frames by converting pixel data from ADU to electrons and adding mask and uncertainty extensions*
`produce_product`()	*generate the 2D dispersion map*
`qc_median_flux_level`(frame[, frameType, ...])	*calculate the median flux level in the frame, excluding masked pixels*
`qc_ron`([frameType, frameName, masterFrame, ...])	*calculate the read-out-noise from bias/dark frames*
`report_output`([rformat])	*a method to report QC values alongside intermediate and final products*

continues on next page

Table 31 – continued from previous page

`subtract_mean_flux_level`(rawFrame)	*iteratively median sigma-clip raw bias data frames before calculating and removing the mean bias level*
`update_fits_keywords`(frame)	*update fits keywords to comply with ESO Phase 3 standards*
`verify_input_frames`()	*verify input frames match those required by the ``soxs_spatial_solution`` recipe*
`xsh2soxs`(frame)	*perform some massaging of the xshooter data so it more closely resembles soxs data - this function can be removed once code is production ready*

clean_up()

update product status in DB and remove intermediate files once recipe is complete

Usage

```
recipe.clean_up()
```

clip_and_stack(*frames*, *recipe*, *ignore_input_masks=False*, *post_stack_clipping=True*)

mean combine input frames after sigma-clipping outlying pixels using a median value with median absolute deviation (mad) as the deviation function

Key Arguments:

- `frames` – an ImageFileCollection of the frames to stack or a list of CCDData objects
- `recipe` – the name of recipe needed to read the correct settings from the yaml files
- `ignore_input_masks` – ignore the input masks during clip and stacking?
- `post_stack_clipping` – allow cross-plane clipping on combined frame. Clipping settings in setting file. Default *True*.

Return:

- `combined_frame` – the combined master frame (with updated bad-pixel and uncertainty maps)

Usage:

This snippet can be used within the recipe code to combine individual (using bias frames as an example):

```
combined_bias_mean = self.clip_and_stack(
    frames=self.inputFrames, recipe="soxs_mbias", ignore_input_masks=False,␣
↪post_stack_clipping=True)
```

detrend(*inputFrame*, *master_bias=False*, *dark=False*, *master_flat=False*, *order_table=False*)

subtract calibration frames from an input frame

Key Arguments:

- `inputFrame` – the input frame to have calibrations subtracted. CCDData object.
- `master_bias` – the master bias frame to be subtracted. CCDData object. Default *False*.
- `dark` – a dark frame to be subtracted. CCDData object. Default *False*.
- `master_flat` – divided input frame by this master flat frame. CCDData object. Default *False*.
- `order_table` – order table with order edges defined. Used to subtract scattered light background from frames. Default *False*.

Return:

- `calibration_subtracted_frame` – the input frame with the calibration frame(s) subtracted. CCDData object.

Usage:

Within a soxspipe recipe use `detrend` like so:

```
myCalibratedFrame = self.detrend(
    inputFrame=inputFrameCCDObject, master_bias=masterBiasCCDObject,␣
↪dark=darkCCDObject)
```

Todo:

- code needs written to scale dark frame to exposure time of science/calibration frame

get_recipe_settings()

get the recipe and arm specific settings

Return:

- `recipeSettings` – the recipe specific settings

Usage:

```
usage code
```

prepare_frames(*save=False*)

prepare raw frames by converting pixel data from ADU to electrons and adding mask and uncertainty extensions

Key Arguments:

- `save` – save out the prepared frame to the intermediate products directory. Default False.

Return:

- `preframes` – the new image collection containing the prepared frames

Usage

Usually called within a recipe class once the input frames have been selected and verified (see `soxs_mbias` code for example):

```
self.inputFrames = self.prepare_frames(
    save=self.settings["save-intermediate-products"])
```

produce_product()

generate the 2D dispersion map

Return:

- `productPath` – the path to the 2D dispersion map

Usage

```
from soxspipe.recipes import soxs_spatial_solution
recipe = soxs_spatial_solution(
    log=log,
    settings=settings,
    inputFrames=fileList
```

(continues on next page)

(continued from previous page)

```
)
disp_map = recipe.produce_product()
```

qc_median_flux_level(*frame*, *frameType='MBIAS'*, *frameName='master bias'*, *medianFlux=False*)
calculate the median flux level in the frame, excluding masked pixels

Key Arguments:

- `frame` – the frame (CCDData object) to determine the median level.
- `frameType` – the type of the frame for reporting QC values Default "MBIAS"
- `frameName` – the name of the frame in human readable words. Default "master bias"
- `medianFlux` – if serendipitously calculated elsewhere don't recalculate. Default *False*

Return:

- `medianFlux` – median flux level in electrons

Usage:

```
medianFlux = self.qc_median_flux_level(
    frame=myFrame,
    frameType="MBIAS",
    frameName="master bias")
```

qc_ron(*frameType=False*, *frameName=False*, *masterFrame=False*, *rawRon=False*, *masterRon=False*)
calculate the read-out-noise from bias/dark frames

Key Arguments:

- `frameType` – the type of the frame for reporting QC values. Default *False*
- `frameName` – the name of the frame in human readable words. Default *False*
- `masterFrame` – the master frame (only makes sense to measure RON on master bias). Default *False*
- `rawRon` – if serendipitously calculated elsewhere don't recalculate. Default *False*
- `masterRon` – if serendipitously calculated elsewhere don't recalculate. Default *False*

Return:

- `rawRon` – raw read-out-noise in electrons
- `masterRon` – combined read-out-noise in mbias

Usage:

```
rawRon, mbiasRon = self.qc_ron(
    frameType="MBIAS",
    frameName="master bias",
    masterFrame=masterFrame
)
```

report_output(*rformat='stdout'*)
a method to report QC values alongside intermediate and final products

Key Arguments:

- `rformat` – the format to outout reports as. Default *stdout*. [stdout|. . .]

Usage:

```
self.report_output(rformat="stdout")
```

`subtract_mean_flux_level`(*rawFrame*)

iteratively median sigma-clip raw bias data frames before calculating and removing the mean bias level

Key Arguments:

- `rawFrame` – the raw bias frame

Return:

```
- `meanFluxLevel` -- the frame mean bias level
- `fluxStd` -- the standard deviation of the flux distribution (RON)
- `noiseFrame` -- the raw bias frame with mean bias level removed
```

Usage:

```
meanFluxLevel, fluxStd, noiseFrame = self.subtract_mean_flux_level(rawFrame)
```

`update_fits_keywords`(*frame*)

update fits keywords to comply with ESO Phase 3 standards

Key Arguments:

- `frame` – the frame to update

Return:

```
- None
```

Usage:

```
usage code
```

Todo:

- add usage info
- create a sublime snippet for usage
- write a command-line tool for this method
- update package tutorial with command-line tool info if needed

`verify_input_frames`()

verify input frames match those required by the ``soxs_spatial_solution`` recipe

If the fits files conform to required input for the recipe everything will pass silently, otherwise an exception shall be raised.

`xsh2soxs`(*frame*)

perform some massaging of the xshooter data so it more closely resembles soxs data - this function can be removed once code is production ready

Key Arguments:

- `frame` – the CCDDate frame to manipulate

Return:

- `frame` – the manipulated soxspipe-ready frame

Usage:

```
frame = self.xsh2soxs(frame)
```

2.11.23 soxs_stare *(class)*

class soxs_stare(*log*, *settings=False*, *inputFrames=[]*, *verbose=False*, *overwrite=False*)

Bases: `soxspipe.recipes._base_recipe_._base_recipe_`

The soxs_stare recipe

Key Arguments

- `log` – logger
- `settings` – the settings dictionary
- `inputFrames` – input fits frames. Can be a directory, a set-of-files (SOF) file or a list of fits frame paths.
- `verbose` – verbose. True or False. Default *False*
- `overwrite` – overwrite the product file if it already exists. Default *False*

See `produce_product` method for usage.

Todo:

- add usage info
- create a sublime snippet for usage
- create cl-util for this class
- add a tutorial about `soxs_stare` to documentation

Methods

`clean_up`()	*update product status in DB and remove intermediate files once recipe is complete*
`clip_and_stack`(frames, recipe[, ...])	*mean combine input frames after sigma-clipping outlying pixels using a median value with median absolute deviation (mad) as the deviation function*
`detrend`(inputFrame[, master_bias, dark, ...])	*subtract calibration frames from an input frame*
`get_recipe_settings`()	*get the recipe and arm specific settings*
`prepare_frames`([save])	*prepare raw frames by converting pixel data from ADU to electrons and adding mask and uncertainty extensions*
`produce_product`()	*The code to generate the product of the soxs_stare recipe*
`qc_median_flux_level`(frame[, frameType, ...])	*calculate the median flux level in the frame, excluding masked pixels*

continues on next page

Table 32 – continued from previous page

`qc_ron`([frameType, frameName, masterFrame, ...])	*calculate the read-out-noise from bias/dark frames*
`report_output`([rformat])	*a method to report QC values alongside intermediate and final products*
`subtract_mean_flux_level`(rawFrame)	*iteratively median sigma-clip raw bias data frames before calculating and removing the mean bias level*
`update_fits_keywords`(frame)	*update fits keywords to comply with ESO Phase 3 standards*
`verify_input_frames`()	*verify the input frame match those required by the soxs_stare recipe*
`xsh2soxs`(frame)	*perform some massaging of the xshooter data so it more closely resembles soxs data - this function can be removed once code is production ready*

clean_up()

update product status in DB and remove intermediate files once recipe is complete

Usage

```
recipe.clean_up()
```

clip_and_stack(*frames*, *recipe*, *ignore_input_masks=False*, *post_stack_clipping=True*)

mean combine input frames after sigma-clipping outlying pixels using a median value with median absolute deviation (mad) as the deviation function

Key Arguments:

- `frames` – an ImageFileCollection of the frames to stack or a list of CCDData objects
- `recipe` – the name of recipe needed to read the correct settings from the yaml files
- `ignore_input_masks` – ignore the input masks during clip and stacking?
- `post_stack_clipping` – allow cross-plane clipping on combined frame. Clipping settings in setting file. Default *True*.

Return:

- `combined_frame` – the combined master frame (with updated bad-pixel and uncertainty maps)

Usage:

This snippet can be used within the recipe code to combine individual (using bias frames as an example):

```
combined_bias_mean = self.clip_and_stack(
    frames=self.inputFrames, recipe="soxs_mbias", ignore_input_masks=False,␣
↪post_stack_clipping=True)
```

detrend(*inputFrame*, *master_bias=False*, *dark=False*, *master_flat=False*, *order_table=False*)

subtract calibration frames from an input frame

Key Arguments:

- `inputFrame` – the input frame to have calibrations subtracted. CCDData object.
- `master_bias` – the master bias frame to be subtracted. CCDData object. Default *False*.
- `dark` – a dark frame to be subtracted. CCDData object. Default *False*.
- `master_flat` – divided input frame by this master flat frame. CCDData object. Default *False*.

- `order_table` – order table with order edges defined. Used to subtract scattered light background from frames. Default *False*.

Return:

- `calibration_subtracted_frame` – the input frame with the calibration frame(s) subtracted. CCDData object.

Usage:

Within a soxspipe recipe use `detrend` like so:

```
myCalibratedFrame = self.detrend(
    inputFrame=inputFrameCCDObject, master_bias=masterBiasCCDObject,␣
↪dark=darkCCDObject)
```

Todo:

- code needs written to scale dark frame to exposure time of science/calibration frame

get_recipe_settings()

get the recipe and arm specific settings

Return:

- `recipeSettings` – the recipe specific settings

Usage:

```
usage code
```

prepare_frames(*save=False*)

prepare raw frames by converting pixel data from ADU to electrons and adding mask and uncertainty extensions

Key Arguments:

- `save` – save out the prepared frame to the intermediate products directory. Default False.

Return:

- `preframes` – the new image collection containing the prepared frames

Usage

Usually called within a recipe class once the input frames have been selected and verified (see `soxs_mbias` code for example):

```
self.inputFrames = self.prepare_frames(
    save=self.settings["save-intermediate-products"])
```

produce_product()

The code to generate the product of the soxs_stare recipe

Return:

- `productPath` – the path to the final product

Usage

```
from soxspipe.recipes import soxs_stare
recipe = soxs_stare(
    log=log,
    settings=settings,
    inputFrames=fileList
)
stareFrame = recipe.produce_product()
```

qc_median_flux_level(*frame, frameType='MBIAS', frameName='master bias', medianFlux=False*)
calculate the median flux level in the frame, excluding masked pixels

Key Arguments:

- `frame` – the frame (CCDData object) to determine the median level.
- `frameType` – the type of the frame for reporting QC values Default "MBIAS"
- `frameName` – the name of the frame in human readable words. Default "master bias"
- `medianFlux` – if serendipitously calculated elsewhere don't recalculate. Default *False*

Return:

- `medianFlux` – median flux level in electrons

Usage:

```
medianFlux = self.qc_median_flux_level(
    frame=myFrame,
    frameType="MBIAS",
    frameName="master bias")
```

qc_ron(*frameType=False, frameName=False, masterFrame=False, rawRon=False, masterRon=False*)
calculate the read-out-noise from bias/dark frames

Key Arguments:

- `frameType` – the type of the frame for reporting QC values. Default *False*
- `frameName` – the name of the frame in human readable words. Default *False*
- `masterFrame` – the master frame (only makes sense to measure RON on master bias). Default *False*
- `rawRon` – if serendipitously calculated elsewhere don't recalculate. Default *False*
- `masterRon` – if serendipitously calculated elsewhere don't recalculate. Default *False*

Return:

- `rawRon` – raw read-out-noise in electrons
- `masterRon` – combined read-out-noise in mbias

Usage:

```
rawRon, mbiasRon = self.qc_ron(
    frameType="MBIAS",
    frameName="master bias",
    masterFrame=masterFrame
)
```

report_output (*rformat='stdout'*)
a method to report QC values alongside intermediate and final products

Key Arguments:

- `rformat` – the format to outout reports as. Default *stdout*. [stdout|. . .]

Usage:

```
self.report_output(rformat="stdout")
```

subtract_mean_flux_level (*rawFrame*)
iteratively median sigma-clip raw bias data frames before calculating and removing the mean bias level

Key Arguments:

- `rawFrame` – the raw bias frame

Return:

```
- `meanFluxLevel` -- the frame mean bias level
- `fluxStd` -- the standard deviation of the flux distribution (RON)
- `noiseFrame` -- the raw bias frame with mean bias level removed
```

Usage:

```
meanFluxLevel, fluxStd, noiseFrame = self.subtract_mean_flux_level(rawFrame)
```

update_fits_keywords (*frame*)
update fits keywords to comply with ESO Phase 3 standards

Key Arguments:

- `frame` – the frame to update

Return:

```
- None
```

Usage:

```
usage code
```

Todo:

- add usage info
- create a sublime snippet for usage
- write a command-line tool for this method
- update package tutorial with command-line tool info if needed

verify_input_frames ()
verify the input frame match those required by the soxs_stare recipe

Return:

```
- ``None``
```

If the fits files conform to required input for the recipe everything will pass silently, otherwise an exception shall be raised.

xsh2soxs (*frame*)

perform some massaging of the xshooter data so it more closely resembles soxs data - this function can be removed once code is production ready

Key Arguments:

- `frame` – the CCDDate frame to manipulate

Return:

- `frame` – the manipulated soxspipe-ready frame

Usage:

```
frame = self.xsh2soxs(frame)
```

2.11.24 soxs_straighten *(class)*

class soxs_straighten (*log*, *settings=False*, *inputFrames=[]*, *verbose=False*, *overwrite=False*)

Bases: `soxspipe.recipes._base_recipe_._base_recipe_`

The soxs_straighten recipe

Methods

`clean_up`()	*update product status in DB and remove intermediate files once recipe is complete*
`clip_and_stack`(frames, recipe[, ...])	*mean combine input frames after sigma-clipping outlying pixels using a median value with median absolute deviation (mad) as the deviation function*
`detrend`(inputFrame[, master_bias, dark, ...])	*subtract calibration frames from an input frame*
`get_recipe_settings`()	*get the recipe and arm specific settings*
`prepare_frames`([save])	*prepare raw frames by converting pixel data from ADU to electrons and adding mask and uncertainty extensions*
`produce_product`()	*The code to generate the product of the soxs_straighten recipe*
`qc_median_flux_level`(frame[, frameType, ...])	*calculate the median flux level in the frame, excluding masked pixels*
`qc_ron`([frameType, frameName, masterFrame, ...])	*calculate the read-out-noise from bias/dark frames*
`report_output`([rformat])	*a method to report QC values alongside intermediate and final products*
`subtract_mean_flux_level`(rawFrame)	*iteratively median sigma-clip raw bias data frames before calculating and removing the mean bias level*
`update_fits_keywords`(frame)	*update fits keywords to comply with ESO Phase 3 standards*
`verify_input_frames`()	*verify the input frame match those required by the soxs_straighten recipe*

continues on next page

Table 33 – continued from previous page

`xsh2soxs`(frame)	*perform some massaging of the xshooter data so it more closely resembles soxs data - this function can be removed once code is production ready*

clean_up()

update product status in DB and remove intermediate files once recipe is complete

Usage

```
recipe.clean_up()
```

clip_and_stack(*frames*, *recipe*, *ignore_input_masks=False*, *post_stack_clipping=True*)

mean combine input frames after sigma-clipping outlying pixels using a median value with median absolute deviation (mad) as the deviation function

Key Arguments:

- `frames` – an ImageFileCollection of the frames to stack or a list of CCDData objects
- `recipe` – the name of recipe needed to read the correct settings from the yaml files
- `ignore_input_masks` – ignore the input masks during clip and stacking?
- `post_stack_clipping` – allow cross-plane clipping on combined frame. Clipping settings in setting file. Default *True*.

Return:

- `combined_frame` – the combined master frame (with updated bad-pixel and uncertainty maps)

Usage:

This snippet can be used within the recipe code to combine individual (using bias frames as an example):

```
combined_bias_mean = self.clip_and_stack(
    frames=self.inputFrames, recipe="soxs_mbias", ignore_input_masks=False,␣
↪post_stack_clipping=True)
```

detrend(*inputFrame*, *master_bias=False*, *dark=False*, *master_flat=False*, *order_table=False*)

subtract calibration frames from an input frame

Key Arguments:

- `inputFrame` – the input frame to have calibrations subtracted. CCDData object.
- `master_bias` – the master bias frame to be subtracted. CCDData object. Default *False*.
- `dark` – a dark frame to be subtracted. CCDData object. Default *False*.
- `master_flat` – divided input frame by this master flat frame. CCDData object. Default *False*.
- `order_table` – order table with order edges defined. Used to subtract scattered light background from frames. Default *False*.

Return:

- `calibration_subtracted_frame` – the input frame with the calibration frame(s) subtracted. CCDData object.

Usage:

Within a soxspipe recipe use `detrend` like so:

```
myCalibratedFrame = self.detrend(
    inputFrame=inputFrameCCDObject, master_bias=masterBiasCCDObject,␣
↪dark=darkCCDObject)
```

Todo:

- code needs written to scale dark frame to exposure time of science/calibration frame

get_recipe_settings()

get the recipe and arm specific settings

Return:

- `recipeSettings` – the recipe specific settings

Usage:

```
usage code
```

prepare_frames(*save=False*)

prepare raw frames by converting pixel data from ADU to electrons and adding mask and uncertainty extensions

Key Arguments:

- `save` – save out the prepared frame to the intermediate products directory. Default False.

Return:

- `preframes` – the new image collection containing the prepared frames

Usage

Usually called within a recipe class once the input frames have been selected and verified (see `soxs_mbias` code for example):

```
self.inputFrames = self.prepare_frames(
    save=self.settings["save-intermediate-products"])
```

produce_product()

The code to generate the product of the soxs_straighten recipe

Return:

- `productPath` – the path to the final product

Usage

```
from soxspipe.recipes import soxs_straighten
recipe = soxs_straighten(
    log=log,
    settings=settings,
    inputFrames=fileList
)
straightenFrame = recipe.produce_product()
```

qc_median_flux_level(*frame*, *frameType='MBIAS'*, *frameName='master bias'*, *medianFlux=False*)

calculate the median flux level in the frame, excluding masked pixels

Key Arguments:

- `frame` – the frame (CCDData object) to determine the median level.
- `frameType` – the type of the frame for reporting QC values Default "MBIAS"
- `frameName` – the name of the frame in human readable words. Default "master bias"
- `medianFlux` – if serendipitously calculated elsewhere don't recalculate. Default *False*

Return:

- `medianFlux` – median flux level in electrons

Usage:

```
medianFlux = self.qc_median_flux_level(
    frame=myFrame,
    frameType="MBIAS",
    frameName="master bias")
```

qc_ron (*frameType=False*, *frameName=False*, *masterFrame=False*, *rawRon=False*, *masterRon=False*)
calculate the read-out-noise from bias/dark frames

Key Arguments:

- `frameType` – the type of the frame for reporting QC values. Default *False*
- `frameName` – the name of the frame in human readable words. Default *False*
- `masterFrame` – the master frame (only makes sense to measure RON on master bias). Default *False*
- `rawRon` – if serendipitously calculated elsewhere don't recalculate. Default *False*
- `masterRon` – if serendipitously calculated elsewhere don't recalculate. Default *False*

Return:

- `rawRon` – raw read-out-noise in electrons
- `masterRon` – combined read-out-noise in mbias

Usage:

```
rawRon, mbiasRon = self.qc_ron(
    frameType="MBIAS",
    frameName="master bias",
    masterFrame=masterFrame
)
```

report_output (*rformat='stdout'*)
a method to report QC values alongside intermediate and final products

Key Arguments:

- `rformat` – the format to outout reports as. Default *stdout*. [stdout|. . .]

Usage:

```
self.report_output(rformat="stdout")
```

subtract_mean_flux_level (*rawFrame*)
iteratively median sigma-clip raw bias data frames before calculating and removing the mean bias level

Key Arguments:

- `rawFrame` – the raw bias frame

Return:

```
- `meanFluxLevel` -- the frame mean bias level
- `fluxStd` -- the standard deviation of the flux distribution (RON)
- `noiseFrame` -- the raw bias frame with mean bias level removed
```

Usage:

```
meanFluxLevel, fluxStd, noiseFrame = self.subtract_mean_flux_level(rawFrame)
```

update_fits_keywords(*frame*)

update fits keywords to comply with ESO Phase 3 standards

Key Arguments:

- `frame` – the frame to update

Return:

```
- None
```

Usage:

```
usage code
```

Todo:

- add usage info
- create a sublime snippet for usage
- write a command-line tool for this method
- update package tutorial with command-line tool info if needed

verify_input_frames()

verify the input frame match those required by the soxs_straighten recipe

Return:

```
- ``None``
```

If the fits files conform to required input for the recipe everything will pass silently, otherwise an exception shall be raised.

xsh2soxs(*frame*)

perform some massaging of the xshooter data so it more closely resembles soxs data - this function can be removed once code is production ready

Key Arguments:

- `frame` – the CCDDate frame to manipulate

Return:

- `frame` – the manipulated soxspipe-ready frame

Usage:

```
frame = self.xsh2soxs(frame)
```

2.12 Functions

`soxspipe.commonutils.dispersion_map_to_pixel_arrays`	*use a first-guess dispersion map to append x,y fits to line-list data frame.*
`soxspipe.commonutils.filenamer`	Given a FITS object, use the SOXS file-naming scheme to return a filename to be used to save the FITS object to disk
`soxspipe.commonutils.getpackagepath`	*Get the root path for this python package*
`soxspipe.commonutils.toolkit.add_recipe_logger`	*add a recipe-specific handler to the default logger that writes the recipe's logs adjacent to the recipe project*
`soxspipe.commonutils.toolkit.create_dispersion_solution_grid_lines_for_plot`	*give a dispersion solution and accompanying 2D dispersion map image, generate the grid lines to add to QC plots*
`soxspipe.commonutils.toolkit.cut_image_slice`	*cut and return an N-pixel wide and M-pixels long slice, centred on a given coordinate from an image frame*
`soxspipe.commonutils.toolkit.generic_quality_checks`	*measure very basic quality checks on a frame and return the QC table with results appended*
`soxspipe.commonutils.toolkit.get_calibration_lamp`	*given a frame, determine which calibration lamp is being used*
`soxspipe.commonutils.toolkit.get_calibrations_path`	*return the root path to the static calibrations*
`soxspipe.commonutils.toolkit.predict_product_path`	*predict the path of the recipe product from a given SOF name*
`soxspipe.commonutils.toolkit.quicklook_image`	*generate a quicklook image of a CCDObject - useful for development/debugging*
`soxspipe.commonutils.toolkit.read_spectral_format`	*read the spectral format table to get some key parameters*
`soxspipe.commonutils.toolkit.spectroscopic_image_quality_checks`	*measure and record spectroscopic image quailty checks*
`soxspipe.commonutils.toolkit.twoD_disp_map_image_to_dataframe`	*convert the 2D dispersion image map to a pandas dataframe*
`soxspipe.commonutils.toolkit.unpack_order_table`	*unpack an order table and return a top-level `orderPolyTable` data-frame and a second `orderPixelTable` data-frame with the central-trace coordinates of each order given
`soxspipe.commonutils.uncompress`	uncompress ESO fits.Z frames

2.12.1 dispersion_map_to_pixel_arrays (*function*)

dispersion_map_to_pixel_arrays (*log*, *dispersionMapPath*, *orderPixelTable*, *removeOffDetectorLocation=True*)

use a first-guess dispersion map to append x,y fits to line-list data frame.

Return a line-list with x,y fits given a first guess dispersion map.*

Key Arguments:

- `log` – logger
- `dispersionMapPath` – path to the dispersion map
- `orderPixelTable` – a data-frame including 'order', 'wavelength' and 'slit_pos' columns
- `removeOffDetectorLocation` – if data points are found to lie off the detector plane then remove them from the resutls. Default *True*

Usage:

```
from soxspipe.commonutils import dispersion_map_to_pixel_arrays
myDict = {
    "order": [11, 11, 11],
    "wavelength": [850.3, 894.3, 983.2],
    "slit_position": [0, 0, 0]
}
orderPixelTable = pd.DataFrame(myDict)
orderPixelTable = dispersion_map_to_pixel_arrays(
    log=log,
    dispersionMapPath="/path/to/map.csv",
    orderPixelTable=orderPixelTable
)
```

2.12.2 filenamer (*function*)

filenamer (*log*, *frame*, *keywordLookup=False*, *detectorLookup=False*, *settings=False*)

Given a FITS object, use the SOXS file-naming scheme to return a filename to be used to save the FITS object to disk

Key Arguments:

- `log` – logger
- `frame` – the CCDData object frame
- `keywordLookup` – the keyword lookup dictionary (needed if `settings` not provided). Default *False*
- `detectorLookup` – the detector parameters (needed if `settings` not provided). Default *False*
- `settings` – the soxspipe settings dictionary (needed if `keywordLookup` and `detectorLookup` not provided). Default *False*

Return:

- `filename` – stanardised name to for the input frame

```
frame = CCDData.read(filepath, hdu=0, unit=u.electron, hdu_uncertainty='ERRS',
        du_mask='QUAL', hdu_flags='FLAGS', key_uncertainty_type='UTYPE')
```

(continues on next page)

(continued from previous page)

```
from soxspipe.commonutils import filenamer
filename = filenamer(
    log=log,
    frame=frame,
    settings=settings
)
```

2.12.3 getpackagepath (*function*)

getpackagepath()

Get the root path for this python package

Used in unit testing code

2.12.4 add_recipe_logger (*function*)

add_recipe_logger(*log*, *productPath*)

add a recipe-specific handler to the default logger that writes the recipe's logs adjacent to the recipe project

2.12.5 create_dispersion_solution_grid_lines_for_plot (*function*)

create_dispersion_solution_grid_lines_for_plot(*log*, *dispMap*, *dispMapImage*, *associatedFrame*, *kw*, *skylines=False*, *slitPositions=False*)

give a dispersion solution and accompanying 2D dispersion map image, generate the grid lines to add to QC plots

Key Arguments:

- `log` – logger
- `dispMap` – path to dispersion map. Default *False*
- `dispMapImage` – the 2D dispersion map image
- `associatedFrame` – a frame associated with the reduction (to read arm and binning info).
- `kw` – fits header kw dictionary
- `skylines` – a list of skylines to use as the grid. Default *False*
- `slitPositions` – slit positions to plot (else plot min and max)

Usage:

```
from soxspipe.commonutils.toolkit import create_dispersion_solution_grid_lines_
↪for_plot
gridLinePixelTable = create_dispersion_solution_grid_lines_for_plot(
    log=log,
    dispMap=dispMap,
    dispMapImage=dispMapImage,
    associatedFrame=CCDObject,
    kw=kw,
    skylines=skylines
)
```

(continues on next page)

(continued from previous page)

```
for l in range(int(gridLinePixelTable['line'].max())):
    mask = (gridLinePixelTable['line'] == l)
    ax.plot(gridLinePixelTable.loc[mask]["fit_y"], gridLinePixelTable.loc[mask][
↪"fit_x"], "w-", linewidth=0.5, alpha=0.8, color="black")
```

2.12.6 cut_image_slice (*function*)

cut_image_slice(*log*, *frame*, *width*, *length*, *x*, *y*, *sliceAxis='x'*, *median=False*, *plot=False*)

cut and return an N-pixel wide and M-pixels long slice, centred on a given coordinate from an image frame

Key Arguments:

- `log` – logger
- `frame` – the data array to cut the slice from (masked array)
- `width` – width of the slice (odd number)
- `length` – length of the slice
- `x` – x-coordinate
- `y` – y-coordinate
- `sliceAxis` – the axis along which slice is to be taken. Default *x*
- `median` – collapse the slice to a median value across its width
- `plot` – generate a plot of slice. Useful for debugging.

Usage:

```
from soxspipe.commonutils.toolkit import cut_image_slice
slice = cut_image_slice(log=self.log, frame=self.pinholeFlat.data,
                                width=1, length=sliceLength, x=x_fit, y=y_fit,␣
↪plot=False)
if slice is None:
    return None
```

2.12.7 generic_quality_checks (*function*)

generic_quality_checks(*log*, *frame*, *settings*, *recipeName*, *qcTable*)

measure very basic quality checks on a frame and return the QC table with results appended

Key Arguments:

- `log` – logger
- `frame` – CCDData object
- `settings` – soxspipe settings
- `recipeName` – the name of the recipe
- `qcTable` – the QC pandas data-frame to save the QC measurements

Usage:

Todo: add usage info create a sublime snippet for usage

```
usage code
```

2.12.8 get_calibration_lamp (*function*)

get_calibration_lamp(*log*, *frame*, *kw*)

given a frame, determine which calibration lamp is being used

Key Arguments:

- `log` – logger
- `frame` – the frame to determine the calibration lamp for
- `kw` – the FITS header keyword dictionary

Usage:

```
from soxspipe.commonutils.toolkit import get_calibration_lamp
lamp = get_calibration_lamp(log=log, frame=frame, kw=kw)
```

2.12.9 get_calibrations_path (*function*)

get_calibrations_path(*log*, *settings*)

return the root path to the static calibrations

Key Arguments:

- `log` – logger
- `settings` – the settings dictionary

Usage:

```
from soxspipe.commonutils.toolkit import get_calibrations_path
calibrationRootPath = get_calibrations_path(log=log, settings=settings)
```

2.12.10 predict_product_path (*function*)

predict_product_path(*sofName*, *recipeName=False*)

predict the path of the recipe product from a given SOF name

Key Arguments:

- `log` – logger,
- `sofName` – name or full path to the sof file
- `recipeName` – name of the recipe being considered. Default *False*.

Usage:

```
from soxspipe.commonutils import toolkit
productPath = toolkit.predict_product_path(sofFilePath)
```

2.12.11 quicklook_image (*function*)

quicklook_image(*log*, *CCDObject*, *show=True*, *ext='data'*, *stdWindow=3*, *title=False*, *surfacePlot=False*, *dispMap=False*, *dispMapImage=False*, *inst=False*, *settings=False*, *skylines=False*, *saveToPath=False*)

generate a quicklook image of a CCDObject - useful for development/debugging

Key Arguments:

- `log` – logger
- `CCDObject` – the CCDObject to plot
- `show` – show the image. Set to False to skip
- `ext` – the name of the the extension to show. Can be "data", "mask" or "err". Default "data".
- `title` – give a title for the plot
- `surfacePlot` – plot as a 3D surface plot
- `dispMap` – path to dispersion map. Default *False*
- `dispMapImage` – the 2D dispersion map image
- `inst` – provide instrument name if no header exists
- `skylines` – mark skylines on image

```python
from soxspipe.commonutils.toolkit import quicklook_image
quicklook_image(
    log=self.log, CCDObject=myframe, show=True)
```

2.12.12 read_spectral_format (*function*)

read_spectral_format(*log*, *settings*, *arm*, *dispersionMap=False*, *extended=True*)

read the spectral format table to get some key parameters

Key Arguments:

- `log` – logger
- `settings` – soxspipe settings
- `arm` – arm to retrieve format for
- `dispersionMap` – if a dispersion map is given, the minimum and maximum dispersion axis pixel limits are computed
- `extended` – the spectral format table can provide WLMIN/WLMAX (extended=False) or WLMINFUL/WLMAXFUL (extended=True)

Return:

- `orderNums` – a list of the order numbers
- `waveLengthMin` – a list of the maximum wavelengths reached by each order
- `waveLengthMax` – a list of the minimum wavelengths reached by each order

Usage:

```
from soxspipe.commonutils.toolkit import read_spectral_format
# READ THE SPECTRAL FORMAT TABLE TO DETERMINE THE LIMITS OF THE TRACES
orderNums, waveLengthMin, waveLengthMax = read_spectral_format(
        log=self.log, settings=self.settings, arm=arm)
```

2.12.13 spectroscopic_image_quality_checks (*function*)

spectroscopic_image_quality_checks(*log*, *frame*, *orderTablePath*, *settings*, *recipeName*, *qcTable*)

measure and record spectroscopic image quailty checks

Key Arguments:

- `log` – logger
- `frame` – CCDData object
- `orderTablePath` – path to the order table
- `settings` – soxspipe settings
- `recipeName` – the name of the recipe
- `qcTable` – the QC pandas data-frame to save the QC measurements

Usage:

Todo: add usage info create a sublime snippet for usage

```
usage code
```

2.12.14 twoD_disp_map_image_to_dataframe (*function*)

twoD_disp_map_image_to_dataframe(*log*, *slit_length*, *twoDMapPath*, *kw=False*, *associatedFrame=False*, *removeMaskedPixels=False*)

convert the 2D dispersion image map to a pandas dataframe

Key Arguments:

- `log` – logger
- `twoDMapPath` – 2D dispersion map image path
- `kw` – fits keyword lookup dictionary
- `associatedFrame` – include a flux column in returned dataframe from a frame assosiated with the dispersion map. Default *False*
- `removeMaskedPixels` – remove the masked pixels from the assosicated image? Default *False*

Usage:

```
from soxspipe.commonutils.toolkit import twoD_disp_map_image_to_dataframe
mapDF = twoD_disp_map_image_to_dataframe(log=log, twoDMapPath=twoDMap,␣
↪associatedFrame=objectFrame, kw=kw)
```

2.12.15 unpack_order_table (*function*)

unpack_order_table (*log*, *orderTablePath*, *extend=0.0*, *pixelDelta=1*, *binx=1*, *biny=1*, *prebinned=False*, *order=False*, *limitToDetectorFormat=False*)

*unpack an order table and return a top-level `orderPolyTable` data-frame and a second `orderPixelTable` data-frame with the central-trace coordinates of each order given

Key Arguments:

- `orderTablePath` – path to the order table
- `extend` – fractional increase to the order area in the y-axis (needed for masking)
- `pixelDelta` – space between returned data points. Default *1* (sampled at every pixel)
- `binx` – binning in the x-axis (from FITS header). Default *1*
- `biny` – binning in the y-axis (from FITS header). Default *1*
- `prebinned` – was the order-table measured on a pre-binned frame (typically only for mflats). Default *False*
- `order` – unpack only a single order
- `limitToDetectorFormat` – limit the pixels return to those limited by the detector format static calibration table

Usage:

```
# UNPACK THE ORDER TABLE
from soxspipe.commonutils.toolkit import unpack_order_table
orderPolyTable, orderPixelTable = unpack_order_table(
    log=self.log, orderTablePath=orderTablePath, extend=0.)
```

2.12.16 uncompress (*function*)

uncompress (*log*, *directory*)

uncompress ESO fits.Z frames

Key Arguments:

- `log` – logger
- `directory` – directory containing .Z file to uncompress

```
from soxspipe.commonutils import uncompress
uncompress(
    log=log,
    directory="/path/to/raw_data/"
)
```

2.13 A-Z Index

CHAPTER THREE

ACKNOWLEDGEMENTS

- SOXSPIPE makes use of `ccdproc`, an Astropy package for image reduction (Craig et al. 2021).

PYTHON MODULE INDEX

INDEX