PyKOALA Documentation

Ángel López-Sánchez, Jamila Scammell, Diana Dalae, Barr Perez, Pablo Corcho-Caballero & Yago Ascasíbar

Wednesday 26th January, 2022

THIS DOCUMENT STILL NEEDS A LOT OF UPDATES included on other Overleaf documents

1 INTRODUCTION

PyKOALA is a Python package to reduce KOALA+AAOmega integral field spectroscopy (IFS) data creating a data cube. It produces full calibrated (wavelength, flux and astrometry) data cubes ready for science.

2 PREPARING RSS FILES WITH 2DFDR

First we need to prepare the RAW data obtained with AAOmega to Raw Stacked Spectra (RSS) files. For that we use AAO standard software `2dFdr`. Running `2dFdr` will perform:

1. the dark frame subtraction,
2. the long-slit flat correction,
3. the cosmic rays removal,
4. the tram extraction (the `OPTEX` method is strongly recommended),
5. the wavelength calibration.

Full details will be available here eventually ...

3 HOW DOES PYKOALA WORK?

3.1 Classes: `RSS` and `interpolated_cube`

PyKOALA works with two well-defined Python classes (see Fig. 1):

1. **RSS** class, that reads and processes RSS files (throughput correction, small wavelength correction, correction for extinction, telluric correction, critical sky subtraction) to create a "clean" RSS file ready to be cubed,
2. **Interpolated_cube** class, that creates cubes from RSS files, reads cubes from external `.fits` files, and performs the alignment of the cubes, correction for atmospheric differential refraction, and applies the flux calibration. Extra tools for creating and plotting maps with the integrated emission in a particular range or emission line, emission line ratios, kinematics, velocity dispersions, and more are also included.

Hence, there are specific tasks that are only available for `RSS` or `Interpolated_cube`. These tasks are added as `.task` to the object, e.g., if `Hiltner600_rss1` is an `RSS` object in PyKOALA, the task `apply_throughput_2D` can be called using `Hiltner600_rss1.apply_throughput_2D()`. All the tasks associated to classes are only useful there, e.g., the telluric correction or the sky subtraction must be done *before* doing the cubing (i.e. when "cleaning" the `RSS` files), while the correction for atmospheric differential refraction or the alignment of each file can be only done once the cube is built.

There are many tasks that are not associated to a specific Python object and can be used with both a `RSS` or an `Interpolated_cube` object, for example `plot_plot()` for fast plotting or `fluxes` for getting the Gaussian and integrated flux of a emission or absorption line. Just be sure you know what you're providing as input!

Additionally, the class `KOALA_reduce` works combining both the `RSS` and `Interpolated_cube` classes. This tasks allows to reduce a full set of `RSS` files and provide a final, flux-calibrated, combined cube ready for science. This class acts as a script for reducing easily the data, and it is called by the automatic script `automatic_KOALA_reduce`.

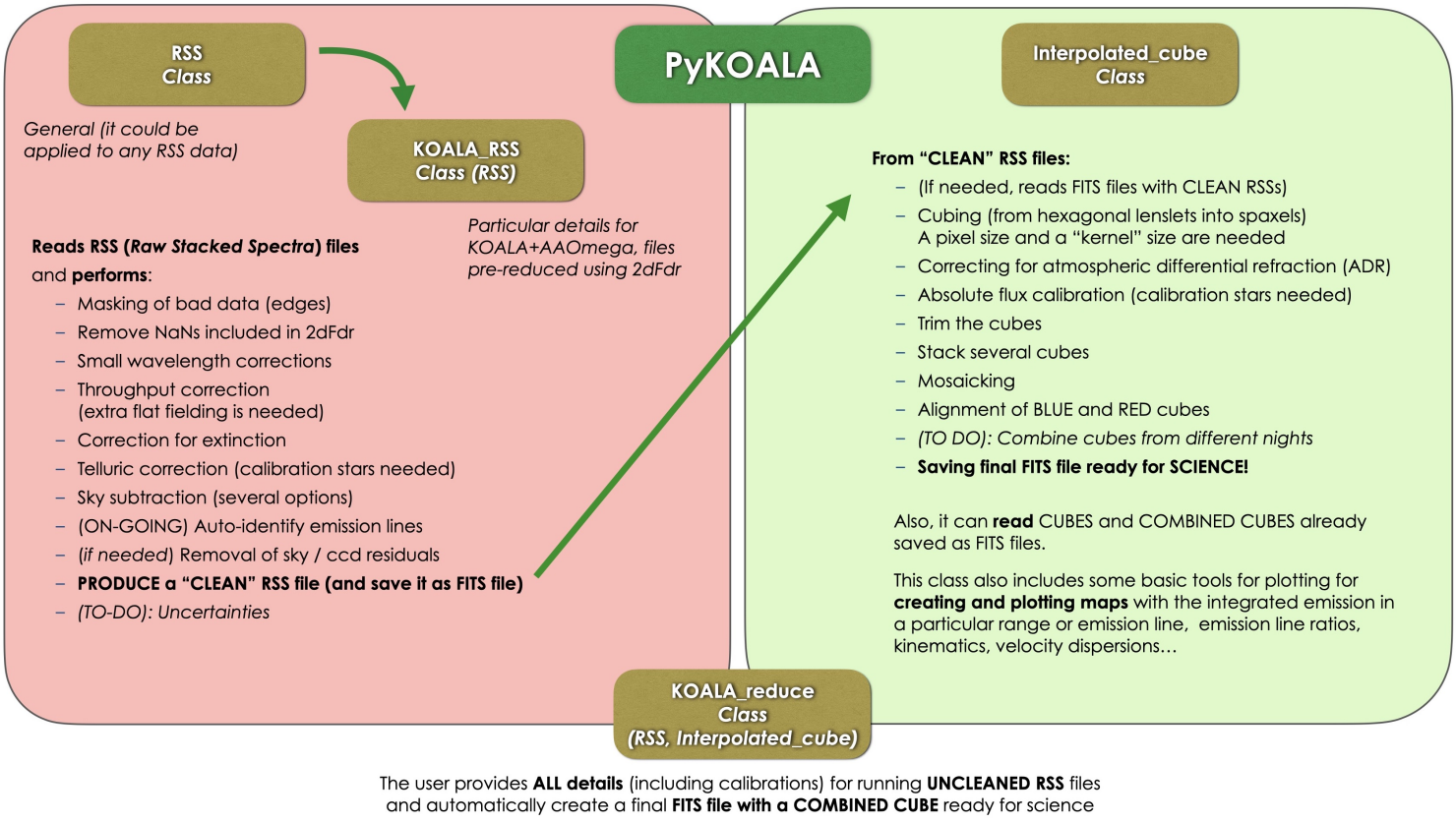

Figure 1: Overview of the PyKOALA classes and main actions included in each of them.

3.2 Overview of the data processing flow with PyKOALA

Independently of the way data are reduced (command line, running `KOALA_reduce`, running the automatic scripts), once the `RSS` files are available (i.e. fibre extracted and wavelength-calibrated, running `2dFdr`), these are the basic steps that are needed for fully reducing IFS data with PyKOALA:

1. Obtain the calibration of the night:
   (a) Obtain the throughput 2D correction using `skyflat` observations (recommended),
   (b) Processing the calibration stars (at least 3 stars are recommended) for:
      i. Getting the absolute flux calibration,
      ii. Getting the telluric correction (only for the red ccd).
2. If needed, processing the SKY data or obtain a 1D / 2D sky file for EACH `RSS` file.
3. Processing the RED data (cleaning `RSS` files, cubing them, combining them into a single cube).
4. Processing BLUE data using same `offsets` (for alignment), `size_arcsec` and `centre_deg` (for creating the same field of view) and `flux_ratios` (if needed, in case frames have different weights because of the weather) as those obtained / used for the RED data
5. Check alignment of RED and BLUE cubes and trim them if needed.

3.3 How to run PyKOALA?

All the PyKOALA code is included in a single, big Python file: `PyKOALA_V1d0.py` (`V1d0` represents the version 1.0 of the code) that can be downloaded from `GitHub`. The PyKOALA distribution also has some extra files needed for calibration, and examples. Copy everything with the same folder structure in a convenient location in your disk (e.g. `/DATA/KOALA/PyKOALA`).

3.3.1 Running PyKOALA in Spyder

It is recommended to use Spyder for running the PyKOALA code. Spyder can be downloaded via Anaconda.

There are a few important setting requirements needed before running PyKOALA in Spyder:

1. **Variables:** Variables assigned on console need to be accessed by script. This is very important for PyKOALA to work!
   (a) Run → Configuration per file → tick "run in console's namespace instead of an empty one" in General settings section → Run
2. **Plots:** Plots should appear in the *Console*. This just makes it easier to see PyKOALA's processes. This is a three step process.
   (a) Python → Preferences → IPython Console → Graphics → Graphics Backend should be "Inline" → Apply and OK
   (b) In the plots window (see red underline in image below) → click the three horizontal lines in the top right corner (see blue square in image below) → untick "mute inline plotting"
   (c) View → Panes → untick "plots".

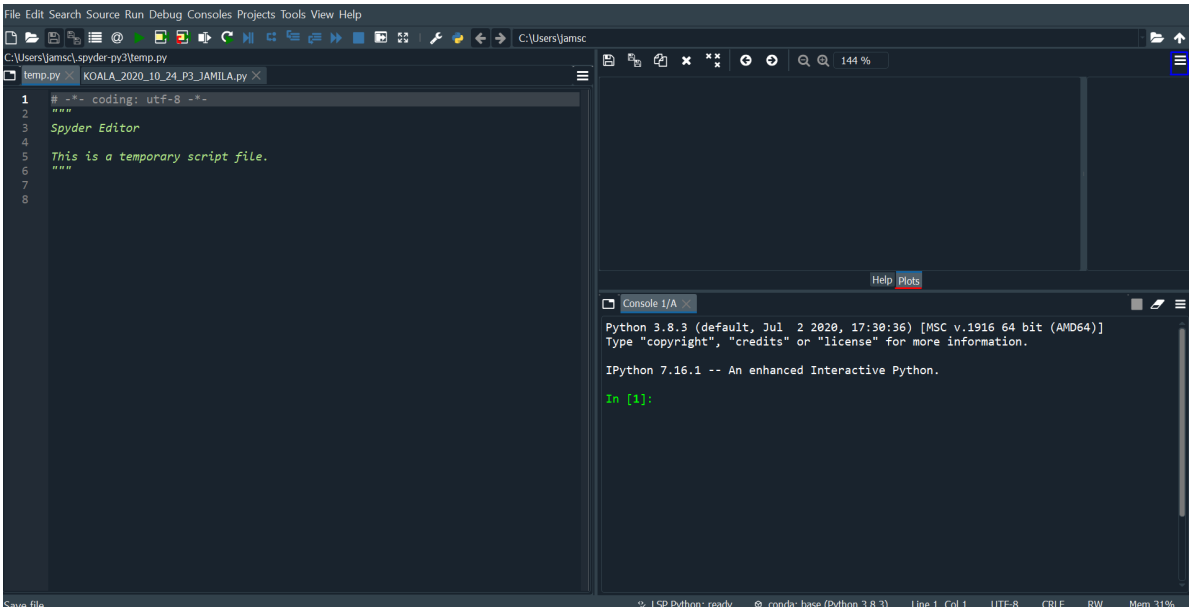

Figure 2: Spyder window ready to run PyKOALA.

3. **Cosmetics (optional):** I personally prefer to only see the *script* and *console* windows (and sometimes the *help* window when looking up functions). It just makes Spyder look tidier and it's simple to just use a split screen.
   (a) View → Panels → untick everything besides "Editor" and "IPython console".

Once Spyder is ready, create an empty `.py` file in the same folder where the PyKOALA code is located (or load the `PyKOALA_testing.py` file included in the PyKOALA distribution) and add this line at the very beginning:

```
exec(compile(open('PyKOALA_V1d0.py', "rb").read(), 'PyKOALA_V1d0.py', 'exec'))
```

this was for Python 3, for Python 2.7 add:

```
execfile('PyKOALA_V1d0.py')
```

Unless you know what you're doing, **there is NO need of loading and running the main Python file into Spyder**. If you do it, because you need to do changes in the code, please make a copy with a different name before editing it.

3.4 Running PyKOALA in a console

PyKOALA can be imported on a console and run in command lines rather than on a software program as Spyder.

```
3  import PYKOALA as PK
```

However, in this case **all** tasks and calls to PyKOALA objects must be done starting with `PK`, e.g.

```
4  cube = PK.Interpolated_cube(rss_file, pixel_size_arcsec=0.7, kernel_size_arcsec=1.5)
```

This has not been fully tested yet, the best way is still loading the `PyKOALA_V1d0.py` file as described in the Spyder section before and forget about adding the `.PK` to the PyKOALA tasks.

We still **strongly recommend** to use a `.py` file for compiling all the steps you're following for reducing the KOALA data with PyKOALA. In case you have to repeat some steps (for sure you will!) or perform a full reduction (e.g. once the issues with the dispersed light correction in 2dFdr is solved), the procedure is already there and can be easily run again.

3.5 Running the automatic scripts

PyKOALA tasks can be run automatically using script files. **This is the easiest way to clean `RSS` files and create a fully-calibrated combined, data cube**. However, if preferred, the user can also run individual tasks in command line, console, or script files. All methods will achieve the same results if the same parameters are used.

Currently PyKOALA includes 3 automatic scripts (see Appendix A, B and C for details):

1. `automatic_calibration_night` for (i) processing the skyflat to obtain the throughput 2D correction, (ii) processing each individual calibration star (this calls `run_automatic_star` and combine the results (for the absolute flux calibration and the telluric correction of the night).
2. `run_automatic_star` for getting the flux calibration (and the telluric correction if needed) of an individual calibration star.
3. `automatic_KOALA_reduce` for fully reducing `RSS` files, cubing them, and combined them in a final, fully-calibrated cube.

The automatic scripts use as an input a `.config` file, which is just a text file with two columns: a **parameter** (left column) and **its value** (right column).

Some `.config` files are available within the folder `CONFIG_FILES`, e.g., `CONFIG_FILES/calibration_night_red.config`. The `.config` file should have all the information needed for running the script.

With this, it is very simple to add files to use and write commands that can be processed automatically, e.g.

```
5  CALIBRATION_NIGHT_FILE = "filePath/fileName.config"
6  automatic_calibration_night(CALIBRATION_NIGHT_FILE)
```

Ideally, this should be all what is needed to type in Python (using Spider or not) for processing the KOALA data with PyKOALA. In reality, being particularly tricky the part of the sky subtraction, or for performing extra tests, running a particular task independently, or creating maps from the cubes, a bit of extra coding will be needed.

3.6 What to expect

The easiest way of reducing the data is to just **run the automatic scripts**. It can be difficult to understand what is needed in each text file for PyKOALA to run, or what each input parameter means or does, but it becomes easier with more experience.

Doing a reduction manually on the console or script is **hard** if you don't know function names (see documentation of function names in Section ****).

If an error does occur and it is to do with the code, double check that everything is typed correctly and if error still persists, contact Dr. López-Sánchez for help (`Angel.Lopez-Sanchez@mq.edu.au`).

4 BASIC RSS PROCESSING:

The standard processing of the RSS files follows these schematic steps (in bold font the letter the code uses for identifying each process) :

1. Reading the data and establishing properties of the Python object,
2. Creating a mask with the valid data (masking the edges of the CCD).
3. Cleaning the CCD defects **C**
4. Fixing small wavelength shifts **W**
5. Applying throughput 2D **T**
6. Correcting for extinction **X**
7. Telluric correction (only for red data) **U**
8. Sky subtraction **S**
9. Correcting negative values **N**
10. Emission lines identification **E**
11. Correcting small CCD / sky residuals **R**
12. Saving the processed / cleaned RSS file.

These steps are followed by KOALA_RSS class, but the user can decide to run each process in a different way.

The upper case letters represents the extra character that is included in an output RSS file when done automatically, e.g., if the RSS file 27feb10037red.fits is processed cleaning CCD defects, applying throughout 2D correction, correcting for extinction, and performing the sky subtraction, the automatic resulting RSS file will be 27feb10037red_TC_X_S___.fits. Note that the S for the sky correction does not include any information about how this has been done or what method has been used (see Sect. **??**).

5 BASIC CUBE PROCESSING:

The standard processing of the clean `RSS` files to build cubes using `Interpolated_cube` follows these schematic steps:

1. Building the cubes from the `RSS` files,
2. Tracing intensity peaks (for finding the centroid of image and getting data for the ADR correction),
3. Perform the alignment of the cubes, obtaining the `offsets`,
4. Rebuilding cubes considering both `offsets` and values for the ADR correction
5. Applying the absolute flux calibration to each cube,
6. Combine flux-calibrated individual cubes into final cube ready for science.

6 OBTAINING THE CALIBRATION OF THE NIGHT

As mentioned before, the easiest way to run PyKOALA is using the automatic scripts. For running the calibration of the night we use the `automatic_calibration_night()` script, that calls an external `.config` file (that is a simple `txt` file) compiling all the information needed for running the script:

```
5  CALIBRATION_NIGHT_FILE = "filePath/fileName.config"
6  automatic_calibration_night(CALIBRATION_NIGHT_FILE)
```

This script will process a `RSS` file containing a `skyflat` to obtain the 2D throughput correction and/or a list of given `.config` files, each of them containing the information for processing a calibration star.

Appendix A shows an example of the `.config` file `calibration_night.config` needed for running the automatic script `automatic_calibration_night()`.

6.1 Processing the `skyflat` to get the 2D throughput correction

6.1.1 INPUT files (RSS):

- `file_skyflat`: the `RSS` file with the `skyflat`.
- `rss_star_file_for_sol`: A `RSS` file of a star, only if fixing `2dFdr` wavelengths is requested.

6.1.2 Description

If the `.config` file includes `do_skyflat = True`, the script will process the `RSS skyflat` file provided by parameter `file_skyflat`, or the Python object provided by `skyflat` (if both are given, the Python object will be considered as this is slightly faster).

The `skyflat` file will be processed correcting the CCD defects (if `correct_ccd_defects = True`) and/or fixing the small wavelength shifts (if `fix_wavelengths = True`). In the second case, an extra parameter is needed: or an input `RSS` file with a star to compute the small wavelength shifts (given with the parameter `rss_star_file_for_sol`) or the solution of the 2nd-order polynomium (given by the parameter `sol`).

Although it is a bit extra time consuming, we recommend to use `fix_wavelengths = True` and later save the values provided in `sol` for correcting **all the RSS files** of the same night (as easy as including this parameter and its value in each of the `.config` files we use), as this correction **should not change** during all the night.

For more details about `correct_ccd_defects` and `fix_wavelengths`, see Sect. **??** .

After processing the `skyflat RSS` file, the script will run the `RSS` task `get_throughput_2D()` to obtain the 2D throughput correction (see Fig. 3). This correction is saved as a `.fits` file and will be needed in **ALL THE SCRIPTS**. In some way, the wavelength-dependence of the throughput can be considered as an extra flatfielding correction.

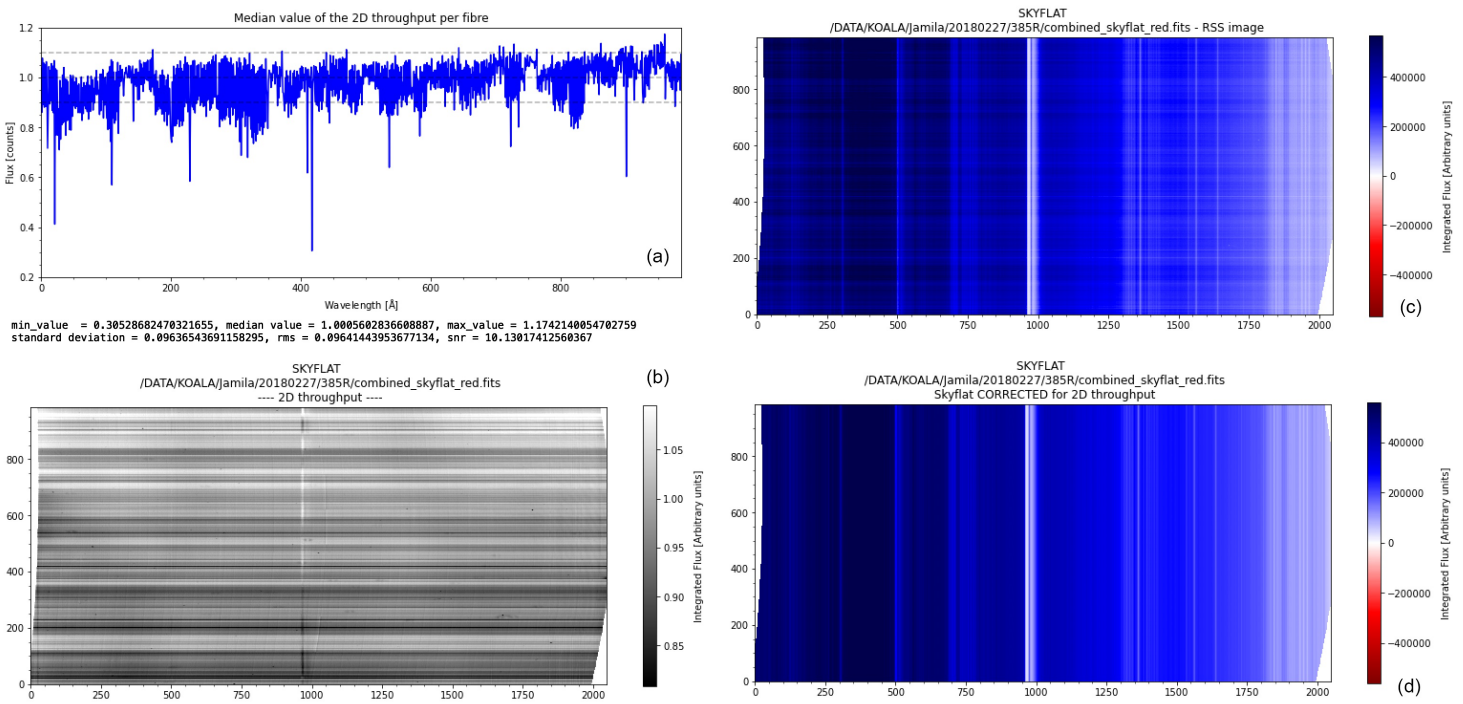

Figure 3: *Some screenshots of the plots resulting on running* `get_throughput_2D()`*: (a) The median value of the throughput per fibre. (b) The 2D throughput correction. (c) The* `skyflat RSS` *before correcting for throughput 2D. (d) The* `skyflat RSS` *after correcting for throughput 2D: all the horizontal jumps have now disappeared thanks to applying this correction.*

6.1.3 OUTPUT files:

- `throughput_2D_file`: This is a `.fits` file containing the 2D values of the throughput, i.e., the correction needed to get all fibres scaled to the same value at all wavelengths (panel b in Fig. 3).

6.2 Automatically Processing calibration stars

6.2.1 INPUT files:

- list of the `.config` files for calibration stars. Can be associated to a Python object using parameter `object`.

6.2.2 Description

The `automatic_calibration_night()` script will also process all the stars taken during the night and produce the absolute flux calibration of the night as well as the telluric correction (only for red). See Sect. 7 for details on how to reduce calibration stars.

For this we add `CONFIG_FILE` and the name of the `.config` file that we create for reducing and processing each star.

This script will combine the results of each star and produce a calibration for the night (also the telluric correction for the red). An example is shown in Fig. 4. Plots comparing the results of each pair of stars are also produced.

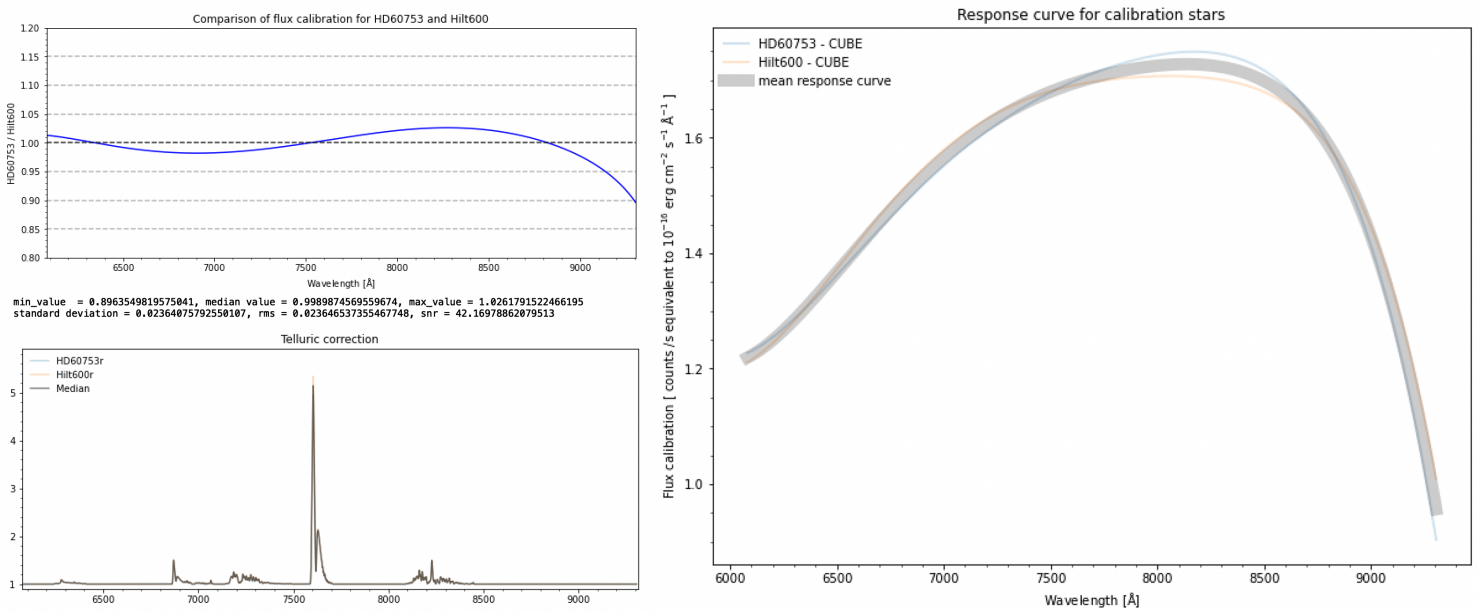

Figure 4: *Comparison of the results of the absolute flux calibration and the telluric correction obtained for stars HD 60753 (blue) and Hilt 600 (red) using the 385R (red) grating. The combined results are shown as a grey line. The variation in the flux calibration is less than 2.4%.*

IMPORTANT: If associating `CONFIG_FILE` with Python objects given by `object`, these must be given in the same order, i.e., the first `CONFIG_FILE` will be associated to the first `object`, the second `CONFIG_FILE` with the second `object`...

6.2.3 OUTPUT files:

- `flux_calibration_file` A `.txt` file compiling two columns, the first one with the wavelength and the second one with the corresponding flux calibration.
- `telluric_correction_file` A `.txt` file compiling two columns, the first one with the wavelength and the second one with the corresponding telluric correction. This is only needed for the red spectrum.

6.3 Fully automatic calibration of the night

If you know what you're doing and you're familiar with the calibration stars you observed, you can just run the following command to get the fully automatic calibration of the night:

```
automatic_calibration_night(path=path, auto=True)
```

being `path` the full path to a folder where the `2dFdr` products are located. This will need ∼10 minutes for completely processing the `skyflat` and 3 calibration stars, each with 3 `RSS` files.

7 REDUCING A CALIBRATION STAR

Observations of spectrophotometric stars and / or telluric stars are needed for properly calibrating the data. Your KOALA observations should have included at least 2 (ideally 3-4 during the night) calibration stars. Usually these have been chosen from a catalogue such as the European Southern Observatory Standard Stars Catalogue.

For reducing a calibration star and obtain the absolute flux calibration (and the telluric correction in the red) we use the `run_automatic_star()` script, that calls an external `.config` file (that is a simple `txt` file) compiling all the information needed for running the script:

```
CALIBRATION_STAR_FILE = "filePath/fileName.config"
Hilt600r=run_automatic_star(CALIBRATION_STAR_FILE)
```

The script returns a Python object (`Hilt600r` in the example, it will also create the Python object named with variable `obj_name` in the `.config` file) with all the information of the calibration star (see Sect. **??** for describing the structure of Python objects created by PyKOALA).

Appendix B shows an example of the `.config` file `calibration_Hilt600_red.config` needed for running `run_automatic_star()`.

Note that `run_automatic_star()` is defined as a task, it can also run providing the parameters within the (), but needs to read a `.config` file, and the parameters and values included in that `.config` file will have preference over the parameters listed within the ().

7.1 INPUT files:

- `RSS star files`: Ideally 3 files per object taken at slightly different positions (dithering).
- `throughput_2D_file`: A `.fits` file with the 2D throughput correction.
- (*optional*) a `.txt` file, `absolute_flux_file`, compiling the absolute flux calibration of the star (the majority of these files are already included in PyKOALA but the user can select other files).

7.2 Description

3 steps are needed for obtaining the absolute flux calibration (and the telluric correction in the red) from a calibration star:

1. "Clean" the provided `RSS` files using `KOALA_RSS()`,
2. cube each `RSS` file, perform the alignment, and combine them into a single cube,
3. run the tasks needed for obtaining the calibrations using the combined cube.

7.2.1 Cleaning `RSS` files of a calibration star

All the details of how cleaning a `RSS` file are described in Sect. **??**. Here we only provide the key parameters needed for cleaning the `RSS` files of calibration stars.

For processing the `RSS` files of calibration stars, we `correct_ccd_defects` and strongly recommend to `fix_wavelengths` using the solution `sol` obtained in Sect. 3. Then, we `apply_throughput` providing the `.fits` file with the 2D throughput correction obtained running the `automatic_calibration_night()` script (see Sect. 3). The data must be corrected for extinction using `do_extinction`. For calibration stars, the easiest and most efficient way of subtracting the sky is using `self` for `sky_method`. This will consider the `n_sky` (typically 50 - 100) fibres with the lowest integrated value (see Fig. 5) to create a 1D sky spectrum.

Additionally, we can choose to `correct_negative_sky`, `remove_negative_pixels_in_sky` and/or `clean_extreme_negatives` but again for calibration stars, as these are bright objects having easily 3-4 order of magnitude difference between the background and the signal, it is not that critical (see e.g. right panel in Fig. 5).

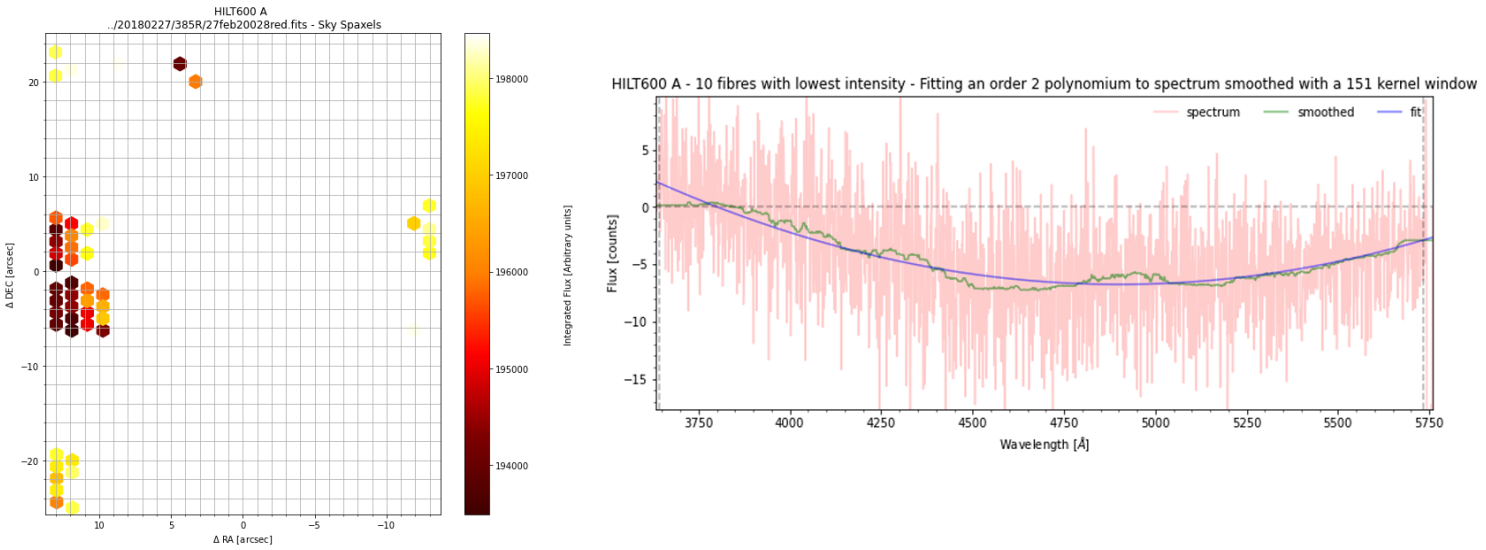

Figure 5: *(Left) Automatic identification of the 50 fibres with the lowest integrated intensity (*`n_sky = 50`*) for obtaining an 1D sky spectrum using the* `self` *method. (Right) Correction of the negative sky (if* `correct_negative_sky = True` *considering the 10 fibres with the lowest integrated intensity values (the integrated value of this spectrum is smaller than 0), fitting a 2nd order polynomium and adding it to all fibres. Notice that this correction is very small (less than 10 counts per wavelength) and hence negligible when considering that the flux of the calibration star is typically thousands or tens of thousands counts per wavelength.*

The important parameters that are needed in the `.config` file for cleaning `RSS` files of calibration stars are:

```
apply_throughput            True
throughput_2D_file          throughput_2D_20180227_385R.fits    # Fits file with the 2D throughput
correct_ccd_defects         True
fix_wavelengths             True
sol                         [0.06039,-0.0010,2.2712e-07]        # If known, otherwise it will compute it automatically
do_extinction               True
sky_method                  self                    # Method applied for subtracting the sky.
n_sky                       50                      # Number of fibres used to create a sky spectrum (default = 100)
#sky_fibres                        range(0,50)      # A list or a range of fibres can be provided using this parameter
#remove_5577                       True             # Only valid for the BLUE ccd if skyline 5577 is observed
#correct_negative_sky              True
#remove_negative_pixels_in_sky     True
#clean_extreme_negatives           True             # if True it replaces extreme negative (lower than percentile_min)
#percentile_min                    0.9              # by median value of fibre
```

7.2.2 Obtaining the combined cube of a calibration star

This part is very easy for stars, the default cubing parameters will do the job without the need of providing anything else, besides of course the `pixel_size_arcsec` and the `kernel_size_arcsec`. Note that it is important that the calibration star is cubed using the same `pixel_size_arcsec` and `kernel_size_arcsec` values that our science cube will have for obtaining the correct flux calibration (this is not important for the telluric correction, as this is normalized).

It is recommended to correct for atmospheric differential refraction (`ADR`) and `trim_cube`. The procedure for aligning cubes works extremely well with calibration stars (see example in Fig. 6, with variations of the order of 0.01"), providing the exact `offsets` that the telescope did between frames.

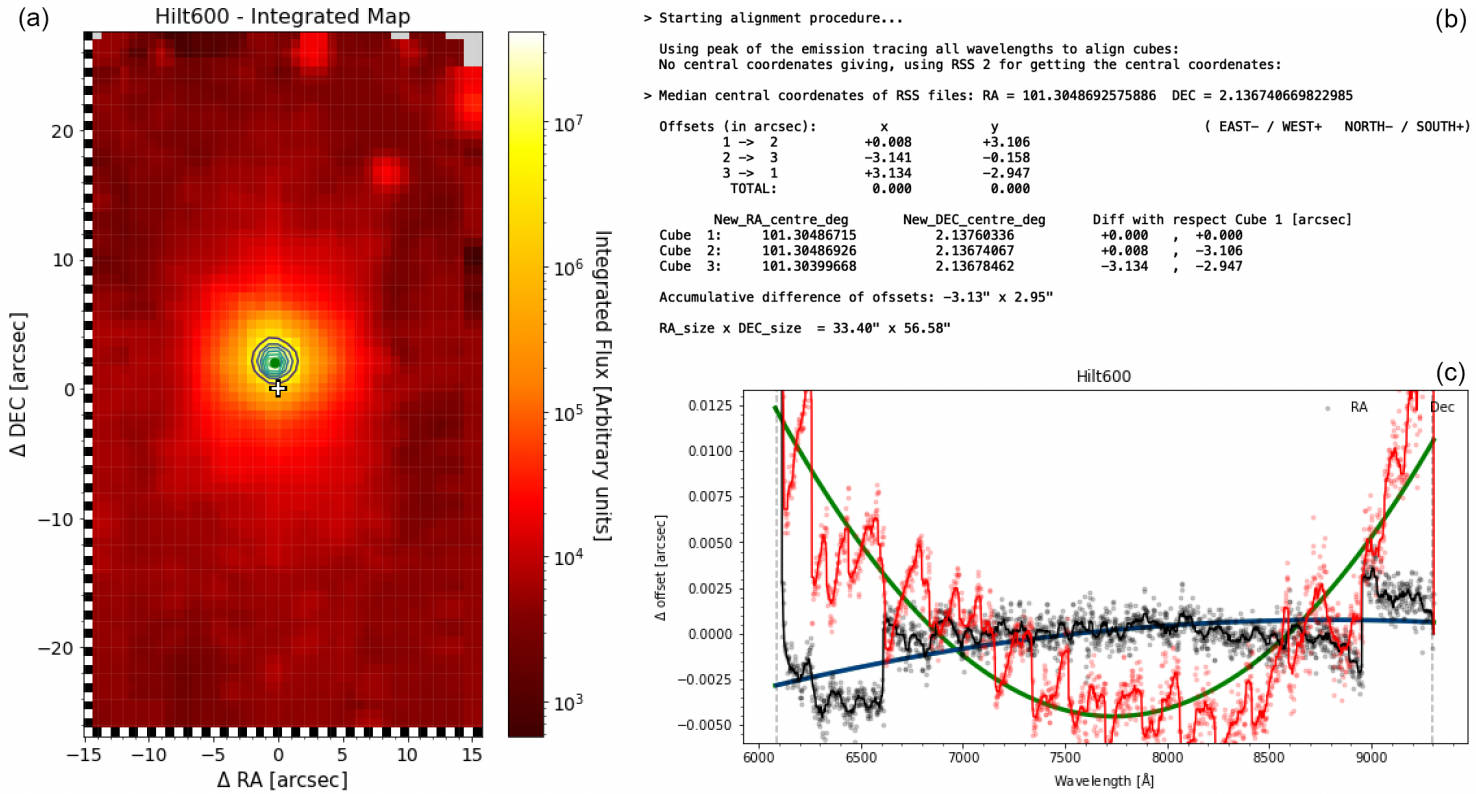

Figure 6: *(a) Integrated map of the combined red cube of the calibration star Hilt600. The color scale is logarithmic, hence the difference in flux between the calibration star and the background is around 4 orders of magnitude. (b) Results of the alignment procedure of the three* `RSS` *files used for this calibration star. The telescope moved 3.1" S from* `RSS 1` *to* `RSS 2` *and 3.1" E and 0.16"N from* `RSS 2` *to* `RSS 3`*. (c) Result of the* `ADR` *correction, that has been applied in both the RA (black) and DEC (red) axes, the median residual value in each axis is smaller than 0.01". The "zig-zag" pattern in the DEC axis, which here has variations of ∼0.005", is consequence of cubing small wavelength intervals (automatically computed to accept variations smaller than 0.01", that is given by the default* `jump = -1` *value) instead of doing it wavelength by wavelength, this can be modified using* `jump = 1`*.*

When combining several `RSS` files, it is possible to scale each cube using the integrated value of the common area as reference. For combining cubes from a calibration star, assuming that the night is photometric, this should not be needed, and hence the default value of `scale_cubes_using_integflux` is `False`.

Once the combined cube is created, the script will save the Python object in the variable we indicated, e.g., `Hilt600r` following Code Line 8. This name **should be the same** that the corresponding `obj_name` parameter in the `.config` file. If running the script of the calibration star from `automatic_calibration_night()`, this name **should also be the same** than that given in the corresponding `object` parameter in the `.config` file for running the automatic calibration of the night.

7.2.3 Extracting the integrated spectrum of the star

The integrated spectrum of the star is obtained calling the `Interpolated_cube` task `half_light_spectrum()` considering `r_max = 5`, i.e., extracting all the light in a circular aperture centred at the peak of the star and with radius 5 times the half light radius. The value of `r_max` can be changed in the `.config` file). For further improving the quality of the spectrum, the background is subtracted considering an annulus with size `sky_annulus_low_arcsec = 5` and `sky_annulus_high_arcsec = 10` (default values, these can be changed in the `.config` file), as shown in Fig. 7.

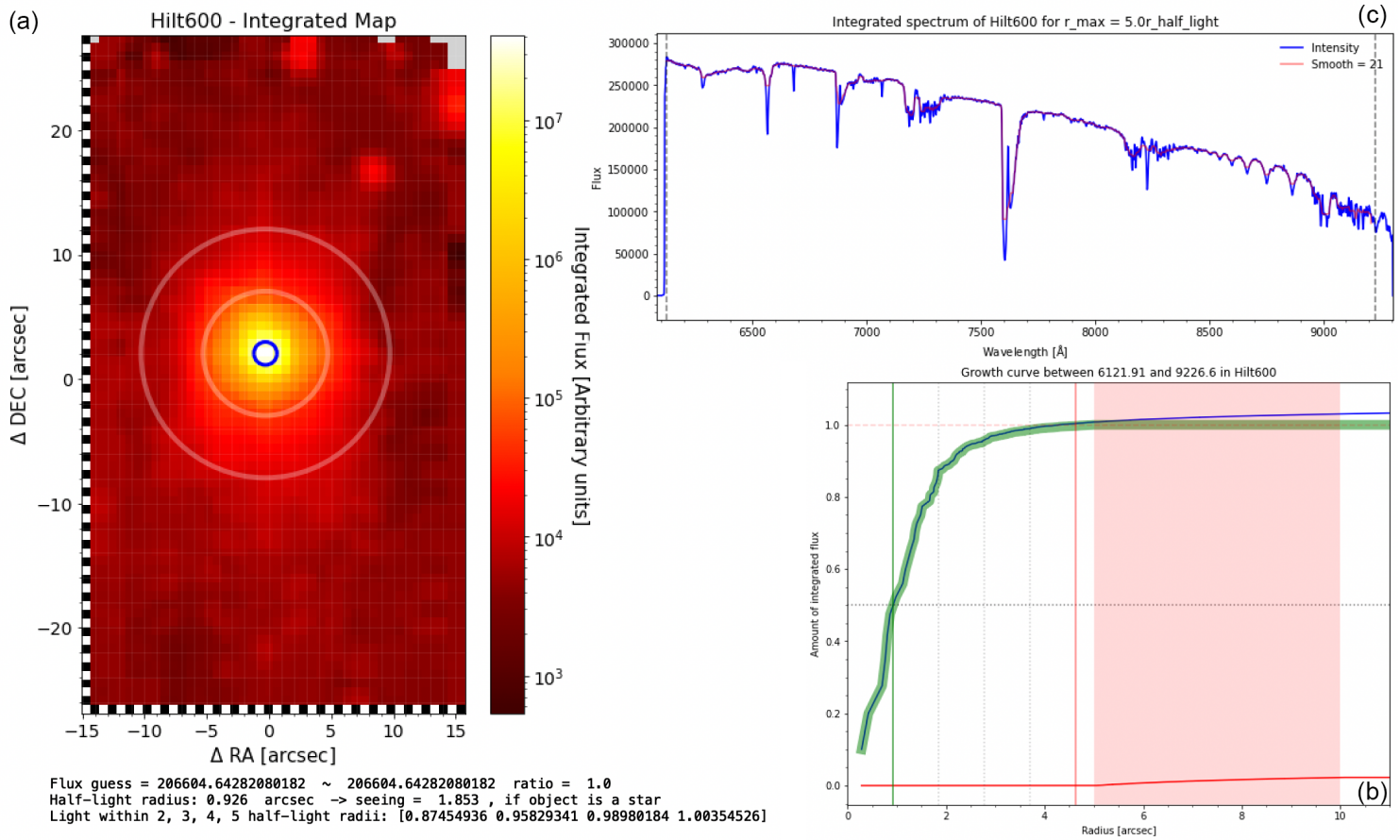

Figure 7: *(a) Integrated map of the calibration star Hilt600 showing the size of the seeing (blue circle), 1.85" in this case, corresponding to twice the value of the half light radius (0.93" in this case), and the two limits for subtracting the background:* `sky_annulus_low_arcsec = 5` *and* `sky_annulus_high_arcsec = 10` *. (b) The growth curve of the calibration star (blue line) indicating the value of the seeing (green vertical line) and* `r_max = 5` *(red vertical line). The red coloured region corresponds to the sky defined by the annulus. The red continuous line is the contribution of the background (which is 0.76% of the total flux in this case), and it is subtracted to make the total flux of the star (green line). (c) The integrated spectrum of the star (blue) and a smoothed spectrum (red).*

The integrated spectrum is saved in Python object variable `cube.integrated_star_flux`. The seeing is also computed considering that is twice the value of the half light radius, and stored in the Python object variable `cube.seeing`.

This step is automatically performed when running the script `run_automatic_star()`.

7.2.4 Obtaining the telluric correction

This section only applies for the red spectrum. We need to call the `Interpolated_cube` task `telluric_correction_from_star()`, providing a `list_of_telluric_ranges` (the ranges we need to fit for obtaining the telluric correction), and `order_telluric` (the order of the fit, usually 1 –lineal– or 2 recommended).

This step may need a bit of iteration with the user to nicely find the best values of the `list_of_telluric_ranges` and `order_telluric`, although in popular calibration stars the automatic values should be enough (in any case, always check the results!). Be sure that `apply_tc = True` **only** when being completely sure that the telluric correction is good.

By default, PyKOALA considers 5 telluric ranges:

```
list_of_telluric_ranges =  [ [6150,6240,6410,6490],
                             [6720,6855,7080,7140],
                             [7080,7140,7500,7580],
                             [7400,7580,7705,7850],
                             [7850,8090,8450,8700] ]
```

being `[c1,r1,r2,c2]` the values of the telluric feature we need to correct `[r1,r2]` and the two wavelength intervals for fitting the continuum, `[c1,r1]` and `[r2,c2]` (see Fig. 8).

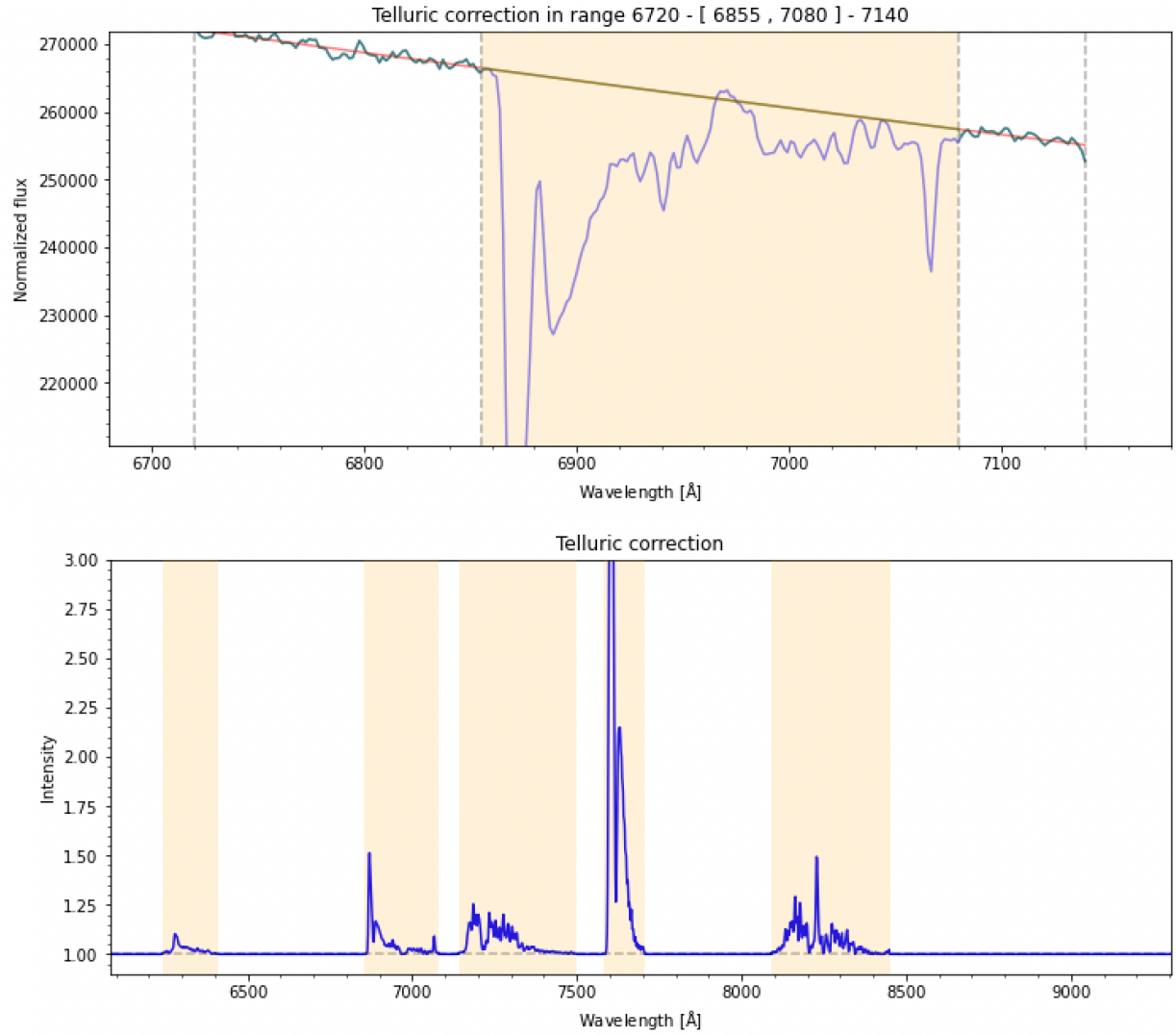

Figure 8: *(Top) Example of obtaining the telluric correction in the star Hilt600 in the range* `[6720,6855,7080,7140]`*; these values are indicated by the dashed vertical lines. The yellow box represents the telluric feauture to be corrected,* `[r1,r2]`*. The blue line is the spectrum. The red continuous line is the* `order_telluric` *polynomium obtained with the fit. The green continuum line is the spectrum corrected for the telluric absorption. (Bottom) Total telluric correction of the star Hilt600. The yellow regions indicate the telluric ranges.*

IMPORTANT: For applying the telluric correction (something that is needed for obtaining a good absolute flux calibration) we need to apply it on the combined datacube of the calibration star. This is done only if parameter `apply_tc` is `True`. However, doing this will modify the combined datacube of the star, that should have to be created again if the applied telluric correction is not correct. Select `apply_tc = True` **only** when being completely sure that the telluric correction is good.

The telluric correction is saved in the Python object variable `cube.telluric_correction`. This step is automatically performed when running the script `run_automatic_star()`. The script will also save the integrated spectrum of the star before and after applying the telluric correction, as well as `cube.telluric_correction`, in `.txt` files.

7.2.5 Obtaining the absolute flux calibration

The absolute flux calibration (also known as the response curve) can be obtained applying the `Interpolated_cube` task `do_response_curve()`. This tasks needs as input a `absolute_flux_file` for the data of the absolute flux calibration of the star (many of these are included in PyKOALA and read by the automatic script), the `step` (in Å) interval that will be used, the `fit_degree` of the polynomium that will fit the smoothed raw response curve (we recommend using low odd values: 3, 5 or 7), and the width (in Å) of the Hα absorption, `ha_width` (recommended).

If `do_response_curve()` is run after the telluric correction is applied, it will need to recompute the integrated spectrum of the star (as explained in Sect. 7.2.3). After that, the task will obtain the raw response curve (parameter `_response_curve_` in the code) applying:

$$\texttt{\_response\_curve\_}(\lambda) = \texttt{measured\_counts}(\lambda)/\texttt{flux\_cal}(\lambda)/\texttt{exp\_time} \quad (1)$$

being `_response_curve_`(λ) the raw response curve at wavelength λ given by the data provided in the absolute calibration file, `measured_counts`(λ) the integrated flux of the star at that λ, `flux_cal`(λ) the absolute flux calibration value provided by `absolute_flux_file` at that λ, and `exp_time` the exposition time (in seconds) of the cube (that can be read from the header or from the Python variable `cube.exptimes`, note that this variable will have a list of the exposition times of the `RSS` files used for building the cube, which are expected to be all equal). See left panel of Fig. 9 for an example of the comparison of `measured_counts`(λ) (green crosses) and (scaled) `flux_cal`(λ) (black stars) in calibration star `Hilt600`.

After this, the task will remove the problematic data around Hα, as well as any other bad wavelength range we wish to skip (provided in `exclude_wlm`, which compiles a list of bad ranges). Then, `do_response_curve()` will obtain the response curve following three methods:

1. Fitting of a `fit_degree_flux`-order polynomium (blue line in the right panel of Fig. 9),
2. Smoothing of the raw response considering an `odd_number` window (this number can be computed automatically if not given) and fitting a `fit_degree_flux`-order polynomium (purple line in the right panel of Fig. 9),
3. Smoothing of the raw response curve considering an `odd_number` and a `smooth` parameter (default value: `smooth=0.05`), orange line in the right panel of Fig. 9).

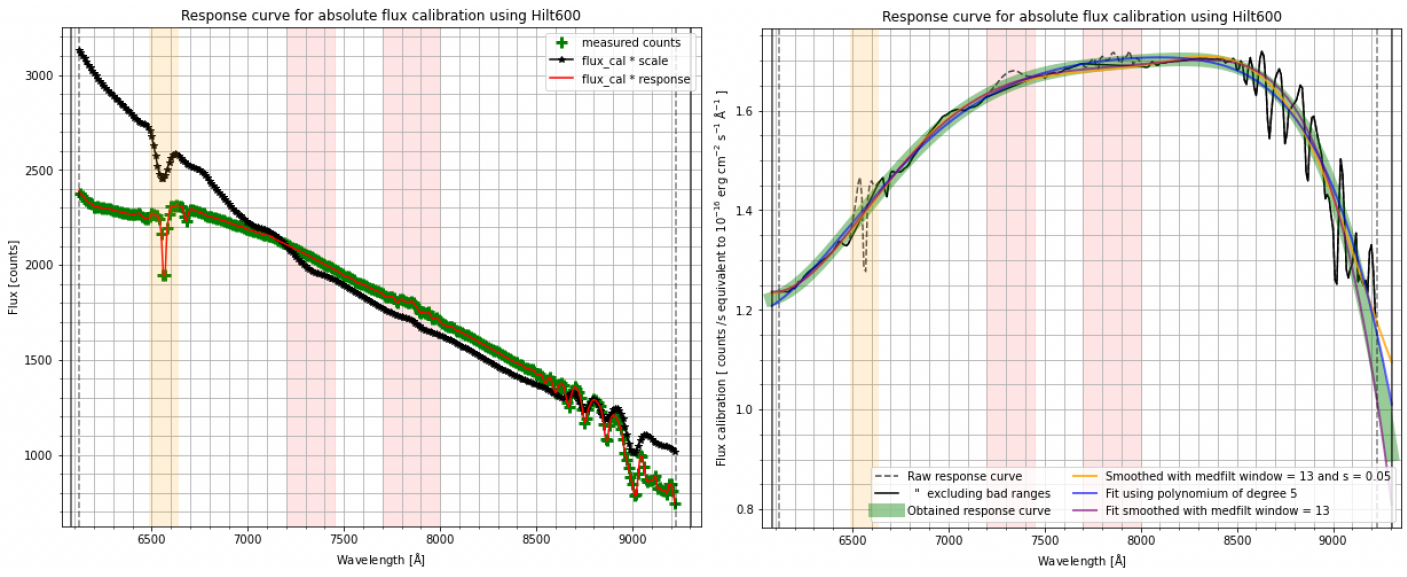

Figure 9: *(Left) Comparison of the* `measured_counts`(λ) *(green crosses and red line) and (scaled) absolute flux values,* `flux_cal`(λ)*, (black stars) in the calibration star* `Hilt600`*. (Right) Raw response curve for the calibration star* `Hilt600` *(black lines, the dashed line indicates the discarded ranges), fitting of a* `fit_degree_flux`*-order polynomium (blue line), fitting of a* `fit_degree_flux`*-order polynomium after smoothing with an* `odd_number` *window (purple line), the smoothing of the raw response curve considering an* `odd_number` *and a* `smooth` *parameter (orange line), and the final response curve (think green line), in this case considering* `fit_weight = 0.5, smooth_weight = 0.0`*. In both figures, the vertical dashed lines delimit the wavelength range* `[min_wave_flux, max_wave_flux]` *(default values are* `[cube.valid_wave_min,cube.valid_wave_max]`*). The yellow region indicates the Hα absorption, and the red regions the wavelength ranges that have not been considered.*

The user can select which one of these solutions (or combination of these solutions, using the parameters `fit_weight` and `smooth_weight`) to obtain the final response curve. As this is only valid in the wavelength interval `[min_wave_flux, max_wave_flux]`, the response curve will be interpolated to include the edges (thick green line in right panel of Fig. 9).

The response curve (i.e. the absolute flux calibration at all wavelengths) is stored in the Python variable `cube.response_curve`. The default method is considering the fitting of a `fit_degree`-order polynomium. This step is automatically performed when running the script `run_automatic_star()`, that will also save the absolute flux calibration (response curve) in a `.txt` file.

7.3 OUTPUT files:

- `.fits` file (`fits_file`) with the combined datacube of the calibration star.
- `.txt` file with the integrated spectrum of the calibration star.
- `.txt` file with the telluric correction (`telluric_file`), only for the red.
- `.txt` file with the absolute flux calibration (`response_file`).
- (Optional) **Cleaned `RSS` files**, saved following the automatic naming (see Sect. 4).
- (Optional) **Individual aligned cubes**.

8 REDUCING A SCIENCE TARGET

Although it is possible to reduce the science object manually, i.e., following the steps summarized in Sects. 4 and 5, we strongly recommend use the `KOALA_reduce()` class or the automatic script `automatic_KOALA_reduce()` which directly calls `KOALA_reduce()`. We can easily run the script with

```
29 KOALA_REDUCE_FILE = "./path/name.config"
30 He2_10r = automatic_KOALA_reduce(KOALA_REDUCE_FILE)
```

The script returns a Python object (`He2_10r` in the example) with all the information of the science target (see Sect. **??** for describing the structure of Python objects created by PyKOALA). Note that this Python object will need a lot of memory if many `RSS` frames (>10) are combined, as it keeps all the information of the `rss` files and the derived `cubes` and `cubes_aligned`, as well as the `combined cube`.

Appendix C compiles an example of the `.config` file `koala_reduce_20180227_He2-10_red.config` needed for running the automatic script `automatic_KOALA_reduce()`.

Although much of the reduction process does not need much supervision from the user (just checking that the tasks work as they should), there is a particular process that needs a lot of attention: the sky subtraction, as we will detail below.

8.1 INPUT files:

- `RSS` **science files, ideally taken with some dithering**
- **Throughput 2D file** This file was obtained using the skyflat..
- **Flux calibration file** This is the response file from the calibration star.
- **Telluric Correction File** Only for red arm, obtained from the calibration star.

```
1  file_in  = "path/name.fits" ## Galaxy RSS file
2  file_out = file_in.replace(".fits","_TC_X.fits")
3
4  ######### Rename file to account for processes applied (see output file section for renaming) ##########
5
6  RSS_Object_Name = KOALA_RSS(file_in, save_rss_to_fits_file=file_out,
7                              no_nans=False,
8                              apply_throughput=True,
9                              nskyflat=True,
10                             throughput_file = "../path/file.dat",
11                             nskyflat_file = "../path/file.fits",
12                             correct_ccd_defects = True,
13                             do_extinction=True,
14                             fix_wavelengths = True,
15
16                             remove_5577 = True, # Only for blue
17                             sol = [-1], # Only for red
```

```
                telluric_correction_file = "path/name.dat", # Only for red
                brightest_line_wavelength = 6581, # Only for red
                sky_lines_file = "name.dat", # Mainly for red
                correct_negative_sky = True,
                clean_cosmics = True,
                clean_extreme_negatives = True,
                percentile_min = 0.9, # Needed for clean_extreme_negatives
                sky_method="self", # Choose sky method
                n_sky=30,
                plot=True, warnings=False)
```

Code 1: Manual code for Galaxy Reduction (creating RSS object)

The RSS files will be cleaned in the same way as the star and saved. After these process have taken place, a data cube will be created per RSS file and then combined to create the final cube. The cubing process is essentially the same as the star. **Remember** to not the *sol* value in the red data and use this value when reducing the blue.

The figures below are taken from the galaxy He2-10 which was observed twice (top and bottom). The process is the same for both sections which can then be combined using the same cubing technique as the star. A brief recap is below.

8.2 Cubing

8.2.1 Alignment

Aligning the cubes using the blue square and green circle method is the same process as the calibration star. Since the galaxy has a nearby star the green circle is slightly weighted off the galaxy's centre. The alignment of both halves will be based on the location of the blue cross.

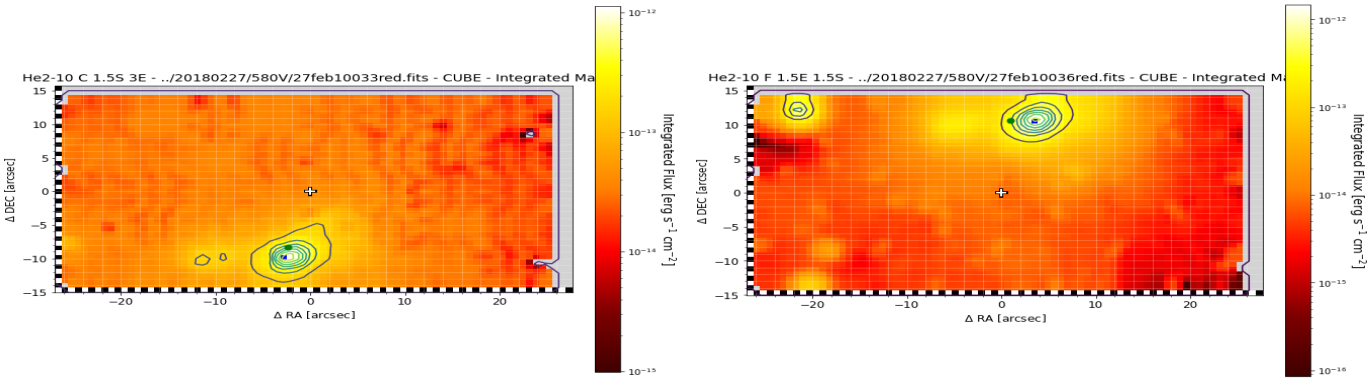

Figure 10: Upper (left) and Lower (right) Galaxy Alignment

8.2.2 Weighted Map

The weighted map shows how all cubes are combined and where each cube sits in relation to the other. The edges will be trimmed as there is little to no information gained. check this part?

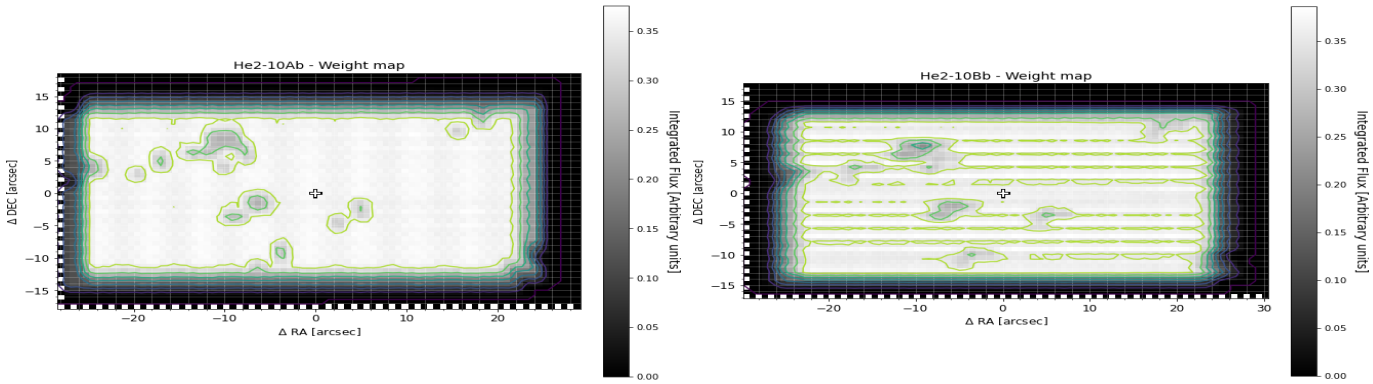

Figure 11: Weighted Map of Upper (left) and Lower (right) Parts of the Galaxy

8.2.3 Final Cube

The final cubes are trimmed and complete.

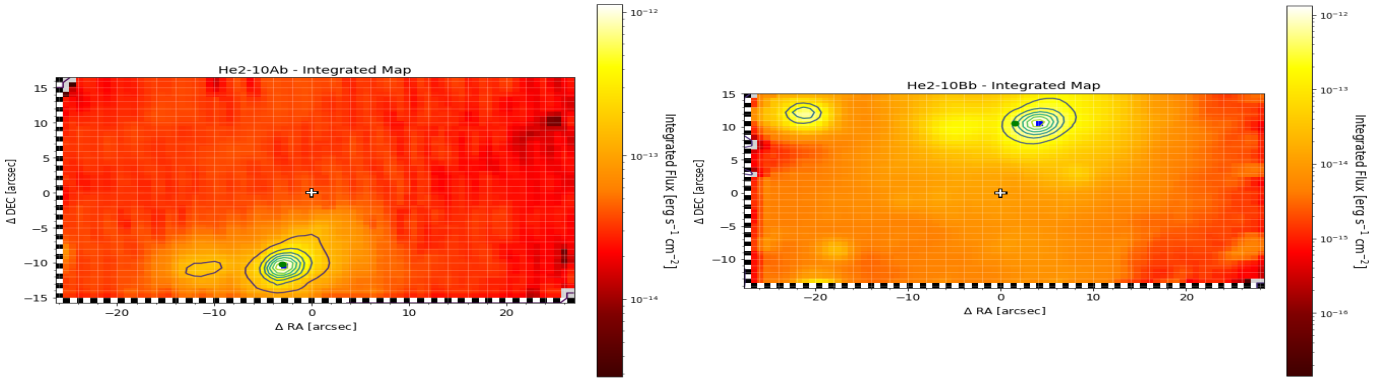

Figure 12: Trimmed Cube Upper (left) and Lower (right) Sections

Since the galaxy was observed in two sections, running a new automatic script (see Appendix 3) can be used to combine both cubes to create a final image of the whole galaxy.

8.2.4 Combining Different Cube Locations

An easy method is to run the same automatic script for the galaxy but combining all cleaned RSS files in the same file and giving the offsets manually. PyKOALA will do the same cubing process and combine each section automatically.

Since the input files are cleaned, the cubing process can start almost immediately (after creating a mask for each file).

The following figure Fig. 13 shows the combined cubes alignment.

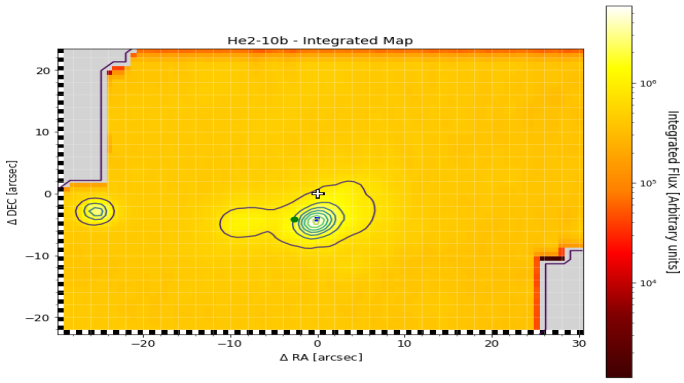

Figure 13: Trimmed Integrated Flux Heat Map

Finally, the final cube of the galaxy is created!

OUTPUT files:

- **fits_file** This will be a used to save the cube.
- **Cleaned RSS files** Each cleaned RSS file will be saved with a specific letter denoting the process or processes that have been completed. If a process was not completed then the file will have an underscore in place of the letter (note that by default all files will start with an underscore).
  - **T** Throughout applied
  - **C** CCD defects corrected
  - **W** Wavelengths were corrected
  - **X** Extinction corrected
  - **U** Telluric corrected
  - **S** Sky subtracted
  - **E** Emission lines identified
  - **N** Negative values corrected
  - **R** Sky and CCD residuals corrected

9 RSS CLASS

This section gives the details of the RSS class in PyKOALA. To create an Python object of the RSS class we simply read a KOALA data file (a RSS obtained with 2dFdr, see Section ***) using KOALA_RSS:

```
path = "/DATA/KOALA/Jamila/20180227/385R/"
file_in  = path+"27feb20025red.fits"
test = KOALA_RSS(file_in)
```

This just creates the object, test, with the information provided in the fits file, nothing else.
RSS objects created by PyKOALA have **properties** and **functions** (callable). Some properties are:

```
test.wavelength                # wavelength vector
test.intensity_corrected       # data: [fibre,wavelength]
test.intensity_corrected.shape # shape of the data: 986 spectra with 2048 wavelengths each
test.history_RSS               # history of the RSS
test.airmass                   # airmass of the observation
test.exptime                   # exposition time of rss
test.filename                  # name of the fits file used for creating this RSS object
```

See the complete list of RSS properties in Table ***. Some useful RSS functions are:

```
test.RSS_image()                            # For seeing the data as a RSS image:
test.RSS_map()                              # To plot the map integrating fluxes:
test.integrated_fibre_sorted[-1]            # To know which spaxel has the brightest spectrum
test.plot_spectrum(443)                     # To see the spectrum of a spaxel:
test.plot_spectra(list_spectra=[10,20,30])  # To plot several spectra in the same plot:
```

Python Notes:
- Parameters or properties **ARE NOT** callable, and hence don't end in ().
- Functions or definitions **ARE** callable, and must end in ().
- Many times the brackets includes the options, e.g. `test.RSS_image(image=test.intensity, cmap="binary_r")`.

In the next subsections we provide a detailed description of how perform the "cleaning" of the RSS files using the KOALA_RSS class in PyKOALA. All these steps are compiled in __init__.

9.1 Reading the data

First we read the data (stored in self.intensity) and establish all the properties of the RSS object that are read from the header (e.g. wavelength, number of wavelenghts given by n_wave, number of good spectra given by n_spectra, filename, exptime, center coordenates, offsets, list of good fibres, ...)

9.2 Creating/applying a mask with the valid data (masking the edges of the CCD)

If no mask is specifically given (this can be a fits file using mask_file or a Python variable using mask), PyKOALA runs function self.get_mask().

If the mask is given, PyKOALA runs function self.read_mask_from_fits_file().

The mask is stored in self.mask, that is a 2D list, with n_spectra elements each. The first list compiles when the good data start in the left edge (short wavelength), while the second list indicates when the good data finishes in the right edge (long wavelength), e.g. self.mask[1][500] will indicate the wavelength index where the good data finishes in fibre 500.

The range where all fibres have good data is stored in test.mask_good_wavelength_range and test.mask_good_index_range, which both are lists with two numbers.

In any case, PyKOALA establish the VALID wavelength range, i.e., the wavelength range for which all fibres have VALID data. This range is given by properties self.valid_wave_min and self.valid_wave_max, that can be given by the user. By default, these are:

```
self.valid_wave_min = self.mask_good_wavelength_range[0]
self.valid_wave_max = self.mask_good_wavelength_range[1]
```

9.3 Correcting for the CCD defects (C)

If correct_ccd_defects=True, PyKOALA runs self.correct_ccd_defects():

```
self.correct_ccd_defects(odd_number=odd_number, remove_5577 = remove_5577_here,
                         fibre_p=fibre_p, apply_throughput=apply_throughput,verbose=verbose, plot=plot)
```

9.4 Fixing small wavelength shifts (W)

When `fix_wavelengths=True` or variable `sol` (an array with 3 numbers, that is the solution to the 2-degree fit for correcting wavelength) is given, PyKOALA runs `self.fix_2dfdr_wavelengths_edges()` (if `sol[0]=-1`), that uses only 2-4 emission lines in the edges of the red ccd, or `self.fix_2dfdr_wavelengths(sol=sol, plot=plot)` in any other case.

If `sol[2]=0` and `sol[0]` is not 0 or -1, `self.fix_2dfdr_wavelengths()` assumes this is the solution for the blue CCD using a linear fit to the skyline at 5577 Å.

9.5 Applying throughput 2D (T)

When `apply_throughput = True`:

```
519  self.apply_throughput_2D(throughput_2D=throughput_2D, throughput_2D_file=throughput_2D_file, plot=plot)
```

9.6 Correcting for extinction (X)

When `do_extinction = True`:

```
520  extinction_file = 'ssoextinct.dat'
521  self.do_extinction_curve(extinction_file, plot=plot)
```

9.7 Telluric correction (only for red data) (U)

When variables `telluric_correction` or `telluric_correction_file` are given for the red CCD,

```
522  w_star,telluric_correction = read_table(telluric_correction_file, ["f", "f"] )
523  self.telluric_correction = telluric_correction
524  self.intensity_corrected[i,:]= self.intensity_corrected[i,:] * telluric_correction
```

9.8 Sky subtraction (S)

Several options here following variable `sky_method`:

1. `"1D"` : Consider a single sky spectrum, scale it and substract it.
2. `"2D"` : Consider a 2D sky. i.e., a sky image, scale it and substract it fibre by fibre.
3. `"self"` : Obtain the sky spectrum using the `n_sky` lowest fibres in the `RSS` file (DEFAULT).
4. `"none"` : No sky subtraction is performed.
5. `"1Dfit"` : Using an external 1D sky spectrum, fits sky lines in both sky spectrum AND all the fibres.
6. `"selffit"` : Using the `n_sky` lowest fibres, obtain an sky spectrum, then fits sky lines in both sky spectrum AND all the fibres.

When variable `is_sky` is `True`, PyKOALA considers the `RSS` file is an OFFSET SKY and performs a median filter with window `win_sky` (that must be an odd number, the default value is 151) to increase the S/N.

9.9 Correcting negative values (N)

After sky subtraction, if `correct_negative_sky=True` and `is_sky=False`, PyKOALA runs `self.correcting_negative_sky()`.

9.10 Emission lines identification (E)

This needs to be checked, it's old...

9.11 Cleaning small CCD / sky residuals (R)

If `clean_sky_residuals = True` and `features_to_fix` is given, PyKOALA runs

```
525  fix_these_features_in_all_spectra(self,features=features_to_fix, #fibre_list=range(83,test.n_spectra),
526                                    sky_fibres = sky_fibres_for_residuals,
527                                    replace=True, plot=plot)
```

If `clean_ccd_residuals = True`, PyKOALA runs `self.intensity_corrected=clean_cosmics_and_extreme_negatives()`.

9.12 Last checks and plots

If any process has been performed, PyKOALA applies the mask, compute the integrated fibre, and (if `plot=True`) plots some corrected vs. uncorrected spectra (for highest intensity fibres and lowest intensity fibres) and the `RSS_image()`.

Finally, PyKOALA prints a summary and the basic information from the header and specifies the processes that have been performed.

9.13 Save processed RSS file if requested

If string variable `save_rss_to_fits_file` is given, PyKOALA saves the processed `RSS` file using

```
528  save_rss_fits(self, fits_file=save_rss_to_fits_file)}.
```

When the string variable is `"auto"`, the name of the output fits file is obtained following the function `name_keys`, that adds the appropriate letters to the input filename indicating what has been done, e.g. it adds `_TCWXUSENR` if throughput (T), correcting for CCD defects (C), small wavelength shifts (W), correction for extinction (X), telluric correction (U), sky subtraction (S), emission line identification (E), correction for negative values (N) and cleaning small CCD/sky residuals (R) have been applied.

10 ADDITIONAL FEATURES

PyKOALA can also be used to analyse the combined finished cube and produce images of flux, relative velocity and ..... in certain wavelengths???? what are the 3 types? .

comment in code below ???????

```
cubo = read_cube(cubo_file) # reading the cube
line = 4875. # WHAT IS THIS FOR AGAIN ????
hb_map=create_map(cubo, line)

""" Image of galaxy in ???? """
cubo.plot_map(mapa=hb_map[0],
              log=True,
              fraction=0.03525,
              plot_centre=False,
              vmin=100,
              vmax = 1E5,
              save_file="He2-10_Hb_flux.png")

""" Image of galaxy in Relative Velocity """
cubo.plot_map(mapa=hb_map[1],
              vmin=-100,
              vmax=100,
              cmap="seismic",
              barlabel="Relative velocity [km/s]",
              fraction=0.03525,
              plot_centre=False)

""" Image of galaxy in Relative Velocity ???? AGAIN ??? """
cubo.plot_map(mapa=hb_map[2],
              vmin=-20,
              vmax=10,
              cmap="seismic",
              barlabel="Relative velocity [km/s]", # NOTE IF WRONG NEED TO UPDATE IMAGE
              fraction=0.03525,
              plot_centre=False)
```

Code 2: Additional Features

Figures 14, Fig. 15 and Fig. 16 below are examples of executing Code 2.

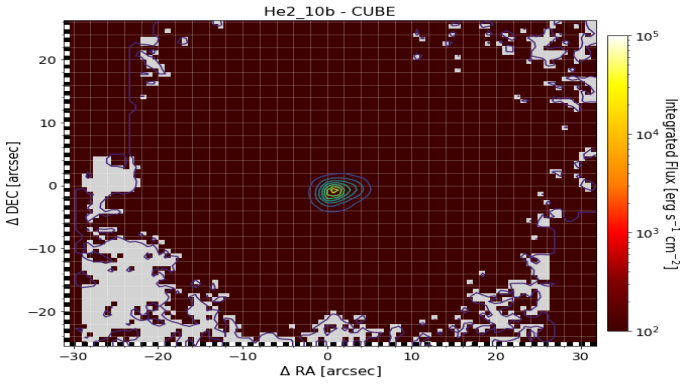

Figure 14: INSERT CAPTION

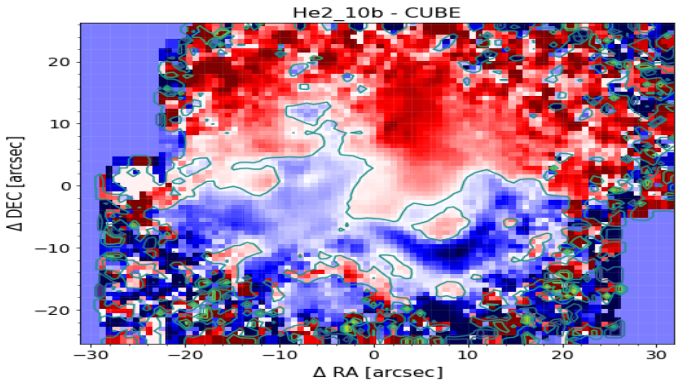

Figure 15: INSERT CAPTION velocity?

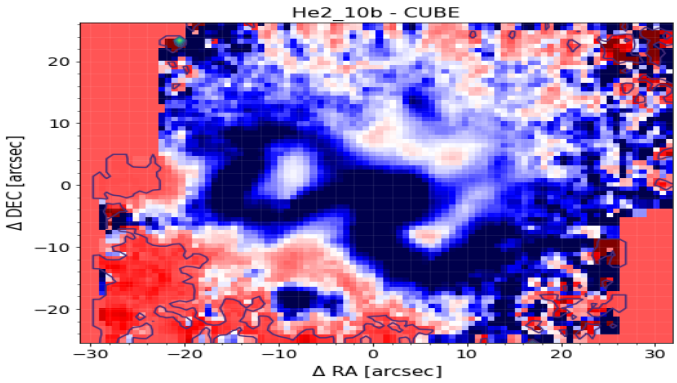

Figure 16: INSERT CAPTION

11 CONCLUSION

Congratulations, you have created a data cube ready for science!

You can use a software program like DS9 to see the reduced files or continue to use PyKOALA to plot (see section 10).

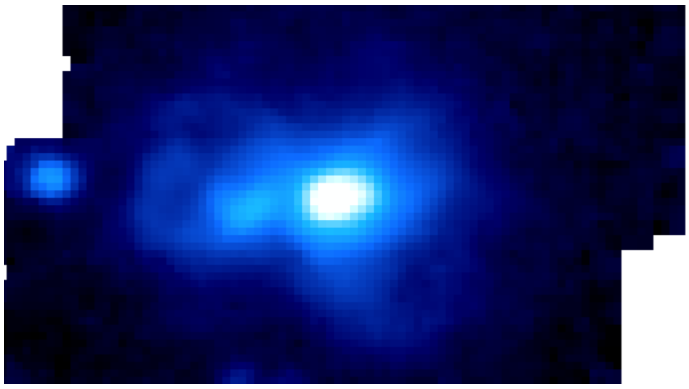

Figure 17: He2-10 galaxy in the blue spectrum using DS9 software

Note: All codes given are a guide and images are for display purposes.

ATTRIBUTES OF A RSS OBJECT IN PYKOALA

Types are f (float), s (string), l (list), t (table), a (array)

Attribute	Type	Description
airmass	f	Airmass of the observation given by (END-START)/2
ALIGNED_DEC_centre_deg	f	ALIGNED DEC Coordinates (in degrees) of the centre of the image
ALIGNED_RA_centre_deg	f	ALIGNED RA Coordinates (in degrees) of the centre of the image
CRVAL1_CDELT1_CRPIX1	[f, f, f]	[CRVAL1, CDTEL1, CRPIX1] from header
DEC_centre_deg	f	DEC Coordinates (in degrees) of the centre of the image
DEC_max	f	Maximum DEC Coordinates (in degrees) in the image
DEC_min	f	Minimum DEC Coordinates (in degrees) in the image
DEC_segment	f	Length of the DEC segment (in arcsec)
description	s	Description of the RSS file
el	l	List compiling the identified or given emission lines. Each element has the form [s, f, f, f]=[name, rest wave, obs. wave., FWHM]
exptime	f	Exposition time, in seconds, of the RSS observation
extinction correction	l[f, ..., f]	Correction for extinction per each wavelength
filename	s	Name of RSS file or path NO DOC **********
grating	s	Grating used in this observation
header2_data	l	All the info included in the HEADER 2 of the RSS file
history	l	NO DOC **********
history_RSS	l	List of what the file has been corrected for
ID	a	Array listing the GOOD fibres (986 for KOALA)
integrated_fibre	a	Array with the integrated fibre value of the GOOD fibres
integrated_fibre_sorted	a	Array of the fibres sorted by integrated intensity, i.e. [0] has the fibre with the FAINTEST spectrum and [-1] the fibre with the BRIGHTEST spectrum.
intensity	l[a][a]	Data of the RSS file: intensity[fibre][wavelength]
intensity_corrected	l[a][a]	Corrected data: intensity corrected[fibre][wavelength]
mask	l	mask (NO DOC **********)
mask_good_index_range	l	NO DOC **********
mask_good_wavelength_range	l	NO DOC **********
mask_list_fibres_all_good_values	l	NO DOC **********
n_spectra	f	Number of spectra (fibres) in the RSS file (986)
n_wave	f	Number of wavelengths in the RSS file (2048)
object	s	Name of the astronomical object observed in the RSS file
offset_DEC_arcsec	l[f, ..., f]	DEC offset (in arcsec) from the center of the image to the fibre
offset_RA_arcsec	l[f, ..., f]	RA offset (in arcsec) from the center of the image to the fibre
PA	f	Position angle (in degrees) of the KOALA IFU for this RSS file
RA_centre_deg	f	RA Coordinates (in degrees) of the centre of the image

RA_max	f	Maximum RA Coordinates (in degrees) in the image
RA_min	f	Minimum RA Coordinates (in degrees) in the image
RA_segment	f	Length of the RA segment (in arcsec)
RSS_list	l[s, ...]	List with RSS files used
sol	l	NO DOC **********
spaxel_size	f	Size of a KOALA lens (hexagon): 1.25 or 0.7 arcsec
valid_wave_max	f	Maximum value of the wavelength to be considered
valid_wave_max_index	f	NO DOC **********
valid_wave_min	f	Minimum value of the wavelength to be considered
valid_wave_min_index	f	NO DOC **********
variance	l[a][a]	Variance (error) of the data: variance[fibre][wavelength]
wavelength	l[f, ..., f]	List of wavelengths (2048 for KOALA)
ZDEND	f	Zenith distance at the end of the observation (in degrees)
ZDSTART	f	Zenith distance at the beginning of the observation (in degrees)

FUNCTIONS OF A RSS OBJECT IN PYKOALA.

Attribute	Description
apply_mask()	Apply a mask to a RSS file
apply_throughput_2D()	Apply throughput_2D using a variable or a fits file
clean_ccd_residuals()	Cleans CCD residuals in given spectra
compute_integrated_fibre()	Compute the integrated flux of a fibre in a particular range
correct_ccd_defects()	Replaces "nan" for median value, remove remove_5577 if requested
correcting_negative_sky()	Corrects negative sky with a median spectrum of the lowest intensity fibres
do_extinction_curve()	Corrects for the extinction curve given airmass and observatory file
find_sky_emission()	Finds the sky emission given fibre list or taking n_sky fibres with lowest integrated value.
find_sky_fibres()	Finds lowest intensity fibers in the array
fit_and_subtract_sky_spectrum()	Given a 1D sky spectrum, fits skylines of each fibre individually and subtracts sky
fix_2dfdr_wavelengths	Using bright skylines(), performs small wavelength corrections to each fibre
fix_2dfdr_wavelengths_edges()	Using bright skylines, performs small wavelength corrections to each fibre
flux_between()	Gives the integrated flux in the range [l min,l max]
get_mask()	Task for getting the mask using the very same RSS file. This assumes that the RSS does not have nans or 0 as consequence of cosmics in the edges. The task is run once at the very beginning, before applying flat or throughput. It provides the self.mask data
get_telluric_correction()	Obtains telluric correction using a spectrophotometric star
identify_el()	Identify fibres with highest intensity (high fibres=10). Add all in a single spectrum. Identify emission features.
median_between()	Provides the median in the range [l min,l max]
plot_combined spectrum()	Plot combined spectrum of a list and return the combined spectrum.
plot_corrected_vs_uncorrected_spectrum()	Compares corrected vs uncorrected spectra in a plot.
plot_spectra()	Plots each spectrum of a list of spaxels.
plot_spectrum()	Plots spectrum of a particular spaxel.
read_mask_from_fits_file()	This task reads a fits file containing a full mask and save it as self.mask. Note that this mask is an IMAGE, the default values for self.mask,following the task 'get_mask' below, are two vectors with the left -self.mask[0]- and right -self.mask[1]- valid pixels of the RSS. This takes more memory & time to process.
RSS_image()	Plot RSS image coloured by variable
RSS_map()	Plot map showing the offsets, coloured by variable
apply_extinction_curve()	Apply the derived extinction curve using airmass.
apply_telluric_correction()	Apply telluric correction using data from a sample star
apply_self_sky()	Applies sky correction using method "self"
apply_1D_sky()	Applies sky correction using method "1D"
apply_1Dfit_sky()	Applies sky correction using method "1Dfit"
subtract_sky()	subtracts the sky stored in self.sky_emission to all fibres in self.intensity_corrected

ATTRIBUTES OF A CUBE IN PYKOALA

Types are f (float), s (string), l (list), t (table), a (array)

Attribute	Type	Description
ADR_total	f	NO DOC **********
ADR_total_NEW	f	NO DOC **********
ADR_x	a	NO DOC **********
ADR_x_NEW	a	NO DOC **********
ADR_x_fit	a	NO DOC **********
ADR_x_max	f	NO DOC **********
ADR_x_max_NEW	f	NO DOC **********
ADR_y	a	NO DOC **********
ADR_y_NEW	a	NO DOC **********
ADR_y_fit	a	NO DOC **********
ADR_y_max	f	NO DOC **********
ADR_y_max_NEW	f	NO DOC **********
CRVAL1_CDELT1_CRPIX1	[f, f, f]	NO DOC **********
data	a	Data of ? NO DOC **********
data_no_ADR	a	Data without ADR? NO DOC **********
description	s	Description of object (NO DOC **********)
DEC_centre_deg	f	NO DOC **********
DEC_segment	f	NO DOC **********
exptimes	l	Time each individual file took to process
flux_cal_max_wave	f	NO DOC **********
flux_cal_min_wave	f	NO DOC **********
flux_cal_step	f	NO DOC **********
flux_calibration	a	NO DOC **********
grating	s	Grating used (NO DOC **********)
integrated_map	a	NO DOC **********
integrated_map_all	a	NO DOC **********
integrated_star_flux	a	NO DOC **********
kernel_size_arcsec	f	Kernel size in arc seconds
kernel_size_pixels	f	Kernel size in pixels
max_x	f	NO DOC **********
max_y	f	NO DOC **********
n_cols	f	Number of columns (NO DOC **********)
n_rows	f	Number of columns (NO DOC **********)
n_wave	f	NO DOC **********
number_of_combined_files	f	Number of combined files (NO DOC **********)
object	s	Object created (NO DOC **********)
offset_from_center_x_arcsec_integrated	f	NO DOC **********
offset_from_center_x_arcsec_tracing	f	NO DOC **********
offset_from_center_y_arcsec_integrated	f	NO DOC **********
offset_from_center_y_arcsec_tracing	f	NO DOC **********
offsets_files	l	NO DOC **********
offsets_files_position	s	NO DOC **********
PA	f	NO DOC **********
pixel_size_arcsec	f	(NO DOC **********) Pixel size in arc seconds.
RA_centre_deg	f	NO DOC **********

RA_segment	f	NO DOC **********
rss_files	l	Name of RSS files used
seeing	f	NO DOC **********
spaxel_DEC0	f	NO DOC **********
spaxel_RA0	f	NO DOC **********
total_exptime	f	Total execution time
valid_wave_max	f	NO DOC **********
valid_wave_min	f	NO DOC **********
wavelength	a	NO DOC **********
weight	a	NO DOC **********
weight_no_ADR	a	NO DOC **********
weighted_I	a	NO DOC **********
weighted_I_no_ADR	a	NO DOC **********
x_peak	a	NO DOC **********
x_peak_median	f	NO DOC **********
x_peak_median_index	f	Index of median x peak (NO DOC **********)
y_peak	a	NO DOC **********
y_peak_median	f	NO DOC **********
y_peak_median_index	f	Index of median y peak (NO DOC **********)

FUNCTIONS OF A CUBE IN PYKOALA

Attribute	Description
ADR_correction()	Corrects for Atmospheric Differential Refraction (ADR)
add_spectrum_ADR()	Add one single spectrum to the datacube
apply_flux_calibration()	Function for applying the flux calibration to a cube
box_for_centroid()	Creates a box around centroid
build_cube()	This function builds the cube
create_map()	NO DOC ********** Creates Map
do_response_curve()	Compute the response curve of a spectrophotometric star
fit_Moffat_between()	Method of Interpolated_cube instance (NO DOC **********)
get_integrated_map()	Compute integrated map
get_integrated_map_and_plot()	Integrated map and plot
growth_curve_between()	Compute growth curve in a wavelength range. Returns r2_growth_curve, F_growth_curve, flux, r2_half_light
half_light_spectrum()	Compute half light spectrum (for r_max=1) or integrated star spectrum (for r_max=5) in a wavelength range.
mask_cube()	This mask is used to allow spaxels with some or full spectrum through
plot_map()	Show a given map.
plot_spectrum_cube()	Plot spectrum of a particular spaxel or list of spaxels.
plot_wavelength()	Plot map at a particular wavelength or in a wavelength range.
plot_weight()	Plot weight map. TYPO IN DOC (Plot weitgh map.") line 4668
spectrum_of_box()	Given a box (or a center with a width size, all in spaxels), this task provides the integrated or median spectrum and the list of spaxels.
spectrum_offcenter()	This tasks calculates spaxels coordenates offcenter of the peak, and calls task spectrum_of_box to get the integrated or median spectrum of the region.
trace_peak()	NO DOC **********
trim_cube()	Task for trimming cubes in RA and DEC (not in wavelength)

A APPENDIX A

This is the information provided in the `.config` file `calibration_night.config` needed for running the automatic script `automatic_calibration_night()`.

```
#####################################################
# CONFIGURATION FILE FOR RUNNING AUTOMATICALLY      #
# THE REDUCTION OF A CALIBRATION STAR WITH PyKOALA #
#####################################################

date                    20180227
grating                 385R
pixel_size              0.7
kernel_size             1.4

path                    /DATA/KOALA/Jamila/20180227/385R/   # If a full path is not provided for files,
                                                            # use this as the path

# Skyflat section ------------------------------------------------------

file_skyflat            combined_skyflat_red.fits           # Fits file with the RSS skyflat
skyflat                 skyflat_385R                        # Python object with skyflat information

do_skyflat              True

correct_ccd_defects     True
fix_wavelengths         True
sol                     [0.06039,-0.0010,2.2712e-07]        # 2nd order polynomium to fix small wavelength shifts
rss_star_file_for_sol   27feb20028red.fits                  # RSS file of a star to get values for sol

throughput_2D_file      throughput_2D_20180227_385R.fits    # Fits file where the 2D throughput will be saved
throughput_2D           throughput_2D_20180227_385R         # Python variable where 2D throughput will be saved

# Calibration stars section ---------------------------------------------

CONFIG_FILE_path        ./CONFIG_FILES/Jamila/              # Path to folder with the .config files
CONFIG_FILE             calibration_EG274_red.config        # Each of the .config files with calibration stars
CONFIG_FILE             calibration_Hilt600_red.config

object                  EG274r                              # Python objects to be created/read for calibration stars
object                  Hilt600r

abs_flux_scale          [1,1]                               # For scaling the absolute flux of the stars

flux_calibration_file           flux_calibration_20180227_385R_0p7_1k40.dat     # To be created
telluric_correction_file        telluric_correction_20180227_385R.dat           # To be created


cal_from_calibrated_starcubes   False                       # It only reads the provided Python objects with fully
                                                            # processed starcubes and gives combined calibration

# Plotting, printing ----------------------------------------------------

plot                    True                                # Plotting figures? (if False PyKOALA runs faster)
fig_size                12.                                 # Size of the figures (default = 12)
warnings                True                                # Print warnings (default = True)
verbose                 True                                # Print what is happening (default = True).
                                                            # Setting verbose = False makes PyKOALA run faster.
```

B APPENDIX B

This is the information provided in the `.config` file `calibration_Hilt600_red.config` needed for running `run_automatic_star()`.

```
#######################################################
# CONFIGURATION FILE FOR RUNNING AUTOMATICALLY        #
# THE REDUCTION OF A CALIBRATION STAR WITH PyKOALA    #
#######################################################

date                    20180227
grating                 385R
pixel_size              0.7
kernel_size             1.4

# Files ------------------------------------------------------------------------------------------

path_star               /DATA/KOALA/Jamila/20180227/385R/
obj_name                Hilt600r                                # Python object to be created / read
star                    Hilt600                                 # Name of the star
# description           Hilt600r                                # Description
# absolute_flux_file    ./FLUX_CAL/fhilt600_edited.dat          # Many of these are in ./FLUX_CAL/ & included in code

fits_file               Hilt600_385R_0p7_1k40.fits              # fits file to be created with the CUBE
response_file           Hilt600_385R_0p7_1k40_response.dat
telluric_file           Hilt600_385R_0p7_1k40_telluric_correction.dat

rss                     27feb20028red.fits                      # Each of the RSS files for this star
rss                     27feb20029red.fits
rss                     27feb20030red.fits

reduced                 False           # (default = False) If True be sure Python object has been created, it will
                                        #  not read RSS or cubing, it will go directly to calibration
read_cube               False           # (default = False) If True it reads the fits_file as a cube,
                                        # it creates object with name "obj_name" and jump to calibration

# RSS Section ------------------------------------------------------------------------------------

rss_clean               False           # Are the provided RSS clean? If so it just does the cubing, ignoring all parameters on
save_rss                True            # Select "True" for saving the obtained "CLEANED" RSS files

apply_throughput        True
throughput_2D_file      throughput_2D_20180227_385R.fits        # Fits file with the 2D throughput,
                                                                # if throughput_2D given it ignores this
#throughput_2D          throughput_2D_20180227_385R             # Python variable with the 2D throughput

correct_ccd_defects     True                    # Correct for CCD defects (nan, inf, and negative values)
fix_wavelengths         True
#sol                    [-1]                    # If sol=[-1], it will only check 3 skylines in edges (faster)
sol                     [0.06038947169959769,-0.0010064562304288275,2.2712447657371968e-07]
do_extinction           True
sky_method              self                    # Method applied for subtracting the sky. Options: none, self, 1D, 2D, 1Dfit,
                                                # and selffit. Almost always "self" for stars
n_sky                   50                      # Number of fibres used to create a sky spectrum (default = 100)
sky_fibres              range(0,55)
#win_sky                25                      # Kernel size of the median filter for smoothing in fibre direction before getti
                                                # Must be an integer odd number. Default 151. Only for 2D sky
#sky_lines_file         sky_lines_test.dat      # File with the sky lines to me removed (if using 1Dfit or selffit)
#remove_5577            True                    # Only valid for the BLUE ccd if skyline 5577 is observed

correct_negative_sky            True            # Combines lowest intensity fibres, fits continuum, and check that
                                                # no negative integrated values exists. If they are, it adds fits/smooth
                                                # to make continuum of lowest intensity fibres = 0
order_fit_negative_sky          3
kernel_negative_sky             51
individual_check                True
use_fit_for_negative_sky        False
force_sky_fibres_to_zero        True
low_fibres                      10
#high_fibres                    20

#clean_cosmics                  False           # (default = False) Careful here when using this
#only_positive_cosmics          True
#disp_scale                     1.0
#disp_to_sqrt_scale             1.6
#ps_min                         3.
#width                          20.

#remove_negative_pixels_in_sky  False
#clean_extreme_negatives        False           # if True it replaces extreme negative (lower than percentile_min)
#percentile_min                 0.9             # by median value of fibre
```

```
77
78  # Cubing Section ------------------------------------------------------------------------------------------------------
79
80  #size_arcsec            [46.5,24.5]             # Size of the cube
81  edgelow                 5                       # For tracing, in waves from w[0]
82  edgehigh                5                       # For tracing, in waves to w[-1]
83  ADR                     True
84  jump                    -1                      # For ADR, default = -1 (automatic). For quick processing, try jump = 200.
85  trim_cube               True
86  #trim_values            [2,75,2,45]             # Trim data [x0,x1,y0,y1] in spaxels
87
88  scale_cubes_using_integflux     False
89  #flux_ratios                    [0.96,0.99,1]
90
91  # Calibration ----------------------------------------------------------------------------------------------------------
92
93  #r_max                  5          # Max radius, in half-light-radius, to get flux of star
94  sky_annulus_low_arcsec  5.         # Minimum radius in arcsec of the annulus for the sky subtraction (default = 5.)
95  sky_annulus_high_arcsec 10.        # Maximum radius in arcsec of the annulus for the sky subtraction (default = 10.)
96
97  #order_telluric         2
98  #telluric_range         [6150,6240,6410,6490]
99  #telluric_range         [6720,6855,7080,7140]
100 #telluric_range         [7080,7140,7500,7580]
101 #telluric_range         [7400,7580,7705,7850]
102 #telluric_range         [7850,8090,8450,8700]
103
104 apply_tc                True    # CAREFUL WITH THIS! Default = False. Only use True when SURE about the telluric correction.
105                                 #                    Otherwise CUBING must be repeated if something wrong.
106
107 #min_wave_flux          6150
108 #max_wave_flux          9250
109 step_flux               10
110 ha_width                150                     # To avoid the H-alpha absorption line
111 exclude_wlm             [[7200,7450],[7700,8000]]
112 fit_degree_flux         5
113 #odd_number             13
114 #smooth                 0.05
115 #fit_weight             0.5
116 #smooth_weight          0.0
117
118 # Plotting, printing ---------------------------------------------------------------------------------------------------
119
120 #valid_wave_min         3650                    # These two parameters are automatically computed using the automatic mask
121 #valid_wave_max         5700
122
123 plot                    True
124 plot_rss                True
125 plot_weight             False
126 fig_size                12.                     # Default: 12
127 norm                    colors.LogNorm()        # Options: default: colors.LogNorm() [LOG], colors.Normalize() [LINEAL],
128                                                 #          colors.PowerNorm(gamma=0.25)  [sometimes gamma=0.5]
129 warnings                False
130 verbose                 True
```

C APPENDIX C

This is the information provided in the `.config` file `koala_reduce_20180227_He2-10_red.config` needed for running `automatic_KOALA_`

```
#######################################################
# CONFIGURATION FILE FOR RUNNING AUTOMATICALLY        #
# THE REDUCTION OF A CALIBRATION STAR WITH PyKOALA    #
#######################################################

date                          20180227              # Date of the observations. Must be item [0] here
grating                       385R                  # Grating used. Must be item [1] here.
pixel_size                    0.7                   # Pixel size in arcsec. Must be item [2] here
kernel_size                   1.4                   # Kernel size in arcsec. Must be item [3] here

# Files ------------------------------------------------------------------------------------------

path                          ../20180227/385R/                                    # Default path. Must be item [4]
obj_name                      He2_10Ar                                             # Name of the ob
description                   He2_10Ar                                             # By default, it
Python_obj_name               He2_10Ar                                             # Name of the Py


flux_calibration_file         ../20180227/385R/Hilt600_385R_0p7_1k40_response.dat     #change Hilt600 -> ...
telluric_correction_file      ../20180227/385R/Hilt600_385R_0p7_1k40_telluric_correction.dat


fits_file                     ../20180227/385R/He2_10Ar_385R.fits                  # Fits file where the final comb

rss_file                      27feb20031red.fits
rss_file                      27feb20032red.fits
rss_file                      27feb20033red.fits


# RSS Section ------------------------------------------------------------------------------------

rss_clean                     True                                                 # Are the provided RSS c
save_rss                      True                                                 # Select "True" for savi

apply_throughput              True
throughput_2D_file            ../20180227/385R/throughput_2D_385R.fits

correct_ccd_defects           True                                                 # Correct for CCD defect
fix_wavelengths               True                                                 # Fixes small wavelength
sol                           [0.08532918197224319,-0.0009925103155243756,1.582523076135417e-07]     # Solution to the 2-degr

do_extinction                 True                                                 # Applies the correction

do_telluric_correction        True


sky_method                    selffit                                              # Method applied for sub
#remove_5577                  True                                                 # Only for the blue arm,
#sky_spectrum                 [0]                                                  # Object with sky spectru


#sky_file                     skyt7d6                                              # Each of the files to b
#sky_file                     sky2.fits                                            # They have to be in the

sky_lines_file                sky_lines_IR_jamila.dat                                                  # Option

n_sky                         30                  # Number of fibres used to create a sky spectrum (default = 50)
#win_sky                      25                  # Kernel size of the median filter for smoothing in fibre direction befo

#scale_sky_1D                 1.                  # It should be a list for each sky... TO DO
#auto_scale_sky               False               # CHECK THIS, perhaps remove
correct_negative_sky          True                # After all corrections are applied, combines lowest intensity fibres, f
                                                  # no negative values exists. If they are, add continuum to make continuu
brightest_line                Ha                  # Brightest line to be used as reference. This is also needed for 1Dfit
brightest_line_wavelength     6581.               # Observed wavelength of the brightest line. This is also needed for 1Df


#id_el                        False               # Automatically identifies emission lines given the following parameters
#high_fibres                  10                  # Number of highest intensity fibres to combine for searching emission l
#cut                          1.5                 # Cut limit FLUX/CONTINUUM to identify emission lines
#broad                        1.8                 # Broad (FWHM) of the expected emission lines
id_list                [6300.30,6312.1,6363.78,6548.03,6562.82,6583.41,6678.15,6716.47,6730.85,7065.28,7135.78,7318.39,7329.66,
#plot_id_el                   False

clean_cosmics                 False
only_positive_cosmics         True
disp_scale                    1.0
disp_to_sqrt_scale            1.6
```

```
ps_min                          3.
width                           20.

clean_extreme_negatives         False
percentile_min                  0.9


# Cubing Section ------------------------------------------------------------------------------------------------

do_cubing                       True
#do_alignment                   True
#make_combined_cube             True
scale_cubes_using_integflux     True

size_arcsec                      [75,70]

#offsets                        [-3.75,-2.16]                                   # Each of the offsets to be applied betw
#offsets                        [3.75,0]                                        # If they are not given, PyKOALA will co
#offsets                        [0,-4.32]                                       # If given, must be (rss files) - 1 or P
#offsets                        [0,-6.48]
#offsets                        [3.75,0]                                        # EAST+/WEST-  NORTH-/SOUTH+  [1->2E/W,1-

offsets                         [-0.1521712288645949,1.5284432843655793]
offsets                         [3.0779674604339213,1.4719641163703265]
offsets                         [3.5,18.9]
offsets                         [2.1443607139233833,-0.00038260303269410656]
offsets                         [0.8657601723259376,1.4756563842921402]

edgelow                         5                                               # For tracing, in waves from w[0]
edgehigh                        10                                              # For tracing, in waves to w[-1]

ADR                             True                                            # Compute ADR in each of the cubes.
ADR_cc                          False                                           # Compute ADR correction in the combined
#ADR_x_fit                      [-3.27296592e-08,1.79582155e-04,-1.16392805e-01]  # Values for the ADR correction in RA,
#ADR_y_fit                      [1.79815815e-07,-1.82837421e-03,4.59452508e+00]  # Values for the ADR correction in Dec,
jump                            -1                                              # For ADR, default = -1 (wave by wave).

trim_cube                       True                                            # Trim the combined cube to get only val
half_size_for_centroid          8
#trim_values                    [2,75,2,45]                                     # Trim data [x0,x1,y0,y1] in spaxels. If
#box_x                          [30,40]                                         # RA  segment of the box used for doing
#box_y                          [15,24]                                         # Dec segment of the box used for doing

save_aligned_cubes              True                                            # Save aligned cubes as individual .fits

# Plotting, printing ------------------------------------------------------------------------------------------

#valid_wave_min                 3650                # Minimum valid wavelength. By default, PyKOALA will use the mask of bad
#valid_wave_max                 5700                # Maximum valid wavelength. By default, PyKOALA will use the mask of bad
plot                            True                # Plot information plots? (default = True). Setting plot=False makes PyK
plot_rss                        True                # Plot rss images (default = True)
plot_weight                     False               # Plot weight maps for the cubes (default = False)
fig_size                        12.                 # Size of the figures (default = 12)
norm                            colors.LogNorm()    # Options: default: colors.LogNorm() [LOG],  colors.Normalize() [LINEAL]
warnings                        True                # Print warnings (default = True)
verbose                         True                # Print what is happening (default = True).  Setting verbose=False makes

# -------------------------------------------------------------------------------------------------------------
```