KOSMOS MLTI-MLTI-SLIT REDUCTION TUTORIAL

Abdullah Korra

[1]*The College of Idaho, Physics, Caldwell ID, United States.*
[2]*Western Washington University, Physics, Bellingham WA, United States.*
[3]*Saint Martin's University, Physics, Lacey WA, United States*

A Multi-slit Spectroscopy Data Reduction Guide written by undergraduates, for undergraduates,
or anyone else interested in performing Multi-slit reduction
for the ARC 3.5m Telescope equipped with KOSMOS
(Kitt Peak Ohio State Multi-Object Spectrograph) instrument.

October 26, 2024

DRAFT VERSION OCTOBER 26, 2024
Typeset using LaTeX **preprint** style in AASTeX631

Contents

1. PREFACE

The purpose of this document is to guide a KOSMOS user through a spectroscopy data reduction, specifically the **multi-slit** reduction process. It assumes that the user is familiar with Python coding, but does NOT assume the user has reduced astronomical observations before – imagine an undergraduate student starting astronomical research for the first time. The codes used in this reduction are based on the Pyvista astronomical Python packages built for spectroscopy reduction by Jon Holtzman. For more information, see the resources (2). **Throughout this manual, I will use the word "user" to refer to anyone who uses the pipeline.**

2. RESOURCES

- KOSMOS wiki for more information about the instrument and its usage: https://www.apo.nmsu.edu/arc35m/Instruments/KOSMOS/userguide.html
- Pyvista GitHub: https://github.com/holtzmanjon/pyvista
- Github repository with example data and notebooks (CofI_Abdullah_2024): https://github.com/mwbest/CofI_Abdullah_2024.git

3. SOFTWARE SETUP

This reduction is meant to run on a Jupyter Notebook because of Pyvista's interactive image visualization. However, it can also run in Google Colab with a few modifications.

- Install Python on your local computer, including the following libraries: numpy, matplotlib, astropy, pandas, photutils, ccdproc.
- Install Git on your local computer. The instructions for this installation can be found here https://github.com/git-guides/install-git.
- To install pyvista (astro-pyvista) follow the link, it will tell you all you need to know about its installation https://pyvista.readthedocs.io/en/latest/installation.html. For quick installation, run the following on the command line:

```
pip install astro-pyvista
```

- Through the command line, clone CofI_Abdullah_2024 on your device (https://github.com/mwbest/CofI_Abdullah_2024.git).

```
git clone https://github.com/mwbest/CofI_Abdullah_2024.git
```

  Our GitHub repository CofI_Abdullah_2024 includes:

  – A notebook of multi-slit reduction of the KOSMOS. Steps each step explained by this workflow. Notebook name is ('CofI KOSMOS slitmask reduction final draft.ipynb'); a link to the noteboonk https://github.com/mwbest/CofI_Abdullah_2024/blob/main/CofI%20KOSMOS%20slitmask%20reduction%20final%20draft.ipynb

- two half nights of KOSMOS data (in the 'UT230908' and 'UT230909' folders) and their associated input mask files (in the 'kms' folder), to be used by the workflow notebook and other tests; and
- a set of wavelength reference files in "new_wave_lamp" folder, optimized for the specific spectrograph configuration and set of lamp observations taken during the half nights included above: other users may need to update these files, potentially using the extensive library of KOSMOS arc lamps and line lists curated by James Davenport at https://github.com/jradavenport/kosmos-arc.

Running the notebook

Upon cloning CofI_Abdullah_2024, the clone folder will have the sample data and the tutorial notebook described above. To open the notebook in Jupyter, open the directory CofI_Abdullah_2024 in a terminal and write "jupyter notebook." A window named home should pop up in your default browser. In the home tab, you can either run a new notebook and do your reduction following the manual or you can run the existing tutorial notebook named "workflow_draft.ipynb" for a trial run.

4. MULTI-SLIT IMAGE

Before introducing what a multi-slit image looks like, let's define several key terms:

Astronomical image: An astronomical image is a visual representation of data collected from observations of celestial objects and phenomena. These images are two-dimensional arrays of values. With the telescope shutters opened, it gathers light through the camera pixels. Ideally, each pixel should represent an element of an array whose value corresponds precisely and exactly to the amount of light source detected from an astronomical object at that position on the sky. Unfortunately, that is not the case due to the telescope's fundamental preset bias characteristics and some celestial interference. Thus, we have to reduce the images to remove these effects and accurately measure the light received from the celestial object being observed.

FITS Files: Astronomical images are often stored in Flexible Image Transport System (FITS) files, which include both the image data and metadata about the observation, such as the time, location, date, instrument settings, and calibration information.

Spectroscopy: a measurement technique based on the principle that different substances absorb and emit light at specific wavelengths. By examining these wavelengths, astronomers can identify and quantify the elements and molecules present in a sample.

Spectroscopy in Astronomy: Spectroscopy is crucial in astronomy for analyzing celestial objects' composition, temperature, density, and motion. It helps in studying stars, galaxies, and interstellar matter.

Single Slit and Multi-slit Spectroscopy: At the Apache Point Observatory, the ARC 3.5 meter telescope uses the KOSMOS (Kitt Peak Ohio State Multi-Object Spectrograph) instrument to collect light from objects projected onto one or more slits. That light gets dispersed through a grism disperser to spread the source's light out as a function of wavelength. **Single slit** observations are simpler to acquire and reduce, but only allow the observation of one target at a time. Alternatively, one can observe with a 'slit mask'

containing multiple slits, placed carefully to enable the collection of spectra from multiple objects at a time. These masks must be aligned carefully during observations to ensure all the stars fall on the intended slits, but once they do, we can collect data for multiple stars at once. Thus, **multislit** reduction (see Figure 1).

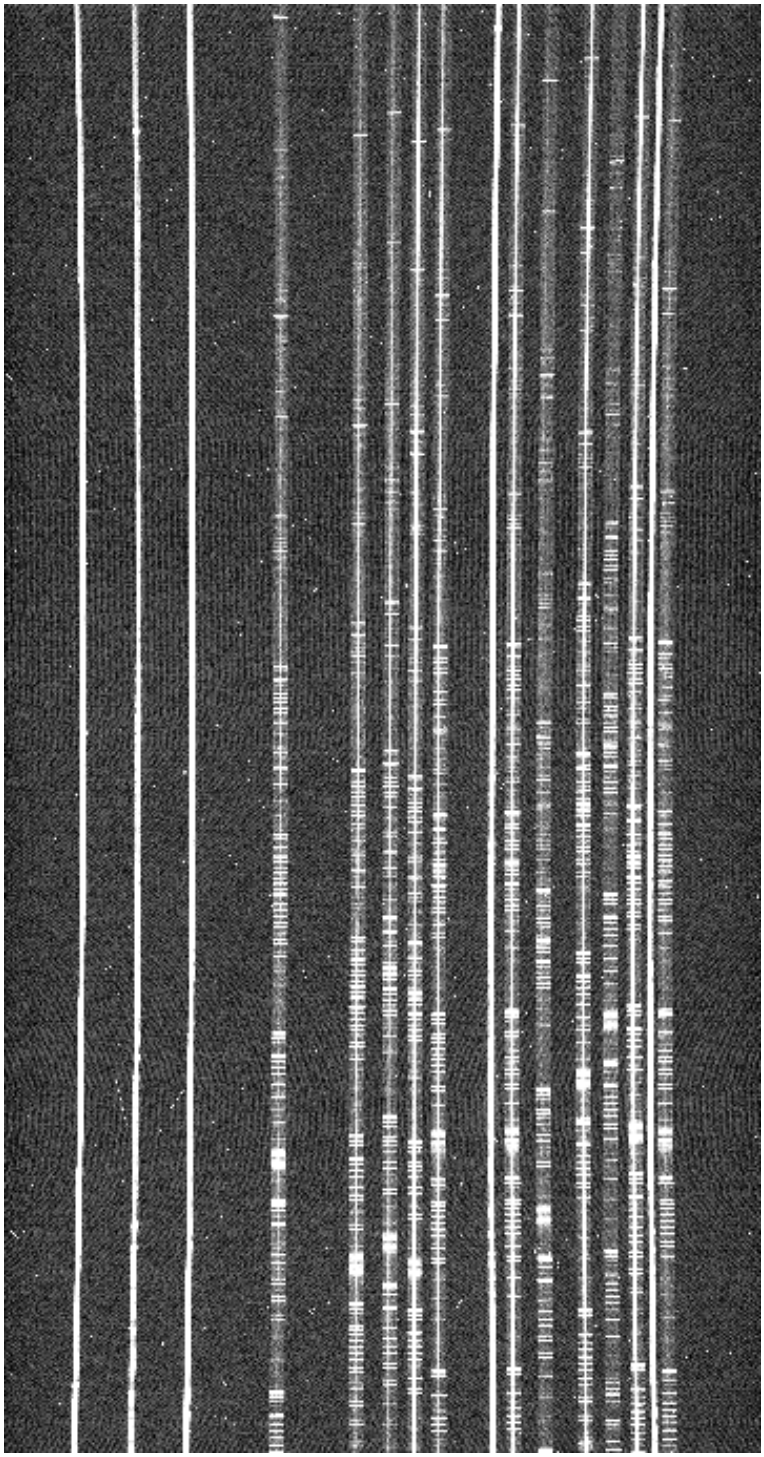


Figure 1: This is what a raw multi-slit image looks like for KOSMOS, with the positional axis oriented horizontally and the wavelength axis oriented vertically.

5. NECESSARY MODULES

After cloning the CofI_Abdullah_2024 repository from https://github.com/mwbest/CofI_Abdullah_2024.git, you can run the notebooks as you wish. Users can change the directories accordingly if they plan to reduce files containing their own observations. In the case of running your notebook, make sure to run the following cells before anything else:

```
!pip install scikit-image
!pip install scikit-learn
!pip install photutils
!pip install specutils
!pip install ccdproc
!pip install PyQt6
!pip install PySide6
!pip install PyQt5
```

```
# Import the following:
from pyvista import imred, tv, spectra, stars, slitmask, image
import numpy as np
import pdb
import copy
import matplotlib.pyplot as plt
import os
from astropy.table import vstack
import pandas as pd
from astropy.io import fits
```

Upon finishing the above step, the user will be ready to dive into image reduction. Thus, it is necessary to know how to attach your files to the notebook and how to call upon them when needed.

6. DIRECTORY SETUP

The code below will attach the users' directory to the notebook.

```
# put directory name with images here
indir='UT230909'
red=imred.Reducer('KOSMOS',dir=indir)
```

Output

```
INSTRUMENT: KOSMOS   config:
  will use format:  UT230909/*{:04d}.f*.fits*
         gain:  [0.6]    rn: [5.]
         scale:  None
  Biastype : 1
  Bias box:
    SC    NC    SR    NR
  2055    43    20  4056
  2105    43    20  4056
  Trim box:
    SC    NC    SR    NR
     0  2048     0  4096
     0  2048     0  4096
  Norm box:
    SC    NC    SR    NR
  1000    51  2000    51
```

Figure 2: The imred.Reducer command will output the parameters it will use to reduce images taken with the instrument specified, in our case KOSMOS

The 'red.log()' command in the next cell outputs a table containing all the files in the specified directory.
Run the cell below

```
red.log().show_in_notebook(display_length=40)
```

Output

Table length=90

idx	FILE	DATE-OBS	OBJNAME	RA	DEC	EXPTIME
0	Flat_SEG3G2.0001.fits	2023-09-09T01:31:47.843412		6:56:00.00	75:00:00.00	2.0
1	Flat_SEG3G2.0002.fits	2023-09-09T01:33:24.544267		6:56:00.00	75:00:00.00	1.0
2	Flat_SEG3G2.0003.fits	2023-09-09T01:34:47.647398		6:56:00.00	75:00:00.00	0.5
3	Flat_SEG3R2.0004.fits	2023-09-09T01:38:24.858222		6:56:00.00	75:00:00.00	0.5
4	Flat_EMPTY_TEST.0005.fits	2023-09-09T01:40:21.135840		6:56:00.00	75:00:00.00	0.5
5	Flat_SEG3R1.0006.fits	2023-09-09T01:43:41.792202		6:56:00.00	75:00:00.00	0.5
6	Flat_SEG3R1.0008.fits	2023-09-09T01:47:35.037138		6:56:00.00	75:00:00.00	0.5
7	Flat_SEG3G2.0009.fits	2023-09-09T01:49:32.885458		6:56:00.00	75:00:00.00	0.5

Figure 3: Table of files

Note: If one wishes to run a pyVista command on a given file, for example: "Flat_SEG3G2.0001.fits," (see figure 3) you often do not have to give the full file name as an input; you usually only need to give the file number as the input – in this case, one (1).

This pipeline is meant to work best on jupyter notebooks. Thus, it is recommended that you first define your image window, and the code below will allow the User to do so.

Run the cell below

```
%matplotlib qt
t=tv.TV()
plotinter=True
```

This code will pop a window outside the notebook. You can interact with your images there as you wish (see figure 4).

Output

Figure 4: The main display window should not be closed, but adjacent windows can be. If the main window is closed, cells requiring it will produce an error. To fix this, re-run the cell that defines the main window.

7. CALIBRATION IMAGES

Astronomers use calibration images to correct for things like readout noise (offset noise), thermal noise, electric noise, and uneven brightness in science images. Calibration images are, **Bias, Dark Flat, arc lamps**.

7.1. *Bias*

Bias is an offset voltage value added to the pixels of a telescope camera to ensure that when images are taken, they do not have a negative value. At the same time, the initial brightness of the pixels affects science

images; therefore, they need to be subtracted. A bias image (see figure 5) is taken in the shortest time the telescope (instrument) can with the shutter closed. The shortest possible time for the KOSMOS instrument is zero seconds.

To subtract any calibration image from our raw science images, we need a master or super bias, darks, and flats. We also take into account something called overscan. The Overscan region is a part of the camera's detector that is covered and receives no light from an observing source. However, the chip(s) will register as if they detected a certain amount of light due to the offset value. Thus, the level of this offset must be removed from all calibration and science images. Overscan removal happens in the background in this pipeline.

The code below will only remove the overscan from the bias image and display the image (see Figure 5).

Run the cell below

```
bi = red.reduce(74)
if t is not None:
    t.tv(bi)
```

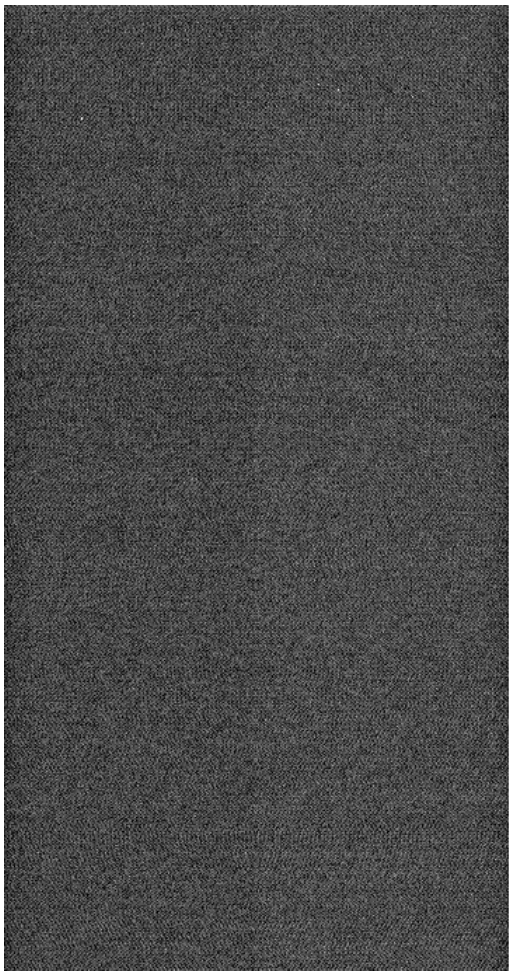


Figure 5: A single bias image.

7.1.1. *Master bias*

A master bias (super bias) combines all the biases to reduce the noise in the measured bias level. This ensures that as bias gets subtracted from the science image, they are not subtracting anything else except the bias level. **The following cells create a master bias and plot it.**

Run the cell below

```
biastims=[74,75,76,77,78] # your file numbers
bias=red.mkbias(biastims,display=None) # makes a master bias
```

Run the cell below

```
# Visulize the master bias
if t is not None:
    t.tv(bias)
```

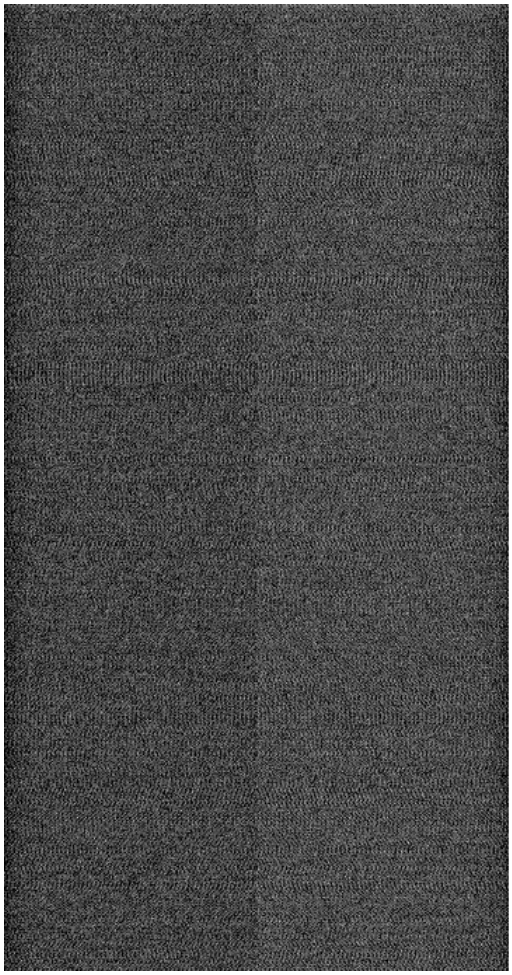
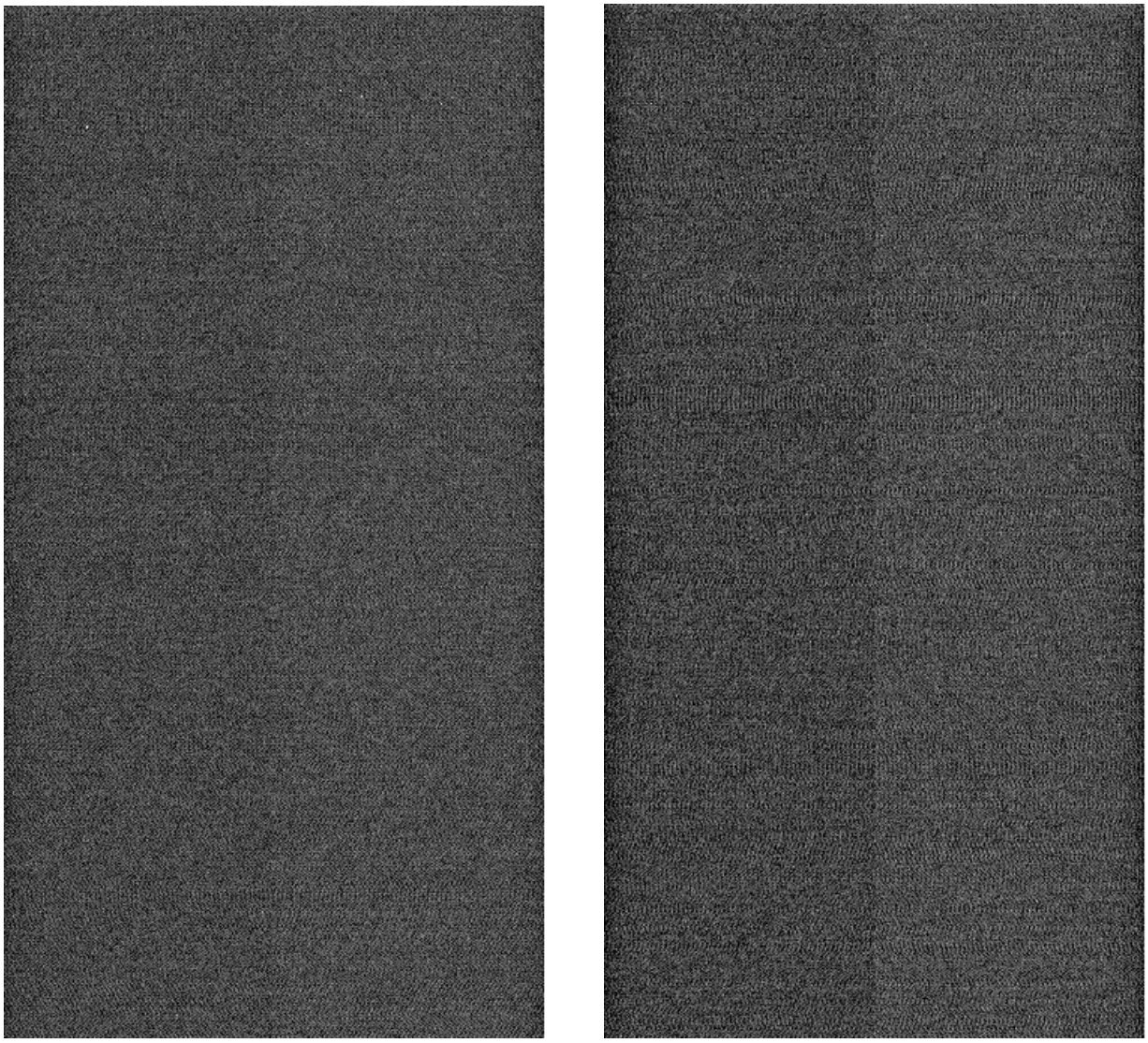


Figure 6: Master bias

The images below show a visual difference between single bias and master bias. Inspecting them closely, one can notice a few differences between them.



(a) Single bias. (b) Master bias

Figure 7: This is a comparison between a single bias frame (left) and a master bias frame constructed from multiple exposures (right).

Important note Bias:

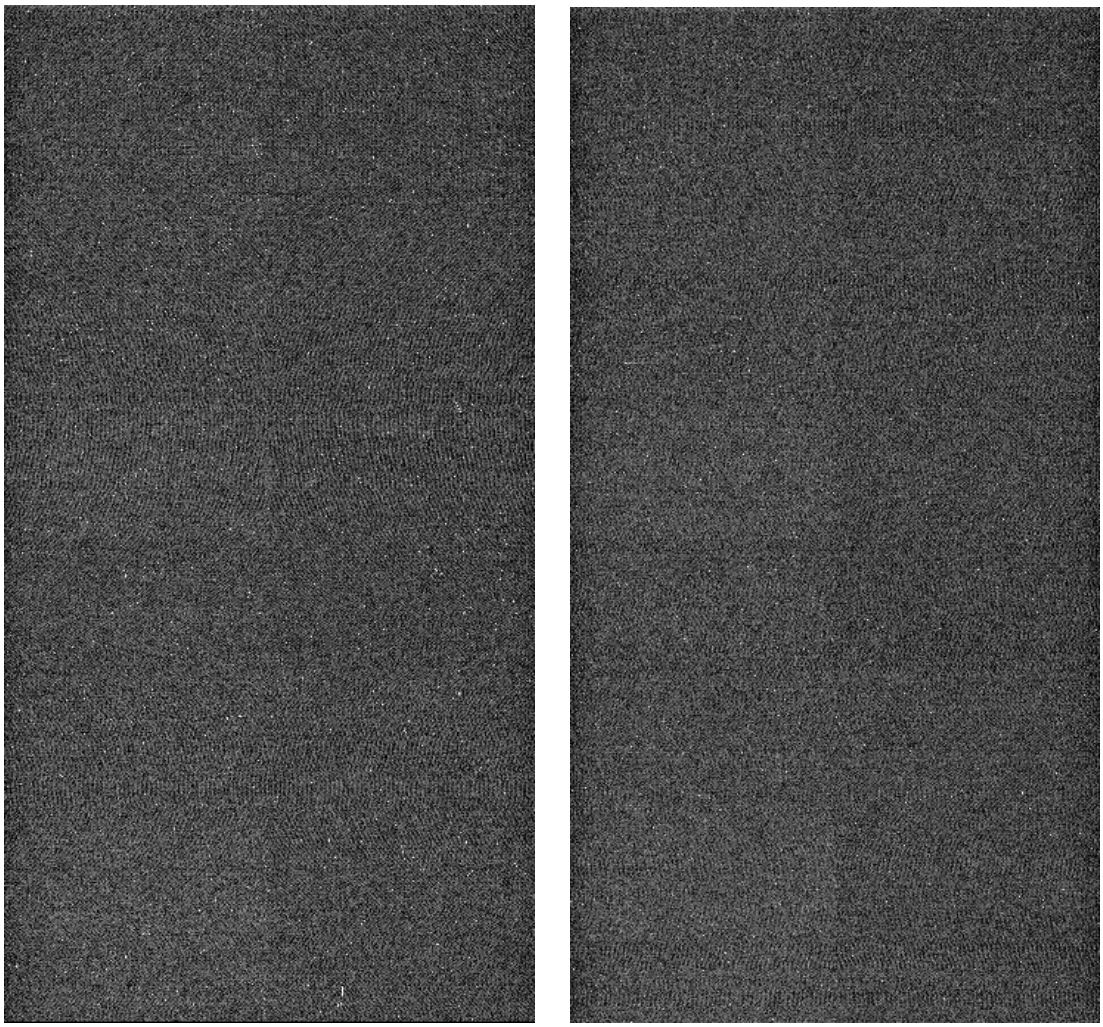
- KOSMOS does have pattern noise, and it can change from bias to bias, so isn't repeatable enough to produce a reliable and robust bias subtraction.
- We are currently averaging over 11 biases, which averages out most of the pattern noise, so it probably isn't ADDING much noise, but it prbably isn't helping either.

8. DARK

Darks correct for thermal noise, hot pixels, and electric noise. As we take science images, as time goes on, the pixels begin to get hotter and brighter. Thus, the concept of thermal noise and hot pixels. An example of electric noise will be some electrons moving at high velocity compared to the rest, causing a mess in the pixels. All of these are registered in science images as they are taken. Therefore, they need to be removed. A dark can be taken before or after your science image using the same exposure time as the science frame, but with the shutter closed. The process of making a master dark is no different from that of master bias.

Run the cell below

```
darktims=[94,95,96]
dark=red.mkdark(darktims,display=None)
```



(a) A single dark. (b) Master dark.

Figure 8: A visual comparison between a single dark and master dark.

Important note on Darks:

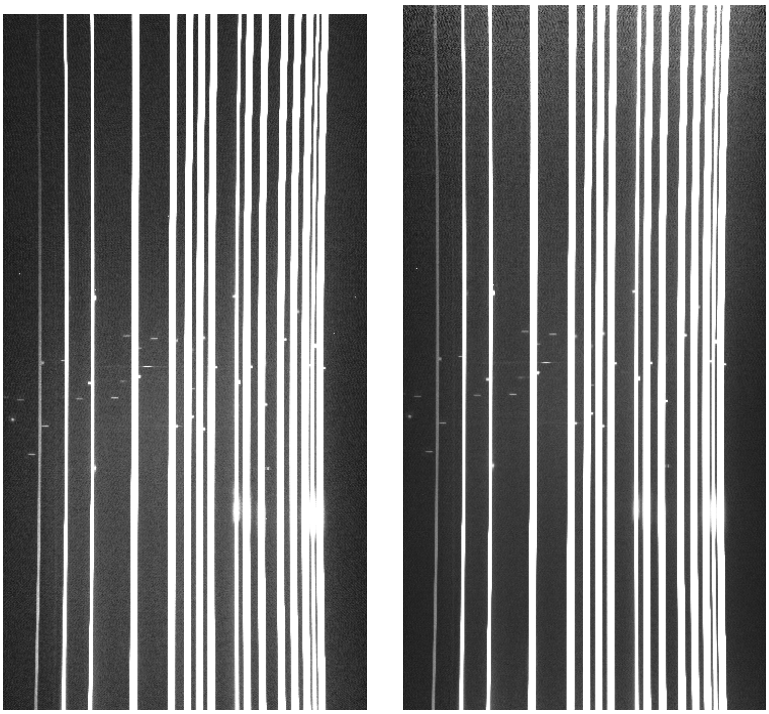
- Dark subtraction - can be useful, but still hard to get enough darks to reduce the readout noise in the combined dark frame (we were currently averaging 3 x 20m exposures, so are only beating readout noise down by root(3))
- The recommendation is to use the 'clip' method in "mkdark()" function to only apply the correction to pixels that are 5 sigma (or whatever is set from the clip parameter) detections of significant positive dark current.

9. FLAT

When one takes a science image with the shutter open, dust or marks of some sort will be on some pixels. These marks reveal that dust or other optical/environmental affects are absorbing some of the light before it gets to the detector. Flats are taken to measure the shape and amount of light that are lost in these marks, so that the lost light can be accounted for and corrected in science images. Flats are taken with bright white light spread over a white surface, which would ensure the same brightness across the pixels in the absence of these dust marks or optical effects. Science images can then be divided by the flat to correct for the uneven illumination that these dust particles and optical effects produce in the pixels in science images.

Run the cell below

```
flatims=[21,22]
flat=red.mkflat(flatims,spec=True)
```



(a) Single flat (b) Master flat.

Figure 9: A visual comparison of a single flat and a master flat

Note: The master flat process is the same as the master bias and dark but simultaneously different. If desired, one can subtract a master bias and master dark from a flat. But the combining process for a master flat is the same.
important! Set spec to true (spec=True) if you want to base your reduction on this pipeline. If True, it creates a "spectral" flat by taking out wavelength shape.

```
flat=red.mkflat(flatims,spec=True)
```

Important note on Flats:

- The recommendation is taking flats through slit mask: since detector response is wavelength dependent, it's important to actually get the right wavelength as projected through the slit mask (which will be different than if we used the long-slit flat).

- cosmetic issues due to motion of slit are probably not a big deal (but reduction could inch into the slitlets to try and avoid the division issues at the edges).

- Littrow ghost issues are... complicated. Jon might think about this, but best mitigation at the moment is to visually look for the artifacts in the flat, and then be cognizant that the final spectrum might have been overdivided by the excess of flux flat, so will have a deficit of flux in the target spectra.

10. ARCS

Arcs refers to the calibration lamps. Calibration lamps are light sources that are used in spectroscopy to ensure the accuracy and reliability of measurements. They are essential tools for calibrating the wavelength scale of spectroscopic instruments. There are several types of calibration lamps, each serving different purposes. In our case, we are using neon lamps. Make a master arc using the code below. The master arc is made using the same combiner as master bias and master dark, but instead of median, it uses sum.
Run the cell below

```
arcs=red.sum([23,24])

# Display the lamp
if t is not None:
    t.clear()
    t.tv(arcs)
```

Looking at the master arc (see figure 10), you will notice that each slit has its portion of light. This means each object must be calibrated with its corresponding arc spectrum.

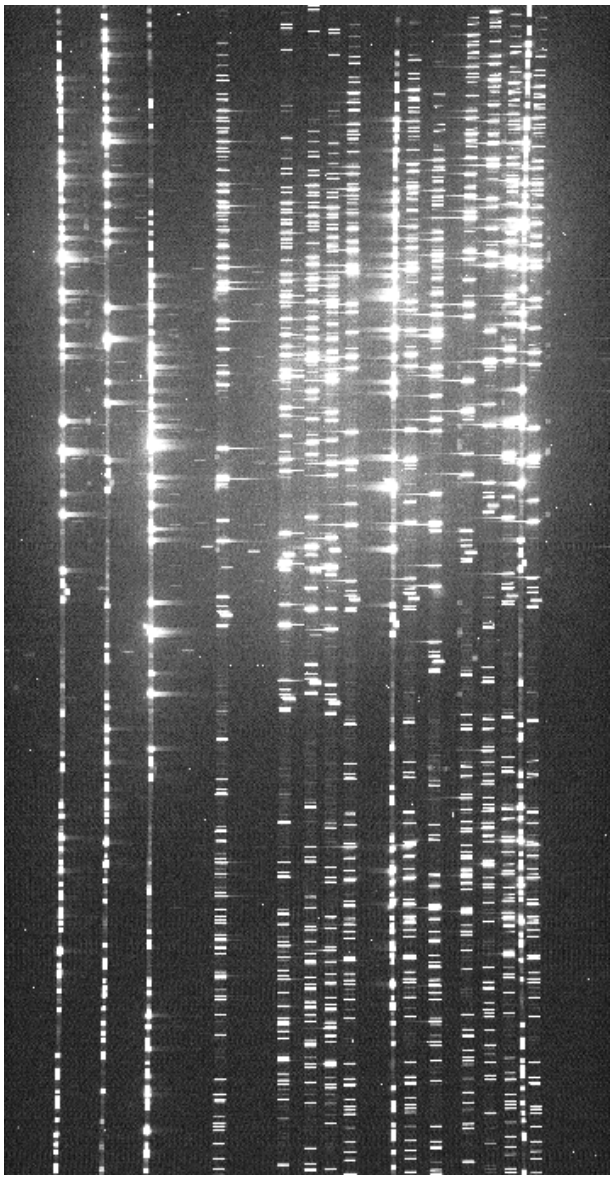


Figure 10: Master arcs

11. FIND AND DISPLAY SLITS ON THE FLAT FIELD IMAGE

Initialize the trace and then use that trace to find the slits in the science image. For finding the slits, it is recommended to use flats, not master flats, a single flat. The reason for using flat is because PyVista tends to keep the overscan data in the reduced images; this is why using the master flat to find the slit location will raise an error. Use the next cells to find and extract your slits.

Run the cell below

```
flat1 = red.reduce(22) # reading the flat
```

- transpose = True, will horizontalize the image in order to the wavelength axis in the x-axis and position in the y-axis.
- display=t, displays the image.
- thresh=0.5, set the threshold of the points of slits at the edges.
- sn = True, look for edges using delta(S/N).

Run the cell below

When done, click the space to close the cell (refer to section) 15 on page 15.

```
# This does not take mkflat().
trace0=spectra.Trace(transpose=True)
t.tvclear()
bottom,top = trace0.findslits(flat1,display=t,thresh=0.5,sn=True)
```

Output

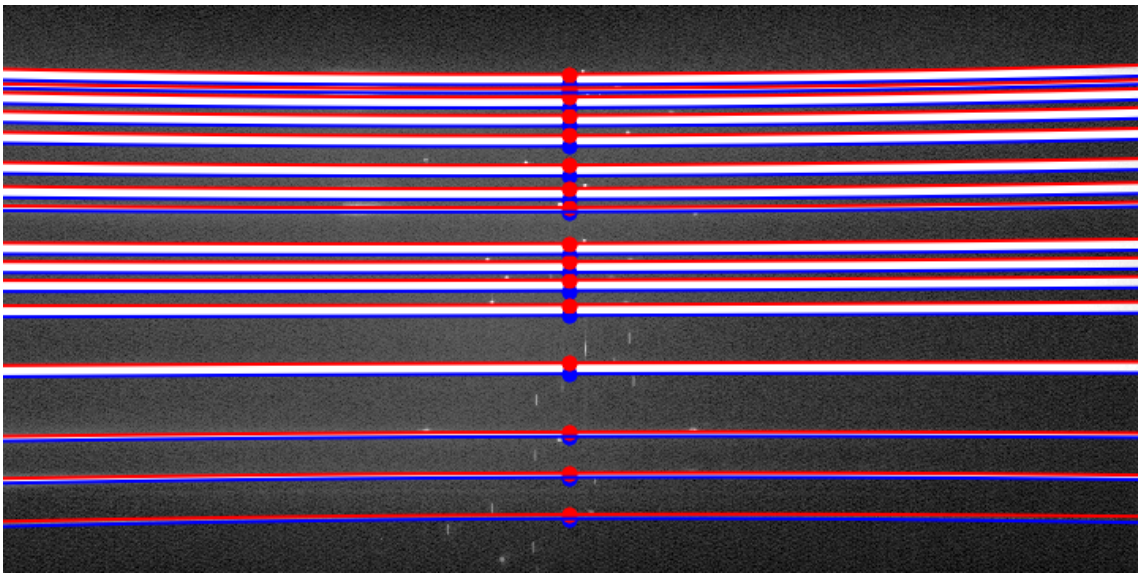


Figure 11: The is a trace of all the slits, including reference stars and source stars.

12. READ SLIT MASK FROM A KMS-FILE

A kms-file is a file containing the information used to make a mask slit. Thus, it is essential to load the correct file. Sorting by 'YMM' is very important for the next steps. As a table, its rows in ascending order correspond to the slit arrangement from left to right in the science image. This allows us to match slits'

positions to their corresponding km-file very easily. Reads slit mask specification data from a specified kms-file and sorts it to prepare it for further processing. If the lengths do not match, it executes the else: block where it prints an error message: print('ERROR, number of identified slits does not match number of targets'). This error handling is important to avoid mismatches that could lead to incorrect data processing or analysis.

Run the cell below

```
kmsfile='kms/Copy of kosmos.23.seg3g2.kms'
targets=slitmask.read_kms(kmsfile,sort='YMM')
if len(targets) != len(bottom) :
    print('ERROR, number of identified slits does not match number of
     ↪  targets')
targets
```

Output

```
   ID    NAME  SHAPE    WID LEN  ROT   ALPHA      DELTA    WIDMM LENMM   XMM      YMM
------- ---- -------- --- ---- --- ---------- --------- ----- ----- ------- -------
TARG113   NN STRAIGHT 4.0  4.0 0.0 212134.279 191220.25 0.683 0.683  -5.252 -34.315
TARG112   NN STRAIGHT 4.0  4.0 0.0 212140.986 191141.81 0.683 0.683 -21.469 -27.768
TARG114   NN STRAIGHT 4.0  4.0 0.0 212123.429 191104.21 0.683 0.683  20.979 -21.323
TARG111   NN STRAIGHT 0.9 10.0 0.0 212132.879 191002.48 0.154 1.707   -1.87 -10.801
TARG110   NN STRAIGHT 0.9 10.0 0.0 212127.784 190908.67 0.154 1.707  10.451   -1.61
```

This is an excerpt of what the kms file looks like.

12.1. *Convert the targets table to a pandas Data Frame*

Converts the targets' table to a pandas Data Frame and displays it, allowing for easier access to targets' indexes.

Run the cell below

```
df = targets.to_pandas() # data transformation
df # A look at your table in panda format
```

	ID	NAME	SHAPE	WID	LEN	ROT	ALPHA	DELTA	WIDMM	LENMM	XMM	YMM
0	TARG113	NN	STRAIGHT	4.0	4.0	0.0	212134.279	191220.25	0.683	0.683	-5.252	-34.315
1	TARG112	NN	STRAIGHT	4.0	4.0	0.0	212140.986	191141.81	0.683	0.683	-21.469	-27.768
2	TARG114	NN	STRAIGHT	4.0	4.0	0.0	212123.429	191104.21	0.683	0.683	20.979	-21.323
3	TARG111	NN	STRAIGHT	0.9	10.0	0.0	212132.879	191002.48	0.154	1.707	-1.87	-10.801
4	TARG110	NN	STRAIGHT	0.9	10.0	0.0	212127.784	190908.67	0.154	1.707	10.451	-1.61

Table 1: kms file in csv format. The important for this transformation is the index notation.

12.2. *Remove specified rows from Data Frame*

If some slits lack a light source or you only need part of the science images, identify and pick the index of the object of interest relative to others in the image. Slit numbers are counted from left to right and match the table above. Remember, Python uses zero-based indexing. In the cell below, you can remove unneeded slits without affecting the original kms-file. The function allows filtering the kms-file by indexes, target IDs, or names.

Targets selection process:

- first select the method of interest by number.
- Then insert the your stars of interest.

The cell returns the trace and targets of the selected stars. You can use the example indexes and target IDs provided. Names are not included here since our stars lack names, but you can filter by names if user's stars are named.

Samples to use for trail run using the sample data:

- indexes: 6,7,9,11,13
- ID: TARG108, TARG107, TARG106, TARG104, TARG102

Cell prompt as the function runs:

```
Select targets based on:
1. Index
2. ID
3. Name
Enter the number corresponding to your choice:  1
You chose to select targets by index.
Enter the indices (comma-separated):  6,7,9,11,13
```

Note: We exclude the entity of the function (select_targets())in here because it required nothing of the user except what is shown below. If user wish to understand it functionalities read through it in the notebook.

```
trace, targets = select_targets(targets, trace0)
```

Outputs

```
1 targets
```

ID	NAME	SHAPE	WID	LEN	ROT	ALPHA	DELTA	WIDMM	LENMM	XMM	YMM
TARG108	NN	STRAIGHT	0.9	10.0	0.0	212127.508	190828.88	0.154	1.707	11.119	5.182
TARG107	NN	STRAIGHT	0.9	10.0	0.0	212133.847	190811.86	0.154	1.707	-4.212	8.077
TARG106	NN	STRAIGHT	0.9	10.0	0.0	212133.934	190720.87	0.154	1.707	-4.422	16.78
TARG104	NN	STRAIGHT	0.9	10.0	0.0	212136.755	190631.54	0.154	1.707	-11.247	25.194
TARG102	NN	STRAIGHT	0.9	10.0	0.0	212134.103	190555.71	0.154	1.707	-4.831	31.313

Table 2: cleaned kms-file with only the interested stars.

```
1 vars(trace)
```


```
{'type': 'Polynomial1D',
 'degree': 2,
 'sigdegree': 0,
 'pix0': 0,
 'spectrum': None,
 'rad': 5,
 'transpose': True,
 'lags': range(-50, 50),
 'model': [Polynomial([1089.05958351, 6.49312832, 5.26758998], domain=[98., 3998.], window=[-1.,  1.], symbol='x'),
  Polynomial([1154.49886865,    6.51239823,    6.81155863], domain=[  98., 3998.], window=[-1.,  1.], symbol='x'),
  Polynomial([1352.12850914,    6.53479617,   11.59481407], domain=[  98., 3998.], window=[-1.,  1.], symbol='x'),
  Polynomial([1542.12042917,    6.63076955,   16.0577897 ], domain=[  98., 3998.], window=[-1.,  1.], symbol='x'),
  Polynomial([1680.82160588,    6.6160561 ,   19.38130477], domain=[  98., 3998.], window=[-1.,  1.], symbol='x')],
 'sigmodel': None,
 'sc0': None,
 'rows': [[1089, 1128],
  [1154, 1193],
  [1352, 1391],
  [1542, 1581],
  [1681, 1720]]}
```


Note: the important things in the return trace, is the **list of values of the rows key and model key**. They represent the edges of the slits with their polynomial fit around the edges. This trace is then use for all 2D extractions in the notebook.

13. 2D EXTRACTION OF THE ARC FRAMES.

This uses the drived trace to extract a 2D image of each slit to be used in the wavelength calibration step.

Run the cell below

When done, click the space to close the cell (refer to section 15) on page 15.

```
1 arcec1=trace.extract2d(arcs, display=t)
```

Output

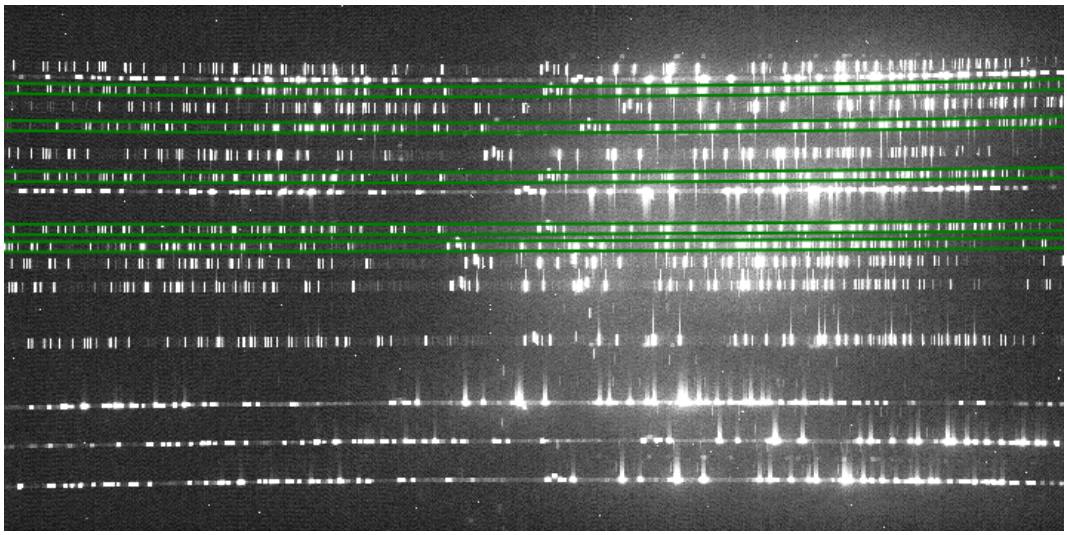


Figure 12: Like the 2D extraction of the stars. The green lines denote the edges of each slit that's been extracted.

13.1. *Updates the header of each extracted 2D image*

Load the target file and add XMM and YMM for each slit to the headers of each extracted image. REQUIRES PERFECT MATCH BETWEEN INPUT TARGETS AND IDENTIFIED SLITS!

- Inside the loop for arc, target in zip(arcec1, targets):, it iterates over pairs of extracted arc data (arc) and target data (target) simultaneously. zip(arcec1, targets) creates these pairs, ensuring that each arc is associated with its corresponding target.

- For each pair, arc.header['XMM'] = target['XMM'] and arc.header['YMM'] = target['YMM'] updates the FITS header of the arc data with the X and Y coordinates from the target data. These headers ('XMM' and 'YMM') represent physical positions or moment measurements that are essential for further analysis or cataloging.

Run the cell below

```
for arc,target in zip(arcec1,targets) :
    arc.header['XMM'] = target['XMM']
    arc.header['YMM'] = target['YMM']
```

Wavelength shift consept explained by Jon Holtzman

"Now loop through each extracted arc to do wavelength calibration. This requires a little effort because the change in the location of the slit relative to the default saved wavelength calibration is significant enough that it can be a challenge to automatically find the lines, since the change in spectrum is more than a simple shift (and, in fact, more than a shift + dispersion change).

However, a simple shift is usually enough to identify some of the lines, and these can be used to bootstrap the wavelength solution; the initial identification is easier if you use an estimate of the shift from the mask design XMM.

You can really help this process if you supply an initial wavelength calibration (a pyvista WaveCal object) that was done using the same lamp(s) as your arc exposures (here, using KOSMOS_red_waves.fits'), and using a master line list that corresponds to these lamp(s) (here, using ne.dat). If you choose another WaveCal to start from, you may need to get a correct approximate relation for the shift from the reference spectrum as a function of XMM. For KOSMOS, seems like -22.5(XMM) gives a rough pixel shift from a center slit location, -22.5(XMM-24.44) for a low slit location.

The next cell just shows how the simple shift fails, even using the XMM for each slit to shift: all of the arc lines don't match up with just a translation."

To undertand this better try changing -22.5 value and see what happens to the plot.

Run the cell below

```
plt.figure()
wav2=spectra.WaveCal('KOSMOS/KOSMOS_red_waves.fits')
plt.plot(wav2.spectrum[0])
for i, arc in enumerate(arcec[0:1]) :
    shift=(arc.header['XMM']*-22.5)
    plt.plot(arc.data[19][int(shift):]*30)
    print(shift)
    plt.draw()
```

Output

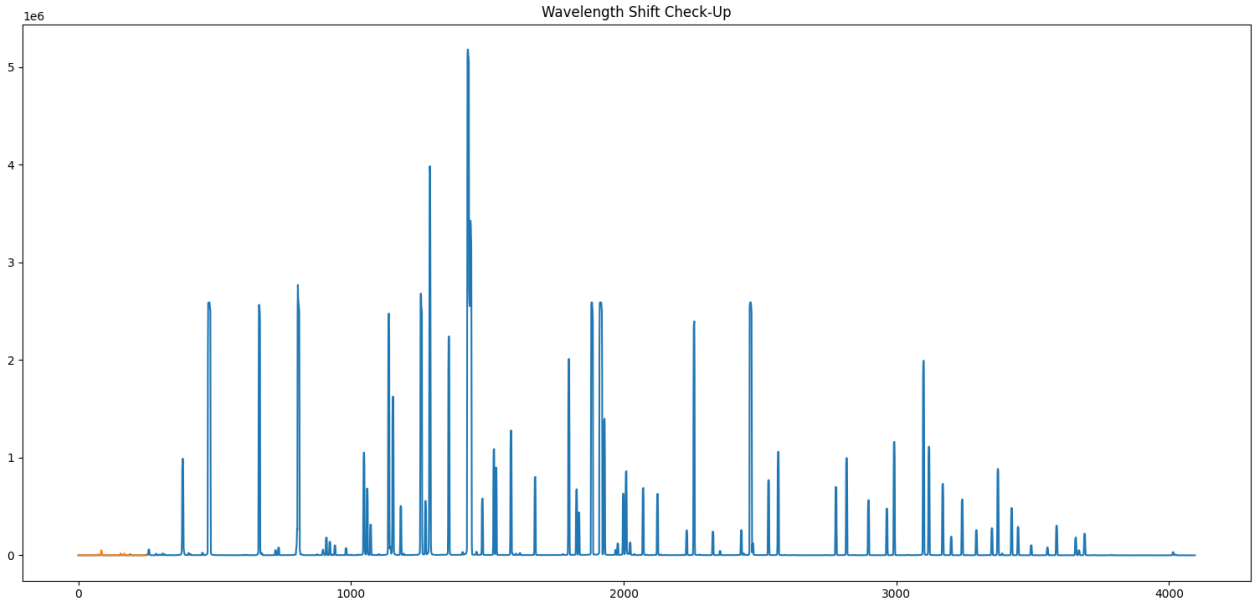


Figure 13: Visual verification a shift between the arc spectrum and and a solution spectrum.

14. THIS CELL PERFORMS WAVELENGTH CALIBRATION FOR EACH ARC SPECTRUM IN 'ARCEC1' USING A PREDEFINED WAVELENGTH SOLUTION FILE

- clobber=False: If you are doing wavelength caibration for the first time set this to ture. If you confident in your solutions set it to false to skip this you each time run the notebook.
- for i, arc in enumerate(arcec1): iterates through each arc spectrum in arcec1, with i as the index and arc as the spectrum.
- wavname = 'CofIwav_:s.fits'.format(targ['ID']): define names for wavelength solutions saving.
- if clobber or not os.path.exists(wavname) : Set the path for files. If they exist call upon them, if not set paths for saving the solutins.
- wav=spectra.WaveCal('KOSMOS/KOSMOS_red_waves.fits'): Initializes a wavelength calibration object wav using a FITS file that contains reference wavelength data.
- wav.fit(degree=3): Dose a fit base on degree, this is also used in detecting and rejecting outliers in case of 2D images.
- nrow=arc.shape[0]: Retrieves the number of rows in the arc data, which represents the slit width of each slit in the spectral data.
- shift=int(arc.header['XMM']*-22.5): Extracts a shift value from the FITS header 'XMM'. This shift is used as an initial guess to align the observed spectrum with the reference.
- lags=np.arange(shift-20, shift+20): Defines a range of lags around the initial shift for more refined wavelength calibration, adjusting the position to better align with reference features.
- The while loop (while iter:) continues until no further identification improvements are made:
  - iter = wav.identify(...): Calls the identify method on wav to align the spectrum. It adjusts the calibration based on the central row of the spectrum (arc[nrow//2]) and updates the iter flag based on whether further adjustments are needed.
  - Parameters like plot=True and plotinter=True enable visualizations of the calibration process to manually inspect and adjust the fit.
  - After the first iteration, lags is reset to a standard range (np.arange(-50, 50)), which focuses on fine-tuning around a new estimated center.
  - plt.close(): Closes the plot to clear memory and avoid overlapping plots.
  - After exiting the loop, wav.identify(...) is called again on the whole arc to perform a 2D wavelength solution across the slit by sampling at 10 locations (determined by nrow//10).
  - The final plt.close() again ensures that all plots are closed, maintaining clean output without residual figures.
  - wav.write(wavname): saving wavelength solution of a specific slit 2D arc.
  - wav.add_wave(arc): add the solution to the arc.
  - t.tv(wav.correct(arc,arc.wave[nrow//2])): Resample input image to desired wavelength scale. The current wavelength solution is used, linearly interpolates to specified wavelengths, on a row-by-row basis, and then displays on the TV.

- 'KOSMOS/KOSMOS_red_waves.fits': this file is found /pyvista/python/pyvista/data/KOSMOS/ directory.

- 'new_wave_lamp/copy_new_neon_red_center.dat': this file is found in new_wave_lamp folder. This file was created using ARC 3.5 meter neon plot, visit (https://www.apo.nmsu.edu/arc35m/Instruments/KOSMOS/images/red_ctr_intNe_plot.png); and some filled-in values from pyvista's "ne.dat" found in /pyvista/python/pyvista/data/lamps/ directory.

Run the cell below

```
clobber=False  # set to False if you want to use any saved ones
for i,(arc,targ) in enumerate(zip(arcec,targets)) :
    wavname = 'CofIwav_{:s}.fits'.format(targ['ID'])
    if clobber or not os.path.exists(wavname) :
        wav=spectra.WaveCal('KOSMOS/KOSMOS_red_waves.fits')
        wav.fit(degree=3)
        nrow=arc.shape[0]

        # get initial guess at shift from reference using XMM (KOSMOD red
        ↪ low!)
        shift=int(arc.header['XMM']*-22.5) # +550  #-wav.pix0)
        lags=np.arange(shift-20,shift+20)

        iter = True
        while iter :
            iter = wav.identify(arc[nrow//2],plot=True,plotinter=True,
                        lags=lags,thresh=10,
                        file='new_wave_lamp/copy_new_neon_red_center.dat')
            lags=np.arange(-50,50)
            plt.close()

        bd= np.where(wav.weights<0.5)[0]
        print(wav.waves[bd])
        # Do the 2D wavelength solution, sampling 10 locations across
        ↪ slitlet
        wav.degree=5
        wav.identify(arc,plot=True,nskip=nrow//10,thresh=10)
        plt.close()
        wav.write(wavname)
        wav.add_wave(arc)
        t.tv(wav.correct(arc,arc.wave[nrow//2]))
```

Wavelength calibration steps: As the user runs the previous cell, an interactive plot window should pop up with identified lines. The plot should have a third-degree polynomial curve shape (see figure 14). If not, click ”I” until you get the shape. If clicking the keys does nothing, stop the cell and re-run it. After getting the shape, use the following method to remove points that do not lay on the curve's path.

Input in plot window:

- l : to remove all lines to the left of the cursor.
- r : to remove all lines to the right of the cursor.
- n : to remove line nearest cursor x position.
- i : return with True value (to allow iteration).
- anything else: finish and return.

Remove points till the rms is below one (1). Then ”I” to re-identify lines if the new identification looks like figure15. Proceed to the next-arc spectrum; if not, repeat the previous steps.

Output 1

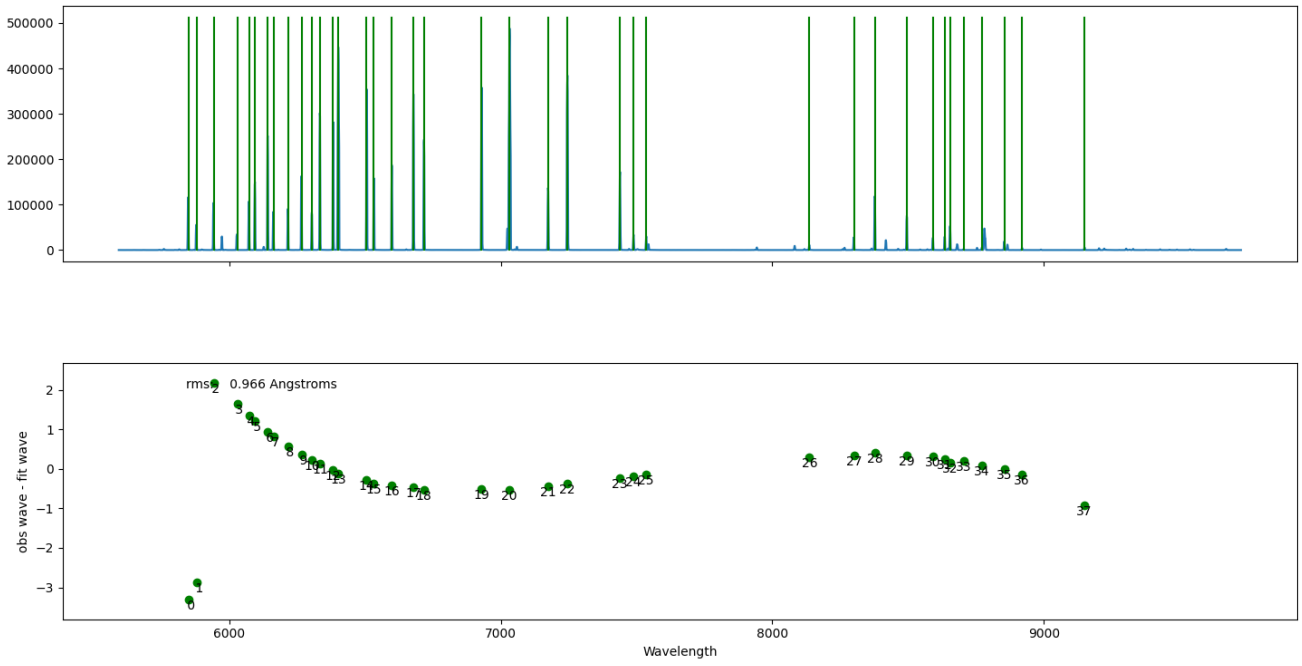


Figure 14: The polynomial curve should look similar or close to it.

Output 2

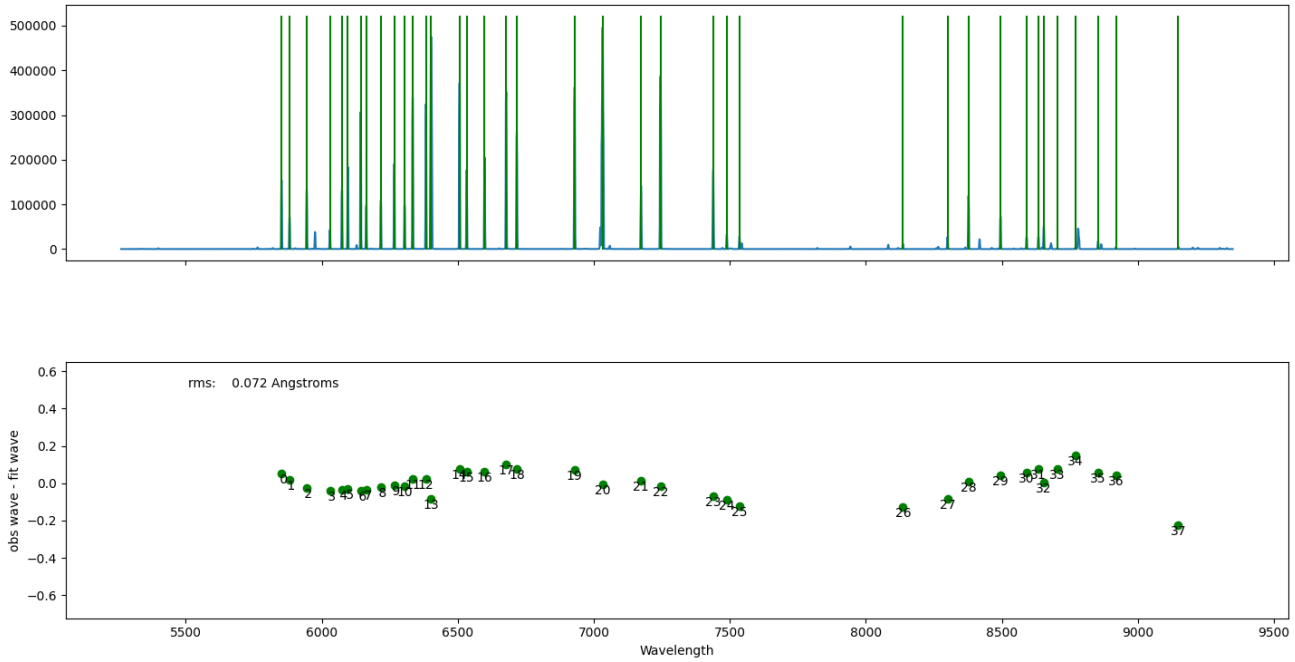


Figure 15: User's plot should look similar to this plot When all the lines are correctly identified through interaction.

It will then take identified lines and sample them at ten places across the slit (see figure 16)

Output 3

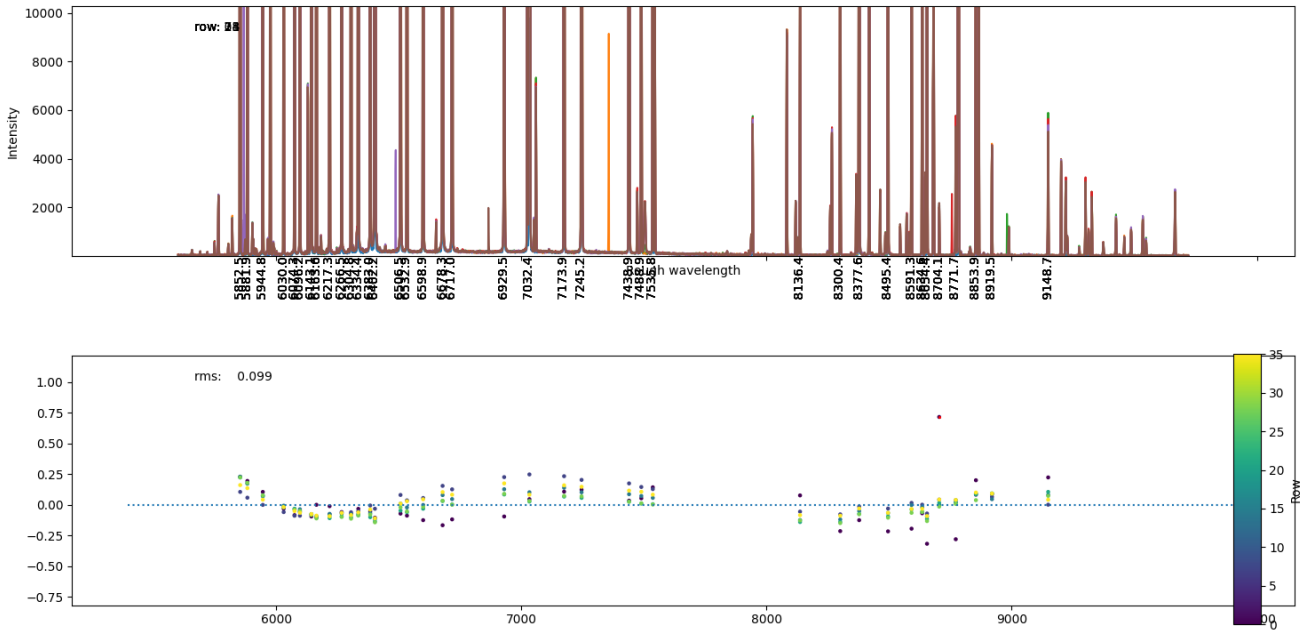


Figure 16: Sampling the identified lines at ten places across the slit for better wavelength solution.

15. SCIENCE IMAGE REDUCTION

After preparing the calibration images, we can reduce the science image. The code below will trim the overscan of your science images, subtract basis and dark, eject cosmic rays, and flat-field it (see figure 19).

As you run the code below, if the display is set to “t” (diaplay=t), pay attention to the output of the cell below. Each time the cell computes an action, it will ask the user to verify it through the main display before proceeding to the next step.

Follow the following steps:

- click space to proceed to the next task.
- You can click ”-” to see the image of the previous task and click ”+” to see its successor image.
- When done, click the space to close the cell.

Figure 17: The above steps are applicable in this window.

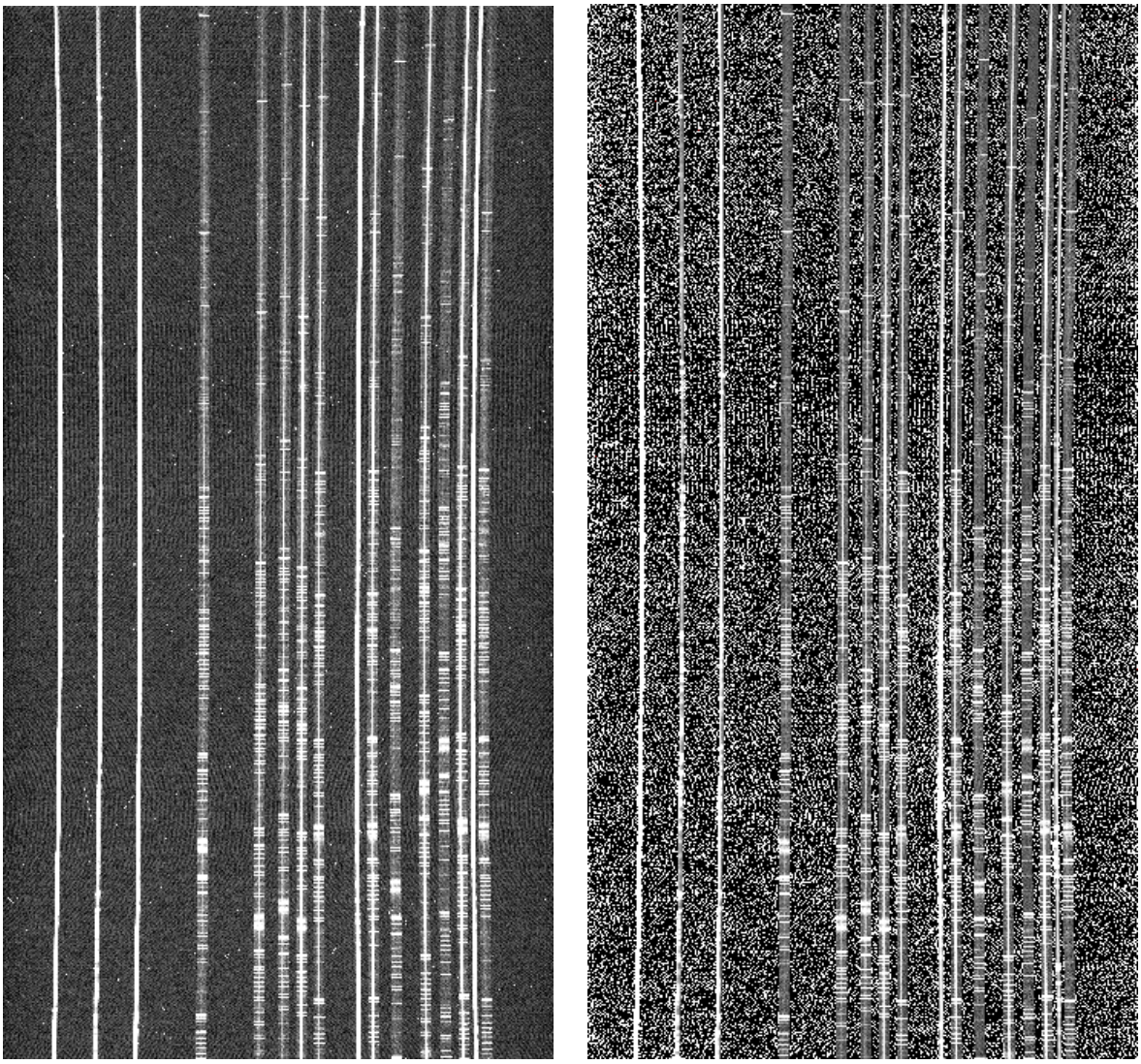
Run the cell below

```
star1=red.reduce(20, crbox='lacosmic', bias=bias, flat=flat, dark=dark,
 ↪  display=t)
```

Cosmic-rays: are particles or celestial objects that are moving at near speed light in the galactic plane and the atmosphere. The telescope collects these particles (see figure 18) as it collected the science images and needed to be removed.

Figure 18: Detected cosmic rays

Looking at the reduced image (see figure19), the user will notice that between the slits, there are some points that are black and some white. The cause for that is the fact that we took our flat through the slits; therefore, the light is uniform within the slit's boundaries but not across the whole image. Thus, a zero division error occurs during flat-fielding at points without light, representing the black spots.
This issue can be solved by taking a flat across the entire lens instead of through slits, but recommended for multi-slits.



(a) Raw science image (b) Reduced science image

Figure 19: Raw image and Reduced image.

Note: though we used the calibration images in reducing the science; it is recommended not to use them and have "crbox='lacosmic',crsig=10" in the reduction. This is because most of relevant noises are taking care of by the cosmic-ray process (see figure 20). I will be using reduction method below if user wishes to use the science image with calibration images applied just replace "imrc" with "star1" in 2D extraction.

```
# basic image reduction with cosmic rays
imcr=red.reduce(20,crbox='lacosmic',crsig=10,display=t)
```

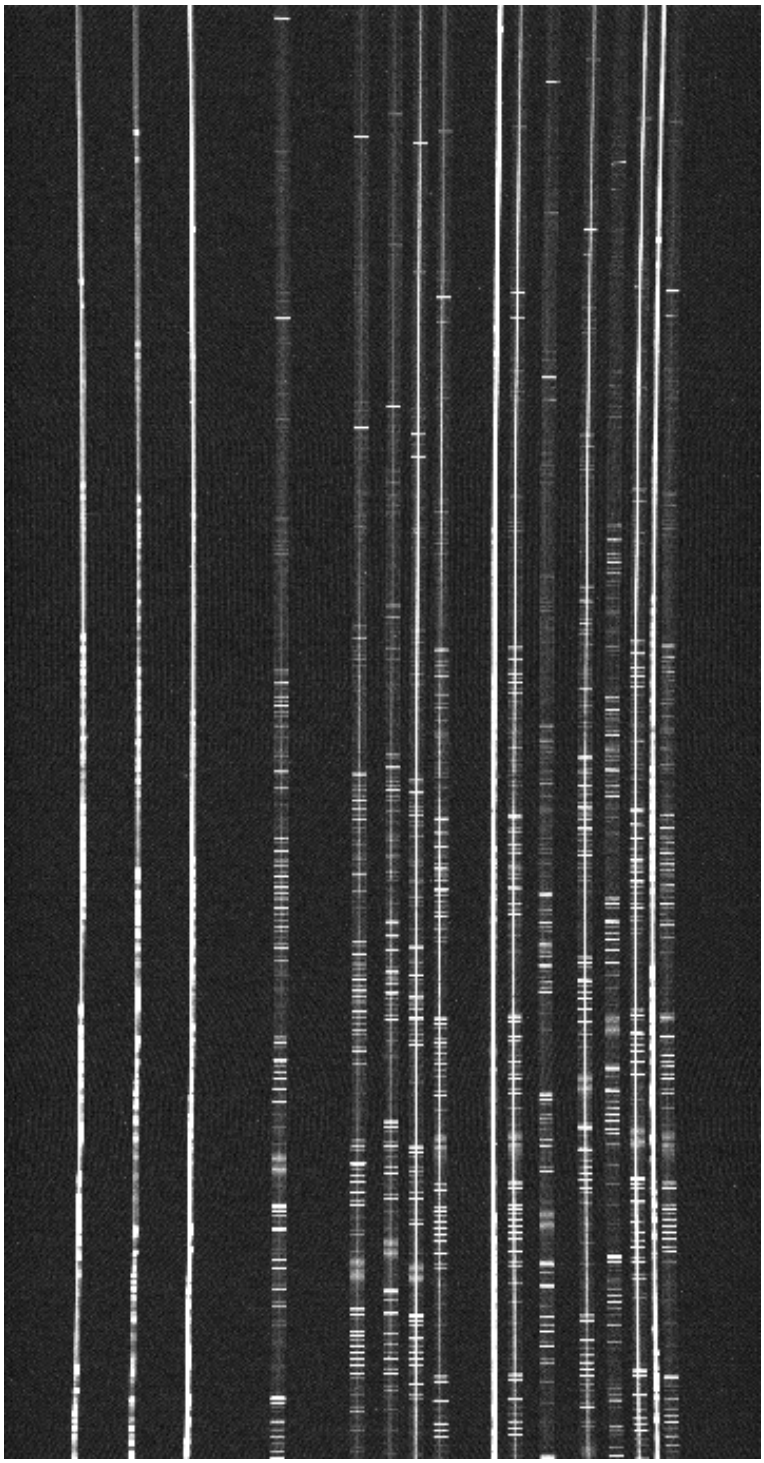


Figure 20: Reduced science without calibration images.

16. 2D EXTRACTION OF STARS

The function ‘multi_extract2d()‘ performs 2D spectral extraction and can apply wavelength calibration or adjustment based on user input. It is interactive, so follow the output prompts from the cell to proceed. The 2D and 1D extraction functions are quite extensive; therefore, I will explain selectively the parts that the user can change to achieve the desired extraction.

2D Cell choice prompt

```
Select a choice of wavelength adjustment:
1. 2D wavelength adjustment.
2. No 2D wavelength adjustment.
Enter the number corresponding to your choice:
```

If the user selects option 1, the function will apply a 2D adjustment, which means that no adjustments are needed during the 1D extraction. If the user prefers not to apply a 2D wavelength adjustment, choose option 2. With option 2 selected, no further action is required from the user. However, if option 1 is chosen, there are parameters available for adjustment to achieve the desired results.

Parameter explaination multi_extract2d()

```
multi_extract2d(red, trace, targets, imcr, thresh=10, display=None, rad=5)
```

- rad=5: This represents the radius used to determine which rows of the 2D spectrum should be excluded from the skyline-based wavelength correction. The rows near the center of the slit are excluded, and rad defines the boundary of that central region.
- thresh=10: This is a threshold value used in the skyline-based wavelength correction process. It specifies the minimum intensity of detected sky lines (emission lines from the sky background) that should be used to correct the wavelength solution. A higher threshold excludes weaker lines from being used in the correction.
- This represents the radius used to determine which rows of the 2D spectrum should be excluded from the skyline-based wavelength correction. The rows near the center of the slit are excluded, and rad defines the boundary of that central region.
- display=None: If provided, shows a 2D extraction on graphical display for visual debugging or quality checks. If set to None, no visual output is shown.

Note: it is recommended to choose option 1, as it provides the same results as option 2 with less user interaction. However, if you select option 2, you will still have the opportunity for wavelength adjustment later in the process. Option 2 is ideal if you’d like to gain a basic understanding of wavelength adjustment, but for a quicker and more streamlined experience, option 1 is preferred.

Run the cell below

```
out=multi_extract2d(red,trace,targets,imcr,display=t)
```

Output 2D extraction trace

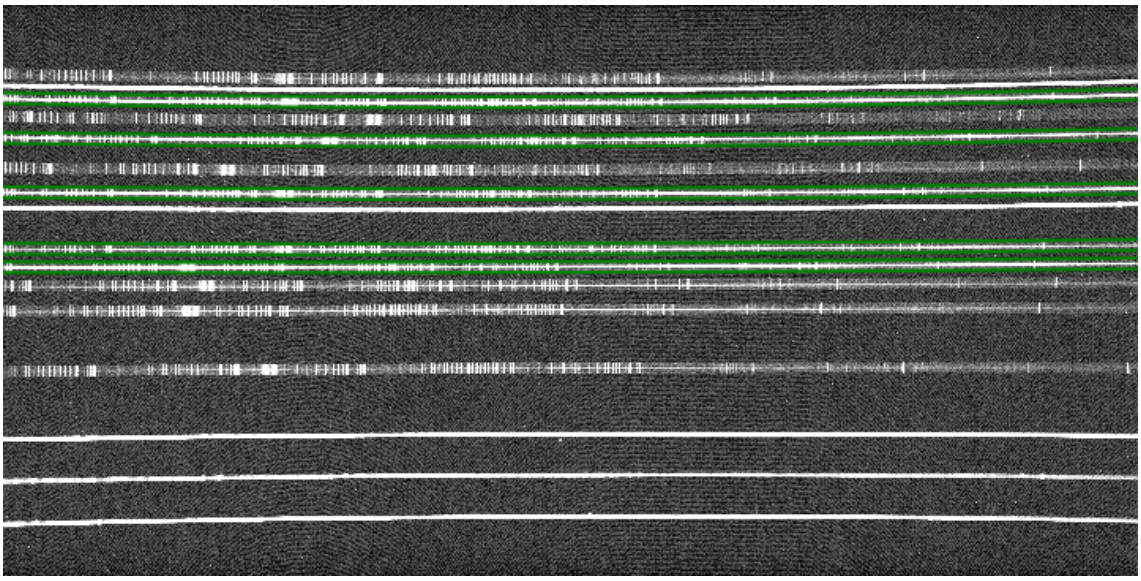


Figure 21: 2D extraction traces of slits

Output 2D Wave adjustment

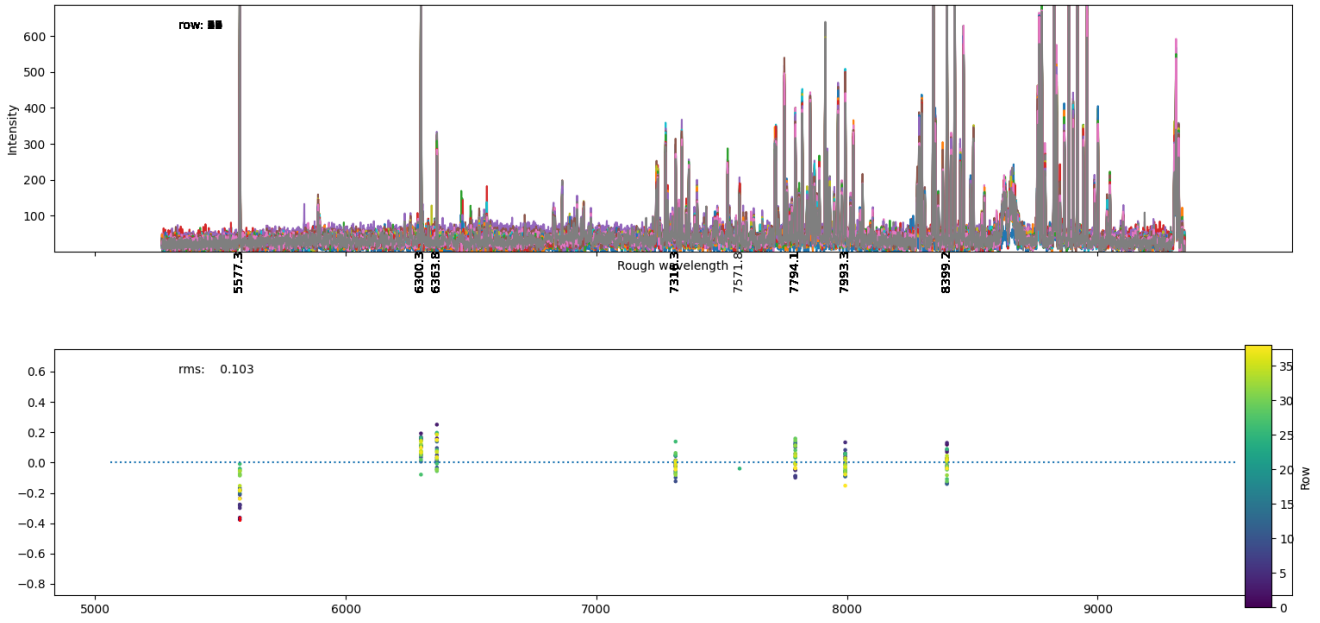


Figure 22: Visual inspection of the 2D wavelength adjustment

Output for option 2

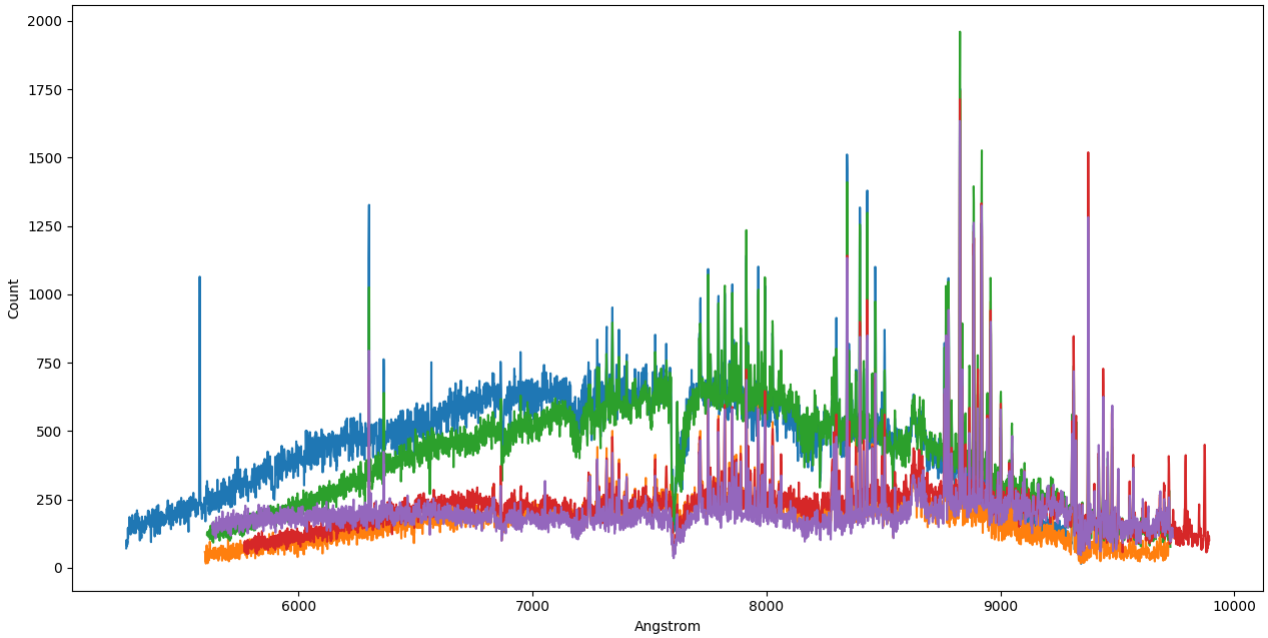


Figure 23: This is a visualization of the center rows of the silts to see how the stars might look after 1D extraction. View it as a prediction of the spectra.

Note: The actual plot for figure 23, the prediction plot are done individually we put them together here for convenience.

17. 1D EXTRACTION/WAVELENGTH CALIBRATION ADJUSTMENT

The ‘multi_extraction1d()‘function takes the outputs from the 2D extraction and treats each independently. It offers two 1D extraction methods for the user to choose from (refer to the cell prompt 17). Since this process is interactive, it is important to carefully review the outputs to proceed correctly to the next step.

1D cell prompt 17

```
If you did 2D adjustment select 1 else select 2:
1. No sky adjustment.
2. Sky adjustment.
Enter the number corresponding to your choice:
```

Before delving into the explanation of parameters at the user's disposal.
There are a few lines of code that are worth explaining to aid a user's understanding of one 1D extraction.

```
trace1 = spectra.Trace(transpose=False, sc0=None, lags=range(-39,39),
    degree=3, sigdegree=3)
```

- This line creates an instance of the Trace class from the spectra module, which is responsible for following the trace of the spectrum in 2D data. The user can create his own trace from scratch using this line of code.
  - transpose=False: Specifies whether to transpose the data or not. False because it has already been transposed.
  - sc0=None: is very important, it allows the trace to follow the peak value. If it is set to a value, the extraction windows will shift on the basis of that value. Which does not look good you can check this visually through the display window by zooming in on it.
  - lags=range(-39,39): The range of row or column offsets (lags) to consider when tracing the spectrum. It defines how far the trace will look for changes in the spectral line positions around the initial center.
  - degree=3: Specifies the degree of polynomial for fitting the trace. In this case, a 3rd-degree polynomial is used to describe the curve of the trace.
  - sigdegree=3: Specifies the degree of smoothing applied to the trace signal (this applicable for Gaussian trace).

```
trace1.rows = [0, spec2d[i].data.shape[0]]
```

- This line sets the rows (or range of data) that the Trace object should consider during the tracing process.
  - rows = [0, spec2d_slice.data.shape[0]]: This means the trace should cover all rows of the 2D spectrum for the i-th slice. spec2d_slice.data.shape[0] represents the total number of rows in the data slice. This is shifting the rows values from initial rows slices from findings slit positions to columns from 0 to 39 of max Kms-file width.

```
trace1.index = [0]
```

- This line sets the starting point (or index) for the trace. The index specifies where the trace should begin. In this case, it starts at row 0.

```
peak, ind = trace.findpeak(spec2d_slice, thresh=10, width=19,
↪  sort=True,ack percentile = 50,method='linear')
```

- This line calls the findpeak method on the trace object to locate peaks in the 2D spectrum for the i-th slice.
- thresh=10: Specifies a threshold for peak detection, where only peaks with intensity above 10 will be considered.
- width=19: Defines the width of the region over which the peak will be searched. This value is subtracted from itself and added to itself to get the full width of the slit.

- sort=True: Instructs the method to sort the peaks based on their intensities; this is because, at times, we get multiple peaks.
- back_percentile = 50: If the user suspects that the code is identifying peaks that do not correspond to the star, this parameter can be adjusted to remove approximately 50% or more of the sky background from the data, helping isolate the correct peak. The default value is set to 10. **Caution**: Setting this percentage too high may result in the removal of valuable data.
- method ='linear': It's a Numpy method of calculating the percentile to remove, check (numpy.percentile manual for various methods available).
- The method returns two values:
  - peak: A list of detected peak positions.
  - ind: The corresponding indices of the peaks.

```
srow = [peak[0]]
```

- This line creates a list srow that stores the starting row for tracing the spectrum, which is set to the position of the first detected peak (peak[0], the brightest peak).

```
trace.trace(spec2d_slice, srow, skip=10, gaussian=True, display=t, rad=5)
```

- This line initiates the tracing process along the spectrum using the trace method of the trace1 object.
  - spec2d_slice: The i-th slice of the 2D spectra data is passed for tracing.
  - srow: The starting row for the trace (derived from the first detected peak).
  - skip=10: The trace will skip every 10 pixel to speed up the tracing process and avoid unnecessary pixel-by-pixel tracing.
  - gaussian=True: Indicates that a Gaussian function will be used to fit the trace along the spectrum, ensuring smoother results. If user wish to use centroid tracing set guassian to false.
  - display=t: Specifies whether to display the tracing process and results. Do a visual inspection by zooming in on the plots.
  - rad=5: The radius around the trace to include during tracing, effectively limiting how far from the trace it will search for additional features.

The above explanations have an assigned value, those are for demonstration basically hard coded but in reality they're not hard coded. See the code bellow.

```
spec1d = multi_extract1d(out, skip=80)
```

This changes the value of skip to speed up the process.

If user choose option 1 then the parameters they can change are:

- thresh, skip, degree, sigdegree, and, display.

If user chose option 2 and the sky_cal is set to false the function will just return spectra without sky adjustment. If sky_cal is set to true then it will conduct a sky adjustments base on sky spectra. **Here are parameters user can change:**

- sky_cal=True: calibration adjustment base on sky background. If set to false return spectrum without adjustment whether 2D adjusted or not.
- thresh_sky: Threshold for detecting sky lines.
- linear=True: if True, allow for dispersion to be adjusted as well as wavelength zero-point requires at least two sky lines! If set false you still have a 0-th order shift adjustment.
- file='pyvista/python/pyvista/data/sky/skyline.dat': The path to the file with known skyline data for 1D adjustment. Pyvista has a default file as well ('skyline.dat').

Run the cell below

Note: As user conduct 1D extraction it's advisable to visually check the trace and the extraction steps. It is even important to do this as user input a rad value for each spectrum look through the windows to visually confirm trace and extraction by zooming in on them (see figure 24 and 25). This is applicable in both options.

```
spec1d = multi_extract1d(out)
```

Output 1: Interactive output

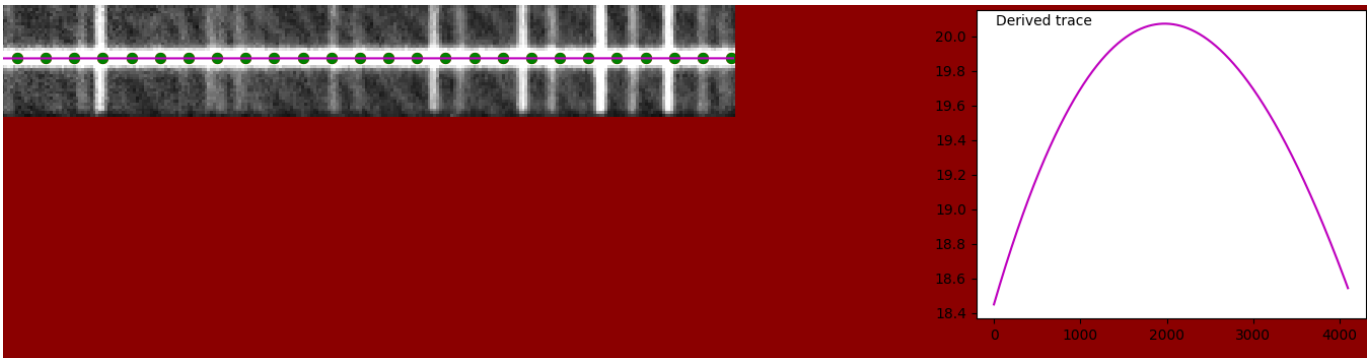


Figure 24: The image on the left is the trace through the peak, and the one on the right is the Gaussian fit of the race.

Output 2: Interactive output

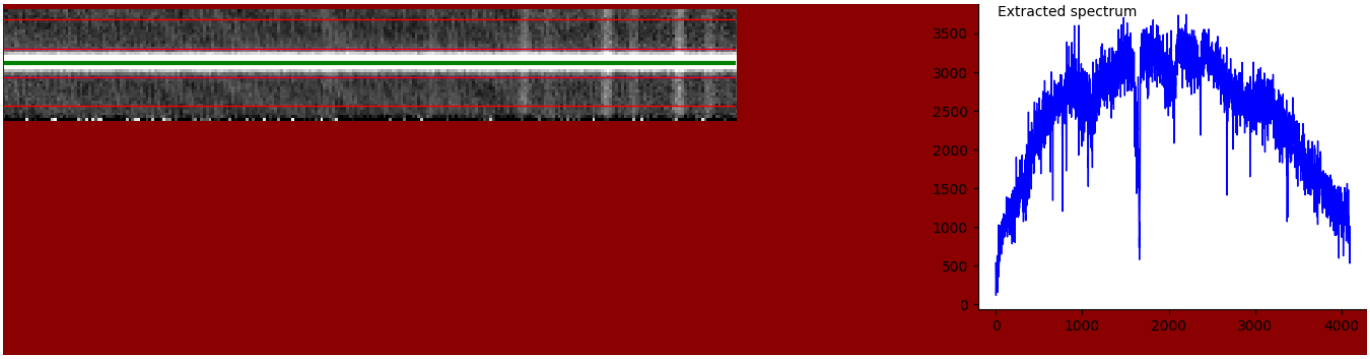


Figure 25: The image on the left, with red and green lines, represents the 2D extracted slit width. The blue image on the right shows the spectrum of a star, which is extracted from the region between the green line and the adjacent red lines. The red lines define the boundaries of the star extraction window on both sides, while also setting the limits for the sky spectrum windows on either side of the star's spectrum.

Output 3 (an extra step for option 2)

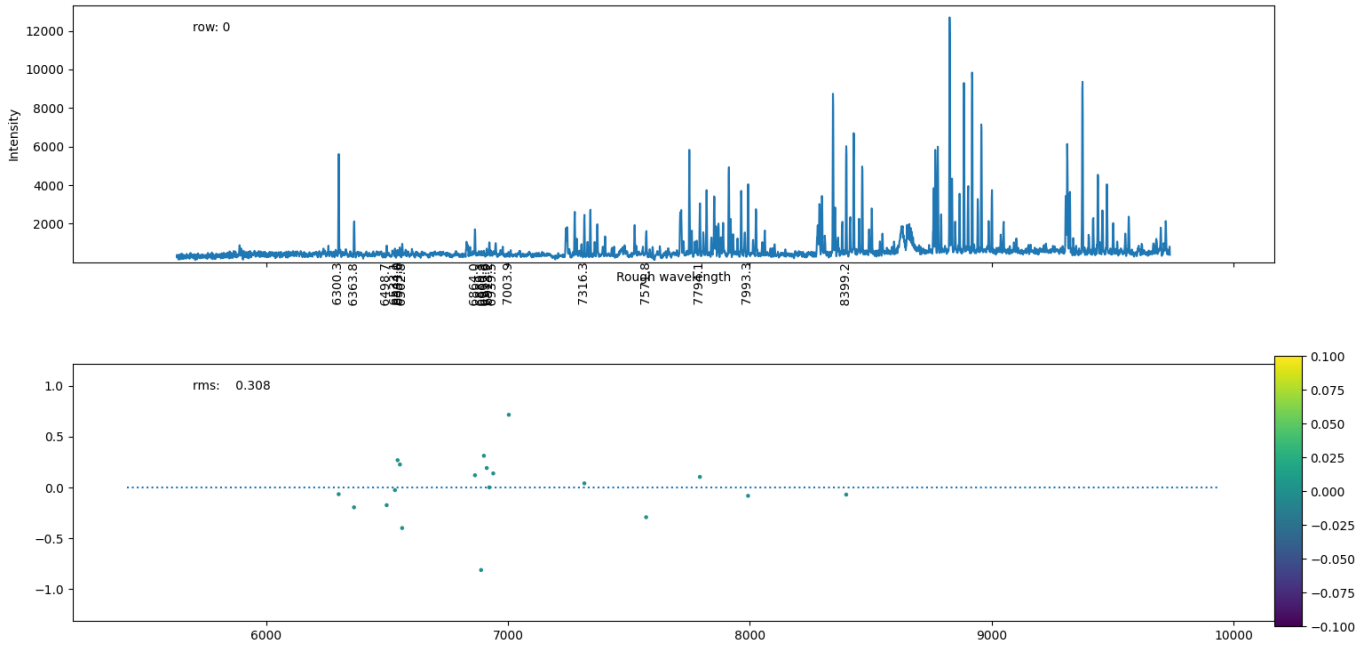


Figure 26: Visual inspection of the 1D sky adjustment process.

Final output 3 for option 1 and 4 for option 2.

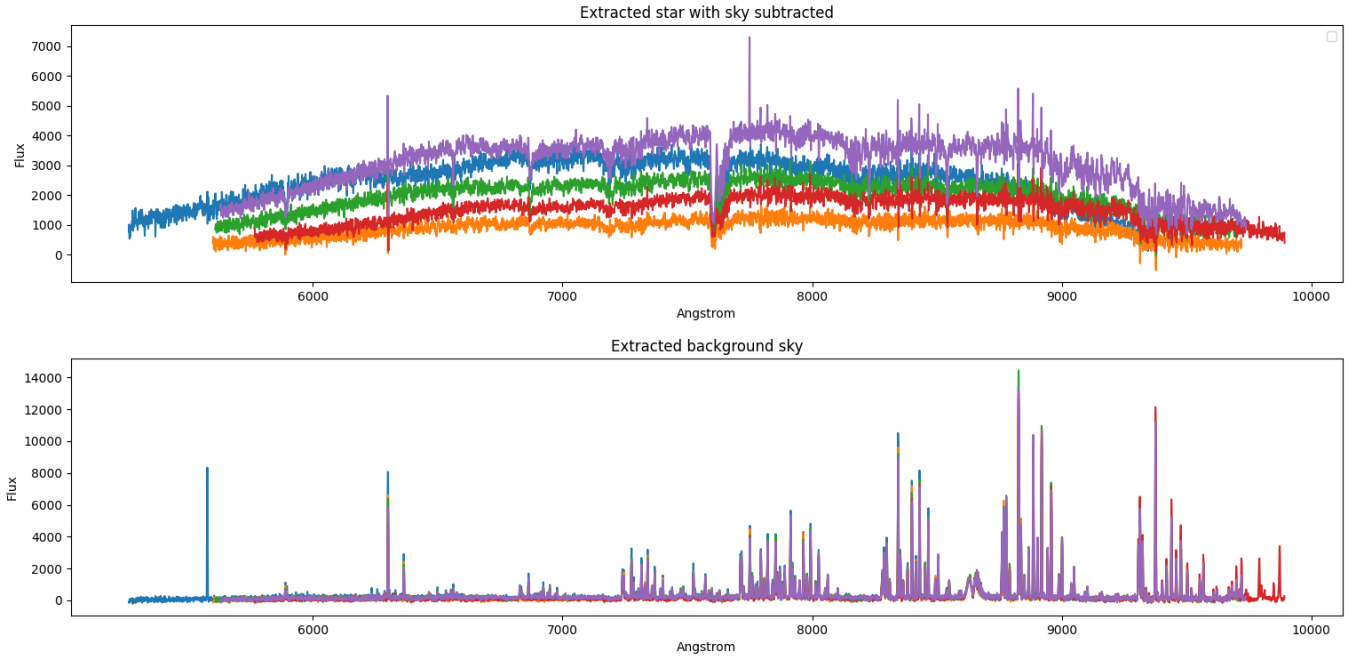


Figure 27: The extracted spectrum of each slit.

Note: The final spectrum will not appear grouped together as shown here, instead each will plotted separately before moving on to the next. We did it like this in here because of convenience.

18. CHECKING THE ACCURACY OF WAVELENGTH CALIBRATION

Users can check the accuracy of wavelength calibration by overlaying emission lines on a sky spectrum and visually examining if those lines coincide with emission lines (see figure28,29). The lines not on the sky spectrum are out of the range of our spectrum. Thus, they do not affect the validity of the calibration.

We did not include the block of code to check accuracy, as it primarily involves basic matplotlib plots. However, understanding the appearance of the spectra after saving is important. Therefore, we have included the code to read the FITS spectrum (see output 18). Make sure to change the file to the appropriate one.

Run the cell below

```
f2_1 = fits.open('spec_2d_adjust_rad_4_0000.fits')
skyline_file_path = 'pyvista/python/pyvista/data/sky/skyline.dat'
skyline_wavelengths = np.loadtxt(skyline_file_path)
f2_1.info()
```

Output 18

```
Filename: spec_2d_adjust_4_0_0000.fits
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU      68   ()
  1                1 ImageHDU         7   (4096, 1)   float64
  2  UNCERT        1 ImageHDU         9   (4096, 1)   float64
  3  BITMASK       1 ImageHDU        10   (4096, 1)   int32 (rescales to uint32)
  4  WAVE          1 ImageHDU         8   (4096, 1)   float64
  5  SKY           1 ImageHDU         8   (4096, 1)   float64
  6  SKYERR        1 ImageHDU         8   (4096, 1)   float64
```

For the rest here, we will only be showing the plots.

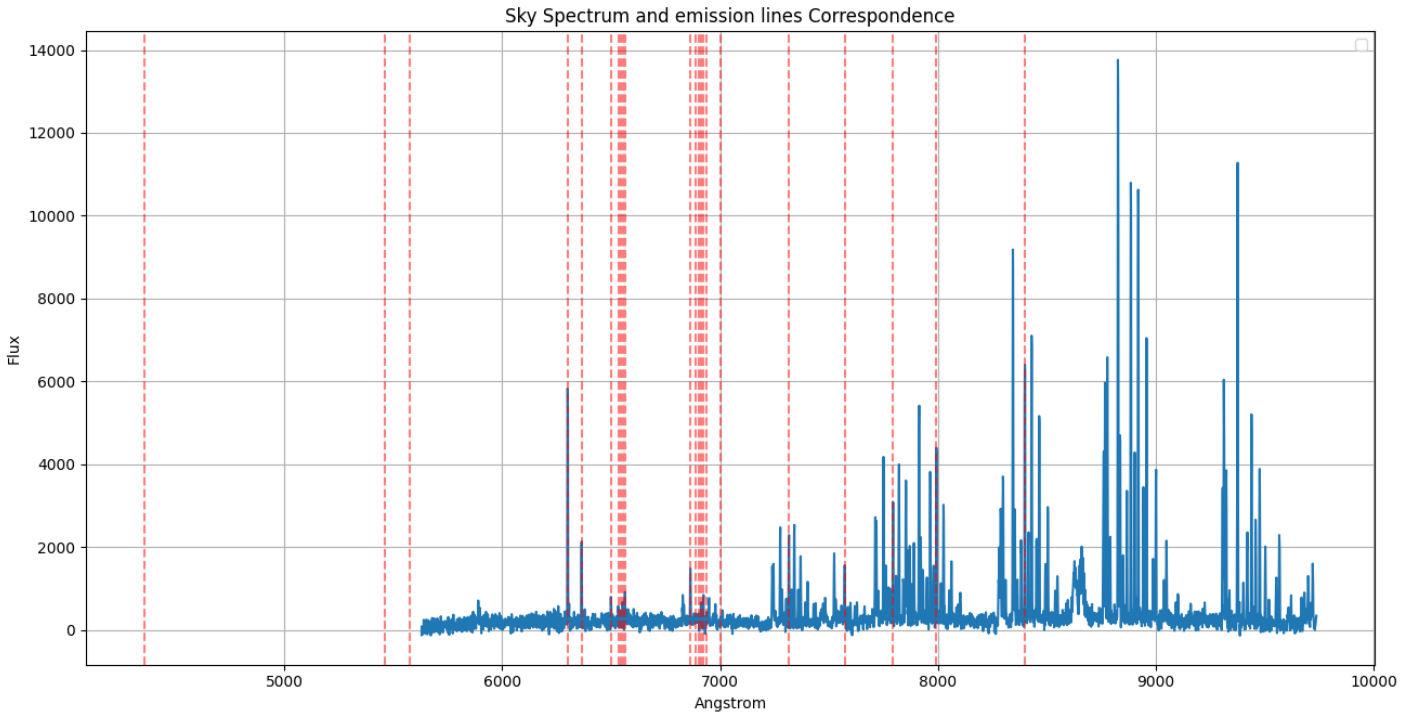


Figure 28: The emission lines are overly on the sky spectrum.

A zoomed-in visualization of figure 28. For this visualization, the user should change the x-limit in the previous cell as desired.

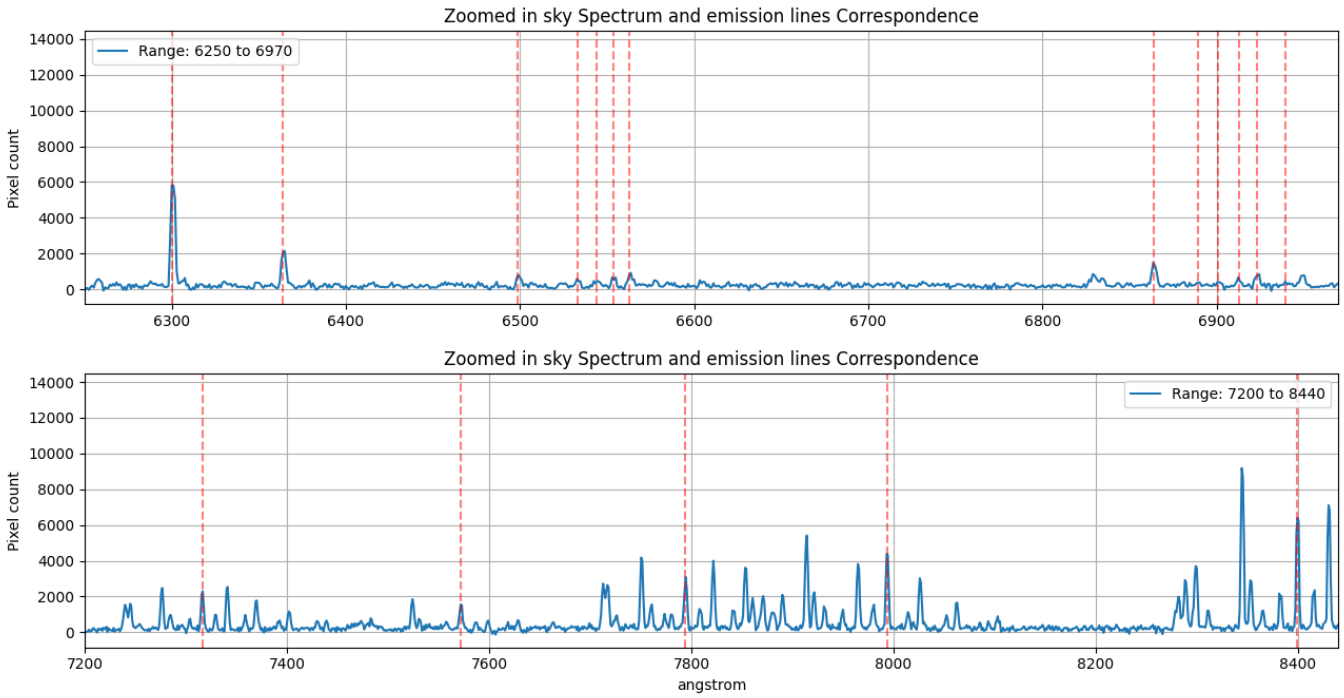


Figure 29: A visually check if lines matched an emission line.

19. SPECTRA STACKING

Important! To proceed with stacking, ensure that you have already reduced at least two exposures of the same star. If not, please do not continue beyond this point, unless you wish to read through the information.

If you have multiple exposures of the star and want to stack them to reduce the noise-to-signal ratio, you can use the code provided below.

The spectra information used to stack spectra can be found in their header.

Run the cell below

```
f02 = fits.open('spec_without_sky_17_004.fits')
f03 = fits.open('spec_without_sky_18_004.fits')
f03.info()
```

Cell description for stacking.

- Import Libraries:
  - Ensure CCDData and Combiner from the ccdproc package are imported.
- Create CCDData Objects:
  - Create CCDData objects from the flux and uncertainty data of each spectrum.
  - Assign flux values to the data parameter.

  - Assign uncertainty values to the uncertainty parameter.
  - Set the unit to 'pixel', 'pixel count', or whatever works best for you.

- Combine Spectra:

  - Put the CCDData objects into a Combiner object.

- Average Combine the Spectra:

  - Use the average_combine method of the Combiner object to average the spectra.

Run the cell below for import.

```
from astropy.nddata import CCDData
from ccdproc import Combiner
```

Run the cell below for stacking.

```
#make a ccddata object out of the flux and uncertainty of each spectra you want to combine
ccd_17 = CCDData(data = f02[1].data[0], uncertainty = f02[2].data[0], unit = 'pixel')
ccd_18 = CCDData(data = f03[1].data[0], uncertainty = f03[2].data[0], unit = 'pixel')
#put them all into Combiner (from ccdproc)
spec_17_18 = Combiner([ccd_17, ccd_18])
#then average combine (it is slightly better than median combining)
avg_test_17_18 = spec_17_18.average_combine()
#Plot the average and compare it to its originals.
plt.figure()
plt.plot(f02[4].data[0],f02[1].data[0], label= 'SEG3G2_17')
#ax1.set_title('SEG3G2_17')
plt.plot(f03[4].data[0],f03[1].data[0], label= 'SEG3G2_18')
#ax2.set_title('SEG3G2_18')
plt.plot(f02[4].data[0], avg_test_17_18.data, label = 'Stacked_spectrum')
#ax3.set_title('Combined 17 & 18')
plt.legend()
plt.xlabel('Angstrom')
plt.ylabel('Pixel count')
```

Output

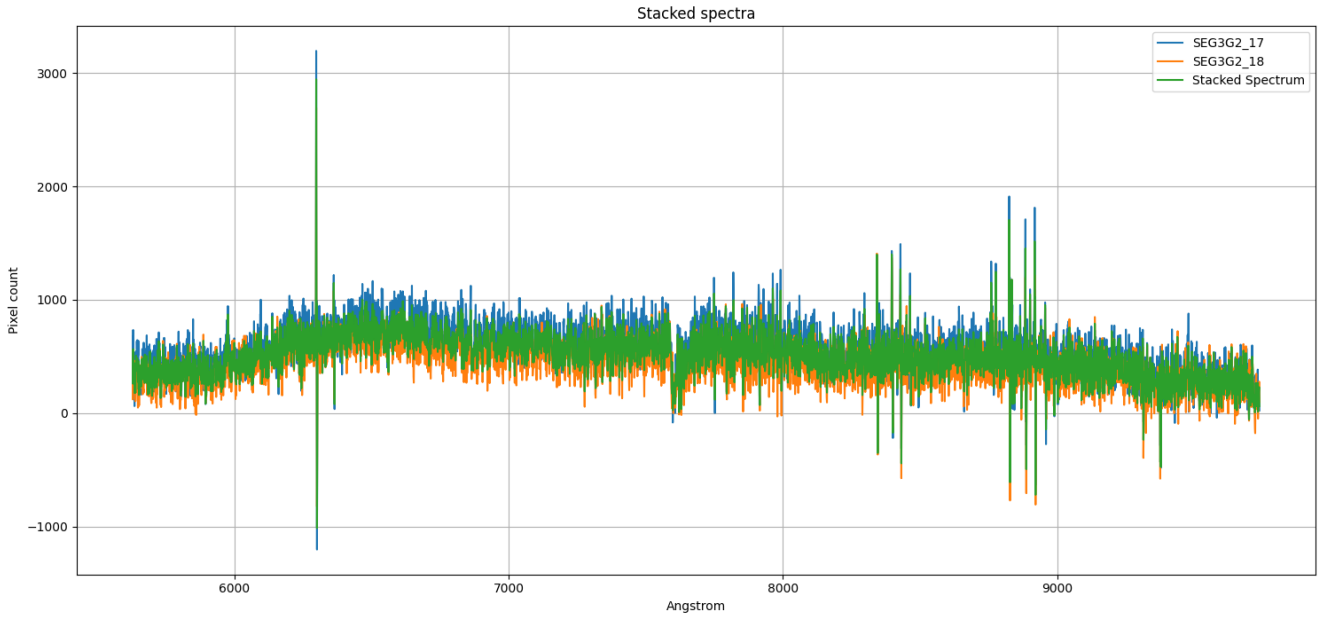


Figure 30: We combined the spectra for exposures 17 and 18 because they appeared similar and excluded exposure 20 due to its high brightness. We used the average combine and did not modify the wavelength.

Note: do not stack a good spectrum with a bad spectrum. We excluded exposure 20, which has excellent spectra compared to exposures 17 and 18.

If the User wishes to save the stacked spectra, follow the instructions below.

- Create a Primary HDU from the average combined spectrum data (avg_test_17_18).
- Create an HDU List containing the Primary HDU.
- Write the HDU List to a FITS file named 'Stacked_spec.fits'.
- Use the overwrite=True parameter to allow overwriting of the file if it already exists.

Run the cell below

```
hdu = fits.PrimaryHDU(avg_test_17_18)
hdul = fits.HDUList([hdu])

# Write to FITS file
hdul.writeto('Stacked_spec.fits', overwrite=True)
```