KCWI Data Reduction Process

This document provides step-by-step instructions for performing data reduction using the Keck Cosmic Web Imager (KCWI).

Prepared by: Saloni Agrawal

Email : saagrawal@ucsd.edu

Undergraduate Researcher

University of California, San Diego

Project with Professor Alison Coil

Step 1: Organize Data by Standard Star

Organize Data Directories

Ensure that your data directory contains data from only one standard star. Create separate directories for data from different standard stars to ensure that the chosen standard star is used for reduction. If you are not sure of which standard star to use and want to see the fits of all beforehand - you can later delete the other standard star files instead of separating them out earlier - this is talked about in a little more detail later when this can be done.

Navigate to the data directory to be used and then perform the following functions.

Step 2: Set Up the Conda Environment

They suggest Firefox in the KCWI documentation for Bokeh plots but I have tested that Google chrome also works perfectly fine.

Create a New Conda Environment

1. Open a terminal and run:

   ```
   conda create --name kcwidrp python=3.7
   ```

```
conda activate kcwidrp
```

I have not tested which other versions of python work well with the DRP - there is more information about this on the DRP webpage. However, this is one of the first common errors that could happen when running the pipeline.

Configure the Environment

2. Create a file named environment.yml in the data directory and add the following content. This is to ensure that all dependencies with the correct versions are installed:

```
name: kcwidrp
channels:
  - defaults
  - conda-forge
  - astropy
dependencies:
  - python=3.7
  - scikit-image~=0.16.2
  - astropy~=4.0
  - ccdproc~=2.2.0
  - numpy~=1.20
  - scipy~=1.4.1
  - pyerfa
  - bokeh~=2.0.0
  - jinja2~=3.0.3
  - psutil~=5.7.0
  - pandas~=1.0.3
  - matplotlib~=3.1.3
  - selenium
  - geckodriver
  - firefox
  - phantomjs
  - pyregion
  - requests
  - pytest~=5.4.1
  - cython
  - setuptools~=46.1.1
  - pip
  - pip:
    - ref_index~=1.0
    - keckdrpframework
```

If you get a “error installing pywheel” error later on when installing the kcwidrp (the next step), you have probably skipped this step or it did not run properly.

3. Run the following commands:

```
conda env update --file environment.yml

pip install kcwidrp
```

**** This information is not relevant for general purposes. Skip to Step 3 : Configure KCWI Reduction Pipeline****

(if using offshore) Initialize Bash Shell

Ensure you are using the Bash shell by running:

```
conda init bash
```

Restart the terminal.

You should see (base) in front of your username.

Step 3: Configure the KCWI Reduction Pipeline

Create and Edit Configuration File

1. Generate the configuration file: `reduce_kcwi --write_config`

2. Open the file kcwi.cfg in a text editor and modify the following parameters:

    - Set clobber = True to overwrite files.

      (You might want to vary this at several steps of the DRP depending on what you are doing. I usually set this to False when I am running the pipeline through all files together and don’t want certain files that have been already created to be recreated as that would eat up more time.)

    - Set plot_level = 1 for diagnostic plots. If you want more plots, this can be set to 2 or 3.
    - Change the default arclamp to FeAr. default_arclamp = “FeAr”

3. You also want to create separate folders for each of your configurations(ie. Differing central wavelengths) with respective files as the DRP later has trouble making MFLATS if multiple configurations are present in the same folder.

Step 4: Run Data Reduction

Option 1 : Run All Steps Together

1. **Execute the pipeline with the configuration file:**

   Blue Side :

   ```
   reduce_kcwi -b -c kcwi.cfg -f kb*.fits
   ```

   Red Side :

   ```
   reduce_kcwi -r -c kcwi.cfg -f kr*.fits
   ```

2. **Provide the pipeline with:**

   - Wavelength Ranges: Specify the <start> and <end> values.
   - Masking Ranges: Mark all absorption and emission lines as requested.
   - Approve or adjust the fits interactively as prompted by the pipeline.
   - There are some more details about this process in the standard star part of the individual steps.

Option 2 : Run Individual Steps

The syntax is similar for red and blue sides — the only difference is the use of -r (red) or -b (blue) when calling reduce_kcwi.

A great resource to understand what files are being created at each point is at the KCWI documentation page: https://kcwi-drp.readthedocs.io/en/latest/data_products.html

1. Generate Input Lists

Before starting reduction, generate lists of files for each calibration step:

```
wb kb*.fits > whatb.list   # Blue side
```

`wr kr*.fits > whatr.list  # Red side`These generate .txt files for input: bias*.txt, cbars*.txt, arcs*.txt, cflat*.txt, dflat*.txt, etc.

Each reduction step is successful if a _ingest file is created with the same base name as the list file.

---

2. Calibration Steps

Bias (BIAS-labeled files)

Creates overscan-subtracted and trimmed bias frames (intb.fits) and a master bias.

```
reduce_kcwi -r -c kcwi.cfg -l bias2x2L2U201_0.txt   # Red side

reduce_kcwi -b -c kcwi.cfg -l bias2x2TUP010_0.txt   # Blue side
```

Dark (DARK-labeled files)

Creates dark-subtracted intd.fits.

```
reduce_kcwi -b/-r -c kcwi.cfg -l dark2x2TUP010_1.0_9192.txt
```

Continuum Bars (CONTBARS)

Creates reduced intensity (int.fits), mcbars.fits (master contbars), and trace.fits.

```
reduce_kcwi -r -c kcwi.cfg -l cbars2x2MedRL_8400_5.0_1338.txt
# Red

reduce_kcwi -b -c kcwi.cfg -l
cbars2x2MedNoneBL_4600_0.7_1338.txt   # Blue
```

Arcs (ARCLAMPS)

Creates the master arc (marc.fits), geometry-corrected cubes (icube, wavemap, slicemap, etc.).

```
reduce_kcwi -r -c kcwi.cfg -l arcs2x2MedRLThAr8400_2.5_1338.txt   # Red

reduce_kcwi -b -c kcwi.cfg -l arcs2x2MedNoneBLThAr4600_20.0_1338.txt # Blue
```

> **Note: FeAr lamp is the default lamp but the config file often has default lamp as ThAr. You should change this to FeAr as it can cause errors at some point. Particularly - if you have a missing master flat when reducing objects**

Internal Flat (Cflat, ARCLAMPS-labeled)

Creates int.fits, intd.fits, and sflat.fits.

```
reduce_kcwi -r -c kcwi.cfg -l cflat2x2MedRL_8400_5.0_1338.txt # Red

reduce_kcwi -b -c kcwi.cfg -l cflat2x2MedNoneBL_4600_0.7_1338.txt   # Blue
```

Dome Flat (DOMEFLATS)

Creates int.fits, intd.fits, and sdome.fits.

```
reduce_kcwi -r -c kcwi.cfg -l dflat2x2MedRL_8400_20.0_1338.txt # Red

reduce_kcwi -b -c kcwi.cfg -l dflat2x2MedNoneBL_4600_9.4_1338.txt   # Blue
```

---

3. Standard Star Reduction

Run before reducing science objects. This step includes interactive masking and continuum fitting. Note: for interactive masking and fitting, you must set plot_level = 2 or 3 in kciw.cfg.

```
reduce_kcwi -r -c kcwi.cfg -l feige1102x2MedRL8400_1338.txt

reduce_kcwi -b -c kcwi.cfg -l bd26d26062x2MedNoneBL4600_1338.txt
```

You'll be prompted to:

- Enter wavelength range:
  New values? <float> <float> (or <cr> - done):

- Add masking regions:
  Mask line: wavelength start stop (Ang) comment <float> <float> <str> (<cr> - done):

The mask is saved in a .lmsk file. To re-run, delete the .lmsk, invens.fits, and corresponding entries in kcwir.proc / kcwib.proc.

---

4. Science Object Reduction

Reduce your science targets using the same syntax. Make sure that you only have files corresponding to the standard star you want to use for flux calibration for the science target in the redux directory and proc table. If there are multiple standard stars, the DRP chooses one on its own.

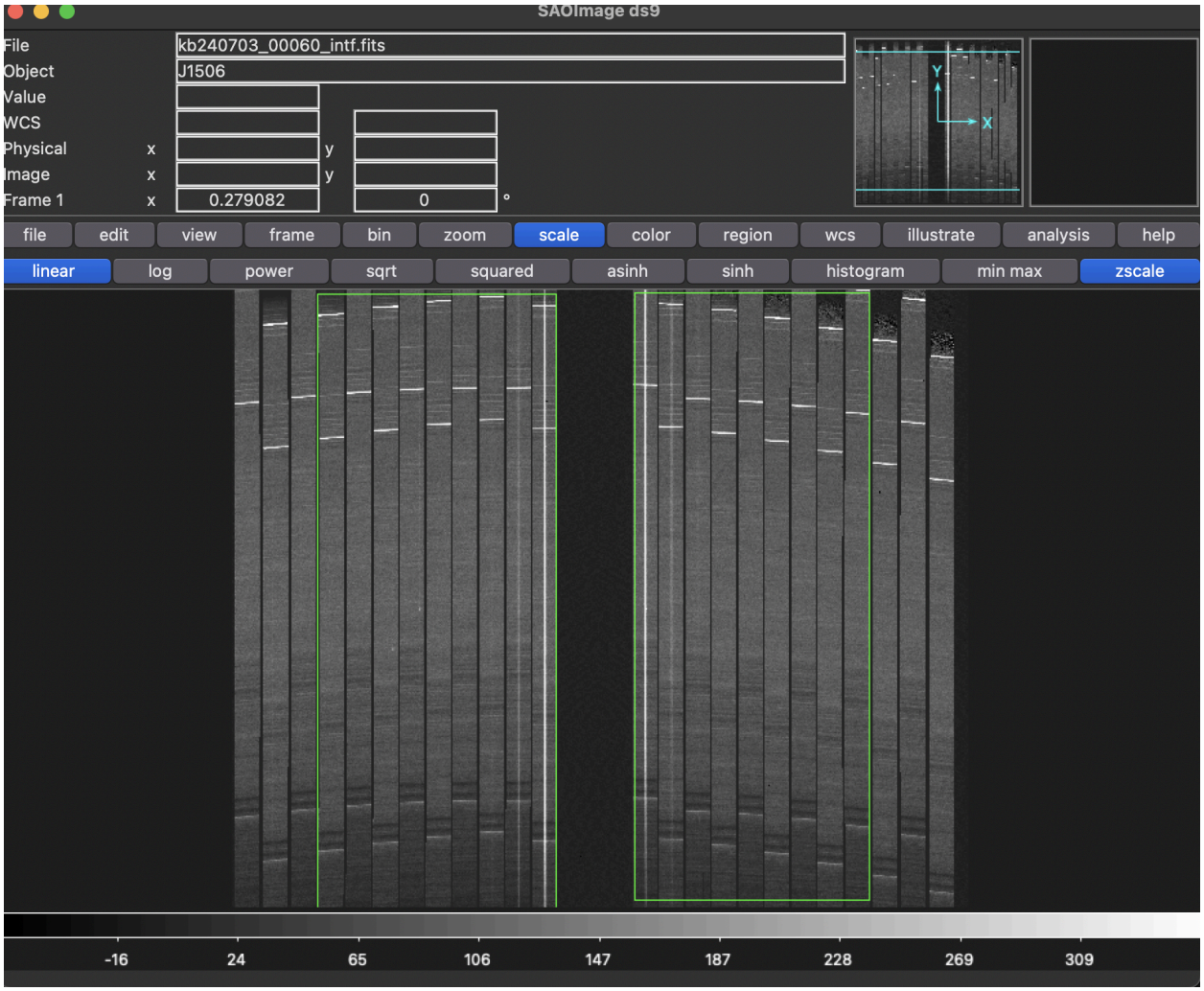
Red side

```
reduce_kcwi -r -c kcwi.cfg -l ORC42x2MedRL8400_1338.txt
```

Blue side

```
reduce_kcwi -b -c kcwi.cfg -l ORC42x2MedNoneBL4600_1338.txt
```

Manual Sky Subtraction Using DS9



1. Generate ds9.reg files for each file that you want to conduct manual sky subtraction for in the ds9 software. This is done by using the intf files and making regions to be excluded ie. mark out the object.

   Process to Make Region in DS9

   - Go to scale > zscale in the DS9 window to make object easier to see
   - Click on Region in the title bar (next to Color and WCS) and choose Region > Shape > Box
   - Click on edit > region inside the DS9 window
   - Make regions by left clicking the mouse and dragging across regions to exclude as sky
   - Now save the file by going to region > save and save the file as a .reg file in the data directory.

2. Run the following script for each file :

   `kcwi_masksky_ds9  <.intf filename> <ds9.reg filename>`

   This creates an smsk.fits file for each file

3. Create a file called kcwi.sky and write a line in the following format for each file
   `raw_object_file.fits raw_sky_frame.fits <mask.fits>`

4. Delete all file entries corresponding to the files to be manually subtracted from the kcwir.proc or kcwib.proc tables

5. Delete all sky.fits files from the redux folder

6. Rerun the reduction for each file to be re-subtracted. Make sure smsk files are in the data directory folder and not the redux folder.
   `kcwi_reduce -b -f <kb240702_00062.fits>`

Cosmic Ray Rejection for Red Side

This information is from the kcwikit > docs > Running_red_DRP.md section of the github page for KcwiKit :
https://github.com/yuguangchen1/KcwiKit/blob/master/kcwikit/docs/Running_red_DRP.md

You probably need to get KcwiKit to use these - but I chose to just get the scripts I needed for this part which was only a couple ie. kcwi_gen_log, kcrm_group_frames, kcrm_create_crmsk. If you decide to use these scripts separately without using kcwikit, you will have to use the following format for each of the steps below:
`python <scriptname>.py <everything else remains the same>`

Also, to apply the crmsk, you need to get KcwiKit version of the DRP. This can be done by replacing your RemoveCosmicRay.py script in KCWI_DRP/kcwidrp/primitives with the following script :
(Here I have taken the normal DRP script and added some lines from the KCWIKit version to ensure that the DRP uses the cosmic ray masks you have put in the redux directory instead of autoscrappy. Please check the logs to make sure that the masks are actually being used. If not, the code prints the location of where the masks are being looked for and so you can ensure that they are at the correct location)

```
from keckdrpframework.primitives.base_primitive import BasePrimitive
from kcwidrp.primitives.kcwi_file_primitives import kcwi_fits_writer


import numpy as np
import os
from astropy.io import fits
from astroscrappy import detect_cosmics
```

```
from regions import Regions  # for reading .reg recovery files


class RemoveCosmicRays(BasePrimitive):
   def __init__(self, action, context):
       BasePrimitive.__init__(self, action, context)
       self.logger = context.pipeline_logger

   def _perform(self):
       self.logger.info("Cleaning and flagging cosmic rays")
       key = 'CRCLEAN'
       keycom = 'cosmic rays cleaned?'
       header = self.action.args.ccddata.header

       exptime = header['TELAPSE']
       nshuf = header['NSHFUP']
       ttime = header['TTIME']
       if nshuf * ttime > exptime:
           exptime = nshuf * ttime

       crmskName = self.action.args.name.replace('.fits', '_crmsk.fits')
       crmskPath = os.path.join(self.config.instrument.output_directory, crmskName)
       self.logger.info(f"Looking for cosmic ray mask: {crmskPath}")
       self.logger.info(f"Input filename: {self.action.args.name}")

       if os.path.isfile(crmskPath):
           self.logger.info(f"Using CR mask: {crmskName}")
           cr = fits.open(crmskPath)
           crmsk = cr['PRIMARY'].data
           crmed = cr['MEDSCI'].data
           cr.close()

           # Check for recovery region file
           crRec = None
           crmskRecName = self.action.args.name.replace('.fits', '_crmsk_recover.reg')
           crmskRecPath = os.path.join(self.config.instrument.output_directory,
crmskRecName)
           if os.path.isfile(crmskRecPath):
               self.logger.info(f"Found recovery region: {crmskRecName}")
               with open(crmskRecPath, 'r') as f:
                   regstr = f.read()
                   if 'physical' in regstr:
```

```
                    regstr = regstr.replace('physical', 'image')
            r = Regions.parse(regstr, format='ds9')
            for region in r:
                mask = region.to_mask().to_image(crmsk.shape).astype(bool)
                crmsk[mask] = 0
                self.logger.info(f"Recovered cosmic ray region from {region}")

        # Apply the mask
        self.logger.info("Applying CR mask and replacing pixels")
        n_crs = int(np.sum(crmsk > 1e-3))
        try:
            self.action.args.ccddata.flags[crmsk > 1e-3] += 4
        except AttributeError:
            self.logger.warning("Flags array not found!")

        self.action.args.ccddata.data[crmsk > 1e-3] = 0
        self.action.args.ccddata.data += crmed

        header[key] = (True, keycom)
        header['NCRCLEAN'] = (n_crs, "number of cosmic ray pixels")
        header['history'] = f"{crmskName} cleaned cosmic rays"

    elif exptime >= self.config.instrument.CRR_MINEXPTIME:
        self.logger.info("No CR mask found — falling back to astroscrappy")
        namps = header['NVIDINP']
        bsec, dsec, tsec, direc, amps, aoff = self.action.args.map_ccd
        read_noise = 0.
        if len(amps) != namps:
            self.logger.warning("Amp count disagreement!")
        for ia in amps:
            read_noise += header.get(f'BIASRN{ia}', header.get(f'OSCNRN{ia}', 3.0))
        read_noise /= float(namps)

        sigclip = self.config.instrument.CRR_SIGCLIP
        if 'FLATLAMP' in header['IMTYPE']:
            sigclip = 10. if self.action.args.nasmask else 7.
        if 'OBJECT' in header['IMTYPE'] and header['TTIME'] < 300.:
            sigclip = 10.

        mask, clean = detect_cosmics(
            self.action.args.ccddata.data, gain=1.0, readnoise=read_noise,
            psffwhm=self.config.instrument.CRR_PSFFWHM,
```

```
            sigclip=sigclip,
            sigfrac=self.config.instrument.CRR_SIGFRAC,
            objlim=self.config.instrument.CRR_OBJLIM,
            fsmode=self.config.instrument.CRR_FSMODE,
            psfmodel=self.config.instrument.CRR_PSFMODEL,
            verbose=self.config.instrument.CRR_VERBOSE,
            sepmed=self.config.instrument.CRR_SEPMED,
            cleantype=self.config.instrument.CRR_CLEANTYPE)

        mask = np.cast["bool"](mask)
        fmask = np.where(mask)
        try:
            self.action.args.ccddata.flags[fmask] += 4
        except AttributeError:
            self.logger.warning("Flags array not found!")

        self.action.args.ccddata.data = clean
        header[key] = (True, keycom)
        header['NCRCLEAN'] = (int(mask.sum()), "number of cosmic ray pixels")
        header['history'] = "Astroscrappy used for CR cleaning"

    else:
        self.logger.info("Exposure time too short — skipping CR cleaning")
        header[key] = (False, keycom)
        header['NCRCLEAN'] = (0, "number of cosmic ray pixels")

    # Log module name
    log_string = RemoveCosmicRays.__module__
    header['HISTORY'] = log_string
    self.logger.info(log_string)

    # Optionally save intermediate
    if self.config.instrument.saveintims:
        kcwi_fits_writer(self.action.args.ccddata,
                         table=self.action.args.table,
                         output_file=self.action.args.name,
                         output_dir=self.config.instrument.output_directory,
                         suffix="crr")

    return self.action.args
```

If you still have problems with this step, I recommend following the KCWIKit version of instructions.
https://github.com/yuguangchen1/KcwiKit/blob/master/kcwikit/docs/Running_red_DRP.md

1. Make machine readable log file
   `kcwi_gen_log <night>.log -f kr*.fits`
2. Group frames for median rejection
   `kcrm_group_frames <night>.log kcrm_<night>.json -c -d -i`
   Edit the kcrm_<night>.json, remove or correct the grouping. Typically, one needs to remove the standard stars in the json file, regroup the frames if necessary
3. Run Cosmic Ray Rejection Code
   `kcrm_create_crmsk kcrm_<night>.json`
   This creates crmsk.fits for each frame as cosmic ray masks
4. Remove all the science frames in the kcwir.proc file
5. Rerun the science frames

Advanced Sky Subtraction Using KSkyWizard (WIP)

This information is from : https://github.com/zhuyunz/KSkyWizard

1. Create a conda environment :

```
conda create -n kskywizard python=3.11
conda activate kskywizard
```

2. Clone repository and install required packages

```
git clone https://github.com/zhuyunz/KSkyWizard.git
cd KSkyWizard
conda env update --file environment.yml
```

3. Install Pypeit :
   `pip install pypeit`

4. Install zap_for_kcwi in a separate directory

```
git clone https://github.com/jasonpeng17/zap_for_kcwi.git
cd zap_for_kcwi
pip install .
```

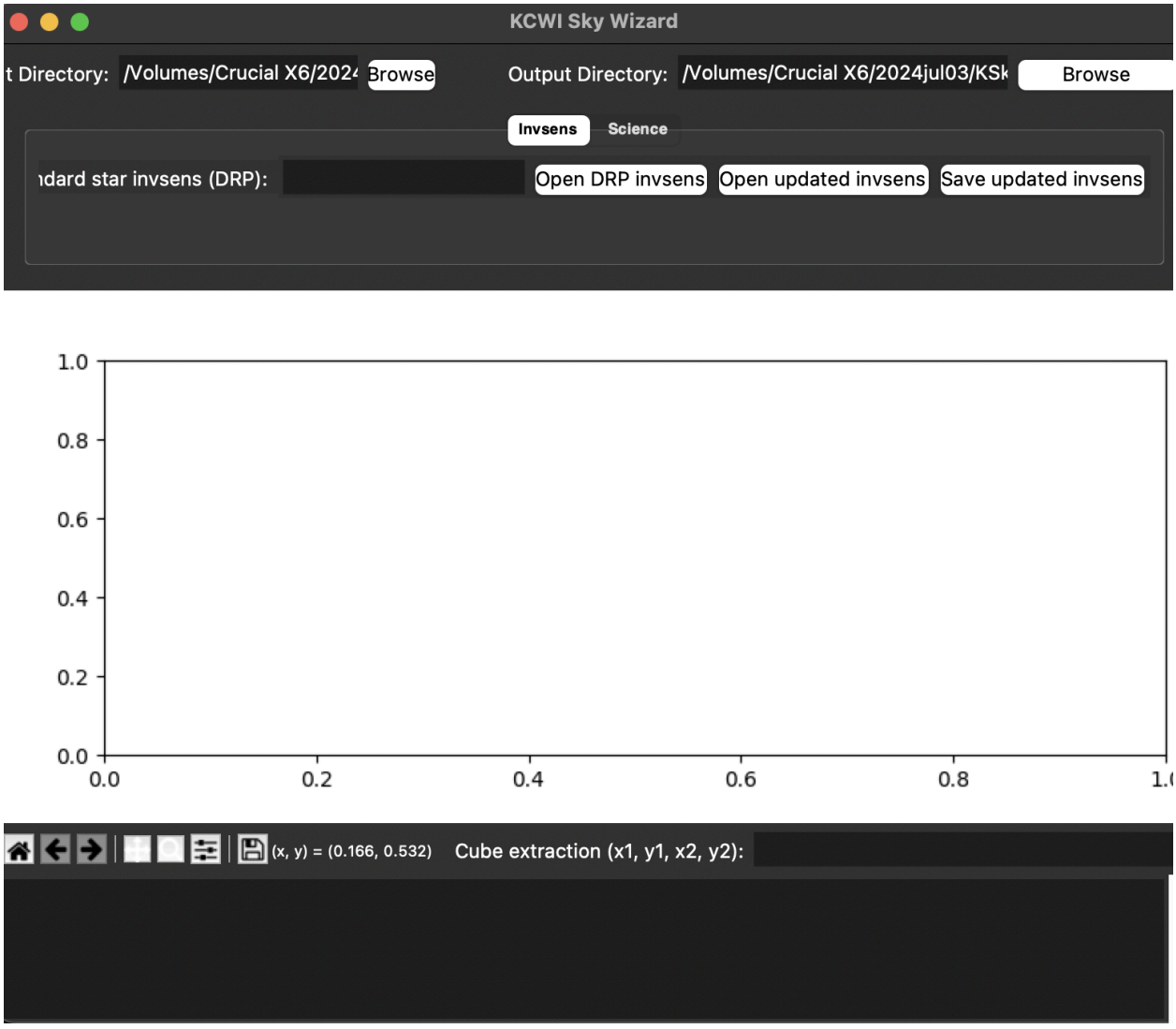
5. Return to KSkyWizard directory and run the setup code
   `cd /path/to/KSkyWizard`

```
pip install .
```

6. Download the telluric data
   ```
   pypeit_install_telluric
   TelFit_MaunaKea_3100_26100_R20000.fits
   ```

7. Configure the telluric data location :
   a. Telluric data should be stored in : `~/.pypeit/cache/download/url/.`
      Cd to this and go inside any of the files (each file is named with a lot of numbers and alphabets eg. 36c0f27b37e580464a881e2c44f66eb7
   b. Each of these sub-files contains :
      i. A contents file
      ii. A url file
      Take note of the full path of the contents file
   c. Locate the installed version of the processing.cfg file.
      ```
      kskywizard --config_path
      ```
      This will return the path of the processing.cfg file. However, this usually does not work for me so I just run kskywizard like in step 8 and the very first line shows me where the processing.cfgfile being used is stored. I then go to this file path and edit the telgridfile variable like in step 7 part d (ie the next part)
   d. Modify the telgridfile variable in the file and point it to the content path from above :
      ```
      telgridfile =
      ~/.pypeit/cache/download/url/5f17ecc1fcc921d6ec01e18d9
      31ec2f8/contents
      ```
8. Start KSkyWizard :
   ```
   conda activate kskywizard
   kskywizard
   ```

This should show a GUI window like below



9. Set the input and output directory:
   Input directory is the redux directory where the DRP output is stored
   Output directory should be separate from the redux directory to avoid overriding the DRP outputs
10. Flux Calibration
    a. Browse and select the *_invens file to use from the open DRP invens button
    b. To refine the sensitivity curve, click on the plotting canvas to activate the interaction mode:
        i. Use the following keyboard shortcuts to adjust regions:
            1. e : Exclude a region from the fit
            2. i : Include a region in the fit
            3. a : add a spline knot
            4. d : delete a spline knot

   ii. Press f to re-fit the sensitivity function iteratively until the model cyan line aligns well with the data
   iii. Press b to revert back to the DRPs curve
c. Fit the Telluric Function
   i. Press t to generate the telluric correction model. This takes some time so wait until the output appears :
d. Once satisfied with the sensitivity curve and telluric fit, press the Save updated invsens button to save the refined data. These updated curves can be reloaded to the tool using the Open updated invsens button for future usage.

11. Reducing Science frames using updated sensitivity curve + sky subtraction :
    a. Switch to the science panel (after selected the updated saved invsens file in the invens panel) and enter the science image number and press enter. Eg : for file kb240702_00060.fits just type 60 and press enter.
    b. f this is the first time that you reduce this frame, please press Load Raw Cube, Save Cropped Cube and Load Cropped Cube. Otherwise, simply press Load Cropped Cube to load the data. This step is simply to crop the datacube outside of the good wavelength region to avoid running into edge problem with ZAP.
    c. If you plan to use the in-field sky, you need to mask out the science target. You can check the _wlimg.fits in the output directory, mask the source using DS9, and save the DS9 region as kr230923_00085.reg for frame kr230923_00085. After that, enter the mask frame number, press enter, and press Update ZAP mask button. You can update ZAP mask multiple times whenever you press the button
    d. Now we run ZAP :
       i. If you want to use the default ZAP setup, please uncheck Use multiple skyseg in ZAP and Additional Sky Seg near Halpha, and press Run ZAP. In this case, it adopts one sky segment and fit the overall variation across the entire wavelength. By default, it adopts cfwidth=300 to remove the science signal. For more information related to the cfwidth, please check the github page of the modified version and the original ZAP paper by Soto et al. (2016). To proceed, please type 'q' in the text widget like the example below and press enter to start ZAP.
       ii. We find that using multiple sky segment sometimes can reduce the sky residual. This approach was adopted in the original ZAP by Soto et al. (2016) when it first came out, but the updated version switched to one sky segment. For how the sky segment is chosen, please refer to Soto et al. (2016) for more details. If you want to use this option, please check Use multiple skyseg in ZAP. Besides, we find that if a very strong emission line happens to fall in a sky segment with relatively sparse sky lines (e.g., Halpha emission), a large cfwidth would aritificially introduce more noise in the sky-subtracted spectrum. By checking Additionally Sky Seg near

Halpha and setting the redshift, the code would generate an additional segment in the region of Halpha with cfwidth=5. This can also lead to out of range errors sometimes so if you get an error of that sort, uncheck this box.

iii. If you are happy with this segment, type 'q' in the text widget to proceed. Otherwise, you can update the segment in the text widget directly to whatever regions you want to use, and type 'u' to update the change. Every tuple indicates (the start of the segment, the end of the segment, cfwdith). You can either delete a tuple or add a new one, or simply changing some of the values in it.

iv. Now the text widget would print out the updated sky segment. Type 'q' if you wanna proceed, or 'r' if you want to reset the sky segment.

Reduction Document Ends Here

Step 5: Resample Data

1. Go to the directory containing `cubea.fits` files.
2. Run the following command for each file:
3. IDL> `ifsr_kcrmresample, '/home/saagrawal/mosaicsred/kr240702_00100', '/home/saagrawal/mosaicsred/ResampledCubes/kr240702_00100_cubea.fits', [6, 16, 29, 81]`

   Format : ifsr_kcrmresample,input file,output file,goodreg

Input Parameters for Resampling

- **instr**: Base filename for input KCWI FITS data cubes.
- **outfile**: Filename for the output resampled cube.
- **goodreg**: 4-element array specifying the spatial region of interest `[xmin, ymin, xmax, ymax]`.

Step 6: Peak Detection and Header Updates

Detect Peaks

Run the following for peak detection: peak = IDL>
IDL>`ifsr_peak('/home/saagrawal/Mosaics/ResampledCubes/kb240702_00052_cubea.fits', [4000, 5500], xranf=[20, 55], yranf=[10, 50], datext=-1, varext=1, dqext=2)`

IDL>`PRINT, peak[0] ; X-coordinate of the peak`

IDL>`PRINT, peak[1] ; Y-coordinate of the peak`

Format : ifsr_peak(input file,[4000,5500],xranf=[20,55],yranf=[10,50], datext = -1, varext = 1, dqext = 2)

Update FITS Headers

Use Python to add the peaks to the file headers:

```
python add_xypeak.py kb240702_00068_cubea.fits 30.527734 34.805140
```

Replace `kb240702_00068_cubea.fits` and the peak values with your file and results.

Step 7: Rotation

Run the following to rotate files:

```
python rotate.py kb240702_00065_cubea.fits <(minus)*degree of
rotation>
```

Step 8: Create a Mosaic

Run the following to create a mosaic from multiple FITS files:

IDL> `ifsr_nmosaic, ['cubea.fits', 'cubeb.fits', 'cubec.fits'], '/data/home/wning/project/pg1700/reduced/mosaic/pg1700.fits', /nophu`

Format : ifsr_mosaic,input files,output file,,/nophu

Flat Field Correction

- Use an internal arc lamp to illuminate the screen uniformly. Since light is not distributed equally to all pixels, this fixes the unequal illumination.
-