WIRC+Pol DRP Documentation

Max Millar-Blanchaer, Kaew Tinyanont and others

Sep 09, 2022

CONTENTS:

1 Installation **1**

2 Quick Start **3**
2.1 Steps of Operation . . . 3

3 Basic Calibration **5**

4 Spectral Extraction **7**
4.1 Inputs for Spectral Extraction . . . 7
4.2 Key Options for Spectral Extraction . . . 7
4.3 Steps of Operation . . . 8

5 Tutorial 1: Single File Data Reduction **9**
5.1 Pipeline Description . . . 9
5.2 Assumptions . . . 9
5.3 This Tutorial . . . 9
5.3.1 Step 0 - Make your calibration files . . . 9
5.3.2 Step 1 - Read in and calibrate your data . . . 11
5.3.3 Step 2 - Extract the thumbnails of the spectral traces . . . 13
5.3.4 Step 3 - Extract the spectra from each thumbnail . . . 15
5.3.5 Step 4 - Calculate the polarized spectra . . . 16
5.3.6 Step 5 - Save the wirc object . . . 17

6 Tutorial 2: Dataset Reduction Tutorial **19**
6.1 This Tutorial . . . 19
6.2 The Recipe . . . 19
6.2.1 Step 0 - Read in the calibration files . . . 20
6.2.2 Step 1 - Read in and calibrate your data . . . 20
6.2.3 Now it's your turn to set up a loop to calibrate the brown dwarf data! . . . 23
6.2.4 Step 2: Source identification . . . 25
6.2.5 Step 3-4: Background subtraction and spectral extraction . . . 27
6.2.6 Now with Steps 2-4 demonstrated for one file, let's run it all on all files: First with the unpolarized standard star . . . 29
6.2.7 Then the brown dwarf . . . 30
6.2.8 Step 5 - Stack the spectra . . . 33
6.2.9 Now do the same for the brown dwarf . . . 36
6.2.10 Step 6: Polarization calibration and calculation . . . 40
6.3 Acknowledgement . . . 44

7 Tutorial 3: Modulation Sequence Tutorial **45**
7.1 Step 1: Read in the files . . . 45

7.1.1 Read in the files - Tutorial Dataset . . . 45
7.1.2 Read in the files - Normal Operations . . . 46
7.2 Step 2: Plot the spectra and make sure they're aligned and ready to go . . . 47
7.3 Step 3: Calculate *q* and *u* for each pair of HWP positions. . . . 48
7.3.1 Now calculate *p* and *:math: `theta`* . . . 48
7.4 Step 4: Calibrate q and u . . . 50

8 Tutorial 4: Convenience Functions Tutorial **55**
8.1 Dataset Reduction Functions . . . 55
8.1.1 reduce_dataset . . . 55
8.1.2 reduce_ABAB_dataset . . . 57
8.1.3 reduce_dataset_distance . . . 57
8.2 Plotting Functions . . . 57
8.2.1 Extraction Sumary . . . 58
8.2.2 Polarization Summary . . . 58
8.2.3 Variability Summary . . . 59
8.2.4 Broadband Summary . . . 60

9 Indices and tables **63**

CHAPTER

ONE

INSTALLATION

The WIRC_DRP can be found in a public Github Repository.

There are several dependecies for the pipeline, but if you follow the install instructions on Github then the only one you'll have to install manually is: image_registration. Please pay attention to their citation request.

CHAPTER

TWO

QUICK START

The Data Reduction Pipeline (DRP) of WIRC+Pol is a modular, object-oriented software designed to reduce data taken by the WIRC+Pol infrared (IR) spectropolarimeter on the 200-inch Hale Telescope at Palomar Observatory. It is also designed to be adaptable to reduce data from other multi-channel spectropolarimeter.

There are two classes of objects the DRP uses: `wirc_data` and `wircpol_source`. Each `wirc_data` object encapsulate one image recorded by WIRC+Pol including the full image, all header informations, and its calibration and reduction history. For each `wirc_data` object, there may be a list of `wircpol_source` objects, each of which describes a source in the `wirc_data` object. This object contains the position of the source along with cutouts of the source's four spectral traces. It may also include spectra and polarization extracted for the source. To reduce WIRC+Pol data, the DRP loads the raw image in as a `wirc_data` object, then operate on it through the following steps to produce polarization spectra.

2.1 Steps of Operation

1. *Basic Calibration*: a science image is dark subtracted and flattened using dark and flat frames taken on the same night, or on a night close to the science observation. This step is performed by the method `wirc_data.calibrate`.
2. background-subtraction: a calibrated science image is then background subtracted. This is a crucial step to remove polarimetric biases from the bright IR sky emission. There are several methods, but generally they rely on a library of background frames taken with some offsets from the science observations. If the slit is used, the source is typically observed at two dither positions inside the slit (AB) so images from one position can be used as background for another position. The method for generating background file for subtraction is `wirc_data.generate_bkg`.
3. Source Finding: after background subtraction, one can either run automatic source finding using the method of `wirc_data.find_sources_v2`, or manually specify the location of the source to be extracted using the `add_source` method. Either way, this will create `wircpol_source` objects associated with that `wirc_data` object. The `get_source_cutouts` method is run to retrieve small thumbnails of the four spectral traces for extraction.
4. *Spectral Extraction*: once the background image and the location of the source(s) are defined, the `extract_spectra` method can be run on each `wircpol_source` object to retrieve four flux spectra from the upper left, lower right, upper right, and lower left spectral traces of a source. The output spectral cube, which is the `trace_spectra` attribute of the `wircpol_source` object, is an array of shape [4, 3, spectral_length] where the first index is the four spectra, the second index is (wavelength, flux, flux_uncertainty), and the last index is the spectrum. For example, the upper left flux is wircpol_source.trace_spectra[0,1,:]. Wavelength calibration can be obtained by the `rough_lambda_calibration` method. Note that this step is sometimes unreliable, and the resulting wavelength solution should be verified.
5. polarization-calculation: and Calibration: this calculation is performed after all science images in the dataset have been reduced and extracted. The calculation is performed on the datacube of all extracted spectra (dimensions

[n_images, 4, 3, spectral_length], where n_images is the number of images in the observation sequence. In addition, we also need an array of the half-wave plate angle (with length equals to n_images). The calculation can be performed by using the function `compute_qu_for_obs_sequence` in `source_utils.py` on the spectra cube and the HWP angle array. Finally, the normalized Stokes parameters from `compute_qu_for_obs_sequence` can be calibrated using the `calibrate_qu` function in `calibration.py`.

For the example of these steps, see the Tutorial notebooks. Functions to perform steps 2-5 automatically are in `dataset.py`.

CHAPTER

THREE

BASIC CALIBRATION

The first step of the DRP is to apply dark subtraction and field flattening to science images. Once an image is loaded as a `wirc_data` object, the calibration is performed by the method `wirc_data.calibrate`. To do this, we need the following calibration files.

1. Master dark frame. This is a median-combined dark frames taken at the same exposure time and coadd as the science image. In case of exposure time mismatch, the DRP will automatically scale the master dark. However, this is much less reliable at subtracting numerous hot pixels in WIRC data. We typically use about 21 dark images to construct the master dark frame. This is done using the `wirc_drp.calibration.masterDark` function.
2. Master flat frame. This is a median-combined flat frames taken using the same filter as the science image. The best practice is to use flat with the PG and the focal plane mask out of the beam. The half-wave plate is also typically removed. Twenty one 1-second exposures with the low lamp is sufficient. This is done using the `wirc_drp.calibration.masterFlat` function.
3. Bad pixel map. In addition to the master flat, the `wirc_drp.calibration.masterFlat` function also produces a bad pixel map by finding pixels that deviate significantly in gain from the rest of the array. This map is used later on in the reduction process to mask them out of spectral extraction.

The basic calibration steps are shown in *Tutorial 1: Single File Data Reduction*.

CHAPTER

FOUR

SPECTRAL EXTRACTION

In this step, the 2D spectrum in WIRC+Pol data is extracted into 1D spectrum. The spectral extraction step is performed by the method `wircpol_source.extract_spectra`, which directly calls the function `spec_utils.spec_extraction`. See *Tutorial 1: Single File Data Reduction* and *Tutorial 3: Modulation Sequence Tutorial* for examples of how to run this function. Here we describe the inputs and each step of spectral extraction.

4.1 Inputs for Spectral Extraction

1. Science thumbnails: an array with shape [4, width, width] with four small thumbnails of the four spectral traces
2. Background thumbnails: same-shape array with background thumbnails
3. Data quality thumbnails: same-shape array containing bad pixel information

4.2 Key Options for Spectral Extraction

1. Optimal extraction (`method = 'optimal_extraction'`): the optimal extraction algorighm of Horne 1986. This algorithm weighs the significant of each pixel by its signal to noise ratio, thereby suppressing noise from low SNR pixels. This is the default extraction algorighm.
2. Simple extraction (`method = 'sum_across_trace'`): in this simpler method, all pixels in the spatial direction are simply summed with equal weight. It may produce noisier 1D spectrum, but suffers less biases from the weighing algorithm.
3. Bad pixel masking (`bad_pix_masking`): The optimal extraction algorighm allows us to easily mask bad pixels out from spectral extraction.
4. Extraction Range (`spatial_sigma`): The DRP automatically detect the trace inside each thumbnail and measure its spatial profile. User can select how many "sigmas" in each direction to extract the spectrum. For example, if the DRP determines that the trace has a Gaussian profile with 2 pixel standard deviation, `spatial_sigma = 5` (default) will extract the spectrum +-10 pixels from the center of the trace.

4.3 Steps of Operation

1. Find Spectral Trace: the function `wirc_drp.image_utils.findTrace` is called to find the trace within each thumbnail. This function a Gaussian profile to each column in the thumbnail to find the peak pixel. It then fits a straight line through it to find the trace. It returns the location, the first estimate of the width of the trace, and the angle of the trace.
2. Rotate Spectral Trace: WIRC+Pol produce spectral traces that are about 45 degrees with respect to the pixel grid. To extract the spectra, we first rotate this to align with the pixel grid. The angle used is either what measured by the previous step, or what supplied by the user via a keyword `trace_angle`, which is a list of four angles in degree of the four traces (per convention, the order of the four traces are always upper left, lower right, upper right, lower left). The default rotation algorithm used is openCV bicubic interpolation. (To be consistent, the rotation method can only be changed by adjusting the DRP code. See the `wirc_drp.spec_utils.frame_rotate` function for the keywords.)
3. Determine Extraction Range: After rotation, the function `image_utils.traceWidth_after_rotation` is called to recompute the trace width. Then the function `spec_utils.determine_extraction_range` is called to robustly find the center of the trace in the rotated thumbnail, and determine the range (in pixel) in the spatial direction over which the spectrum is extracted.
4. Run Spectral Extraction: Once the thumbnails are prepared and the extraction range determined, the `spec_utils.optimal_extraction` or `spec_utils.sum_across_trace` is called to extract the spectrum. The output from the function are: array of extracted spectra, array of flux standard deviation, spectral trace widths, spectral trace angle, and the thumbnails used for extraction (background subtracted). The DRP incorporates these outputs into the `wircpol_source` object from which the `extract_spectra` method was called.

CHAPTER

FIVE

TUTORIAL 1: SINGLE FILE DATA REDUCTION

Welcome to the WIRC+POL data reduction pipeline (DRP) tutorial. The notebook will give you a basic run through of how to reduce WIRC+POL data and give you some details about the setup of the pipeline.

5.1 Pipeline Description

The pipeline has been designed so that once read-in, each data file will be held in an object called a *wirc_data* object. Each *wircpol_data* object will contain, the raw (or calibrated) 2D data itself, relevant information about the data and a number of functions that will calibrate and process the data. It will also contain a list of *wircpol_source* objects, where each *wircpol_source* will correspond to a source in the field. The source object will contain thumbnails of the spectral traces, as well as the extracted spectrum and polarized spectrum for each source.

Note: The current implementation has been designed with WIRC+POL data in mind, but we hope to extent this to WIRC+Spec data soon.

5.2 Assumptions

We assume you have installed the DRP following the instructions in the github README (including the installation of all the dependencies): https://github.com/WIRC-Pol/wirc_drp

5.3 This Tutorial

This tutorial will give an example of reducing a raw WIRC+POL data image from start to finish, and will give you an idea of the implementation of the pipeline along the way. We'll note that many of the data reduction steps are works in progress. The purpose of this tutorial is more to demonstrate the features of the pipeline rather than to teach you how to fully calibrate your data.

5.3.1 Step 0 - Make your calibration files

The very first step will be to create your own master darks and master flats. For the sake of this tutorial we have already done this for you, but we provide some example code commented-out below so you can do it yourself. Already-made master darks and flats have been provided in the tutorial directory.

[30]:
```
#Import the important things
%matplotlib inline
import wirc_drp
```

(continues on next page)

(continued from previous page)

```
import wirc_drp.wirc_object as wo
from wirc_drp.utils import calibration
import matplotlib.pyplot as plt
import numpy as np

import warnings; warnings.simplefilter('ignore')
```

Create a master dark

First you will need a list of files that will go into the master dark. It can be created like this:

```
$ ls dark*.fits > dark_list.dat
```

Then you read it in and create your mater dark by taking the median of all the files in the list. A hotpixel map will also be generated by default.

[28]:
```
### Get the file list
# import astropy.io.ascii as asci
# import os
#os.chdir('/hcig1-nas/wircpol/data/20170611')
#dark_list_fn = "dark_list.dat" #The file name for your list of dark files
#dark_list1s = (asci.read(dark_list_fn, format = 'fast_no_header'))['col1'] #Read in the␣
↪list

### Create the master dark and a bad pixel map.
### This function creates a new fits file based on the last filename in your dark_list␣
↪and appends "_master_dark.fits"
### The hot pixel map will be the same, except with a "_bp_map.fits" suffix.
#dark_name1s, bp_name = calibration.masterDark(dark_list1s) # The output of this␣
↪function is a the filename
                                                            # of the master dark and bad␣
↪pixel maps
```

Create a master flat

First you will need a list of files that will go into the master flat. It can be created like this:

```
$ ls flat*.fits > flat_list.dat
```

Then you read it in and create your mater flat by subtracting the master dark from each image, taking the median of all the files in the list and then normalize by either the median (default) or the mode, set by the *normalize* keyword. A bad pixel map will also be generated by default, which you can supplement with the hot pixel map output by creating the master dark.

[29]:
```
### Get the file list
# flat_list_fname = "flat_list.dat" #The file name for your list of flat files
# flat_list = (asci.read(flat_list_fname, format = 'fast_no_header'))['col1']

### Create the master dark and a bad pixel map. You will need the filename
### This function creates a new fits file based on the last filename in your flat list␣
↪and appends "_master_flat.fits"
```

(continues on next page)

(continued from previous page)

```
# flat_name, bp_name = calibration.masterFlat(flat_list, dark_name1, hotp_map_fname =␣
↪None)
```

5.3.2 Step 1 - Read in and calibrate your data

Assuming that you now have a master dark and master flat file, we can now read in a wirc_data file and perform the calibration.

[4]:
```
#First we'll set up all the directories and filenames:
wircpol_dir = wirc_drp.__path__[0]+"/../"
tutorial_dir = wircpol_dir + "Tutorial/sample_data/Single_File_Tutorial/"

raw_fn =tutorial_dir+"wirc1586.fits"
flat_fn = tutorial_dir+"wirc2012_master_flat.fits"
dark_fn = tutorial_dir+"wirc0141_master_dark.fits"
bp_fn = tutorial_dir+"wirc2012_bp_map.fits"
```

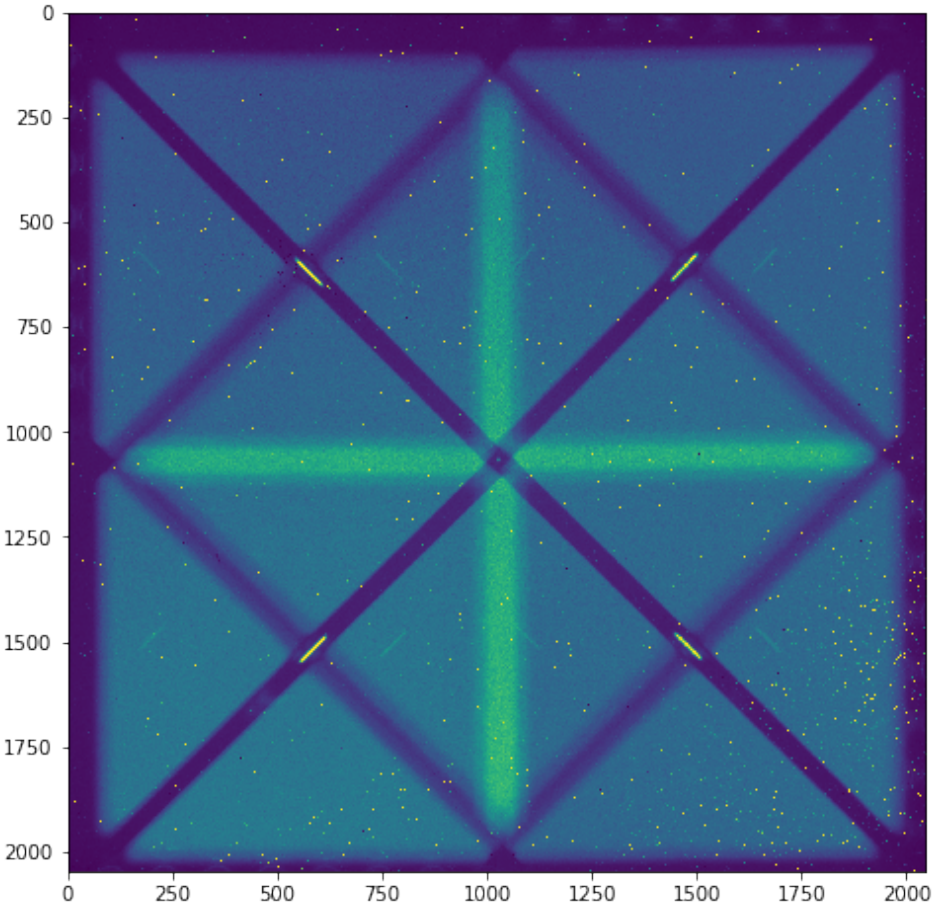
[5]:
```
#Now we'll create the wirc_data object, passing in the filenames for the master dark,␣
↪flat and bad pixel maps
raw_data = wo.wirc_data(raw_filename=raw_fn, flat_fn = flat_fn, dark_fn = dark_fn, bp_fn␣
↪= bp_fn,verbose=True)
```

```
Creating a new wirc_data object from file /Users/maxwellmb/Dropbox (Personal)/Library/
↪Python/wirc_drp/wirc_drp/../Tutorial/sample_data/Single_File_Tutorial/wirc1586.fits
Found a J-band filter in the header of file /Users/maxwellmb/Dropbox (Personal)/Library/
↪Python/wirc_drp/wirc_drp/../Tutorial/sample_data/Single_File_Tutorial/wirc1586.fits
```

The wirc_data object holds the 2D data image in the property full_image. We can take a look at it now:

[6]:
```
plt.figure(figsize=(8,8))
plt.imshow(raw_data.full_image, vmin=0, vmax=5000)
```

[6]:
```
<matplotlib.image.AxesImage at 0x1c2fe1ce48>
```



nbsphinx-code-borderwhite

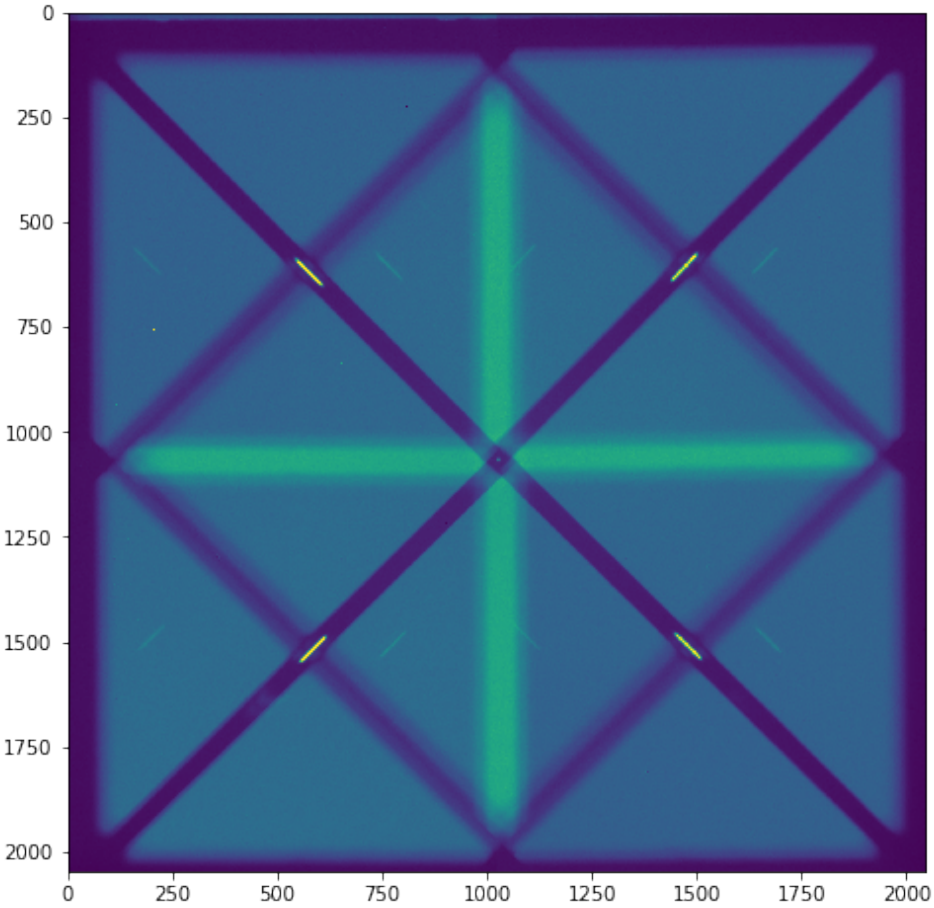
We can see (even in this zoomed out image) that there are a bunch of bad pixels. Let's run the calibration step. It will subtract the dark, divide by the flat, and interpolate over the bad pixels.

[7]:
```
raw_data.calibrate()
```

```
/Users/maxwellmb/Dropbox (Personal)/Library/Python/wirc_drp/wirc_drp/wirc_object.py:229:␣
↪RuntimeWarning: divide by zero encountered in true_divide
  self.full_image = self.full_image/master_flat
/Users/maxwellmb/Dropbox (Personal)/Library/Python/wirc_drp/wirc_drp/wirc_object.py:229:␣
↪RuntimeWarning: invalid value encountered in true_divide
  self.full_image = self.full_image/master_flat
/Users/maxwellmb/Dropbox (Personal)/Library/Python/wirc_drp/wirc_drp/utils/calibration.
↪py:736: RuntimeWarning: invalid value encountered in less_equal
  bad_pixel_map = np.logical_or(bad_pixel_map, redux_science <= 0)
```

[8]:
```
plt.figure(figsize=(8,8))
plt.imshow(raw_data.full_image, vmin=0, vmax=5000)
```

[8]:
```
<matplotlib.image.AxesImage at 0x1c2ffafb00>
```



nbsphinx-code-borderwhite

There are a few remaining artifacts in the image, but overall, not bad!

We're now done with calibration and so we want to save our newly calibrated file. Up until now these steps can apply to both spec mode and pol mode data. However, the rest of the tutorial will just demonstrate how to reduce POL data.

[12]:
```
raw_data.save_wirc_object(tutorial_dir+"calibrated.fits",verbose=True)
```

```
Saving a wirc_object to /Users/maxwellmb/Dropbox (Personal)/Library/Python/wirc_drp/wirc_
↪drp/../Tutorial/sample_data/Single_File_Tutorial/calibrated.fits
```

5.3.3 Step 2 - Extract the thumbnails of the spectral traces

In this step we will create s wircpol_source object and append. Each source object will contain cutout thumbnails of each spectral trace, and eventually spectra and polarized spectra. We will then append the wircpol_source object to the wirc object's "source_list".

[13]:
```
#First we'll create a new data object, mostly just to demonstrate how to read in an␣
↪existing wirc_data object.
calibrated_data = wo.wirc_data(wirc_object_filename=tutorial_dir+"calibrated.fits")
```

```
Loading a wirc_data object from file /Users/maxwellmb/Dropbox (Personal)/Library/Python/
↪wirc_drp/wirc_drp/../Tutorial/sample_data/Single_File_Tutorial/calibrated.fits
Didn't find any background files
```

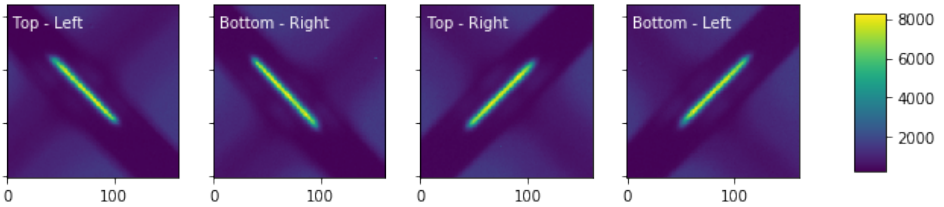
[14]:
```
calibrated_data.generate_bkg()
```

Create a wircpol_source object

[15]:
```
#This specific object will be for a souce in the slit (with coordinates [1063,1027]).
wp_source = wo.wircpol_source([1063,1027],'1',0) #The second argument indicates that␣
↪this source is in the middle slit
```

We'll now get the trace cutouts for this source

[16]:
```
wp_source.get_cutouts(calibrated_data.full_image, calibrated_data.DQ_image, 'J',
↪calibrated_data.bkg_image)
wp_source.plot_cutouts(figsize=(10,6))
```



nbsphinx-code-borderwhite

Now let's add this source to the calibrated_data source list.

[17]:
```
calibrated_data.source_list.append(wp_source)
calibrated_data.n_sources += 1
```

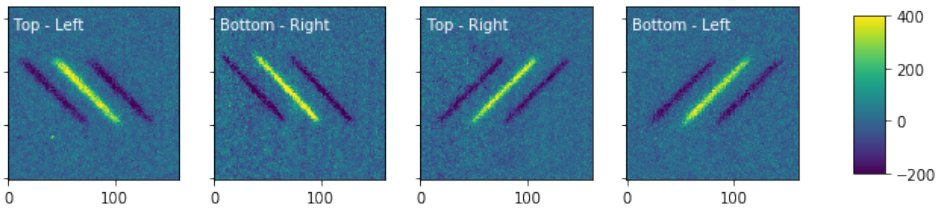
We'll now manually add another source in the field, that isn't in the slit

[18]:
```
calibrated_data.source_list.append(wo.wircpol_source([1035,640],'slitless',calibrated_
↪data.n_sources+1))
calibrated_data.n_sources += 1
calibrated_data.source_list[1].get_cutouts(calibrated_data.full_image, calibrated_data.
↪DQ_image,'J',
                                           calibrated_data.bkg_image)
```

```
Source 1's traces will hit the vertical bar of doom, compensating by subtracting the␣
↪median of the edges of each row
Source 1's traces will hit the vertical bar of doom, compensating by subtracting the␣
↪median of the edges of each row
```

We can also look at what the background subtracted data look like:

[19]:
```
calibrated_data.source_list[1].plot_cutouts(figsize=(10,6),plot_bkg_sub=True,vmin=-200,
↪vmax=400)
```



nbsphinx-code-borderwhite

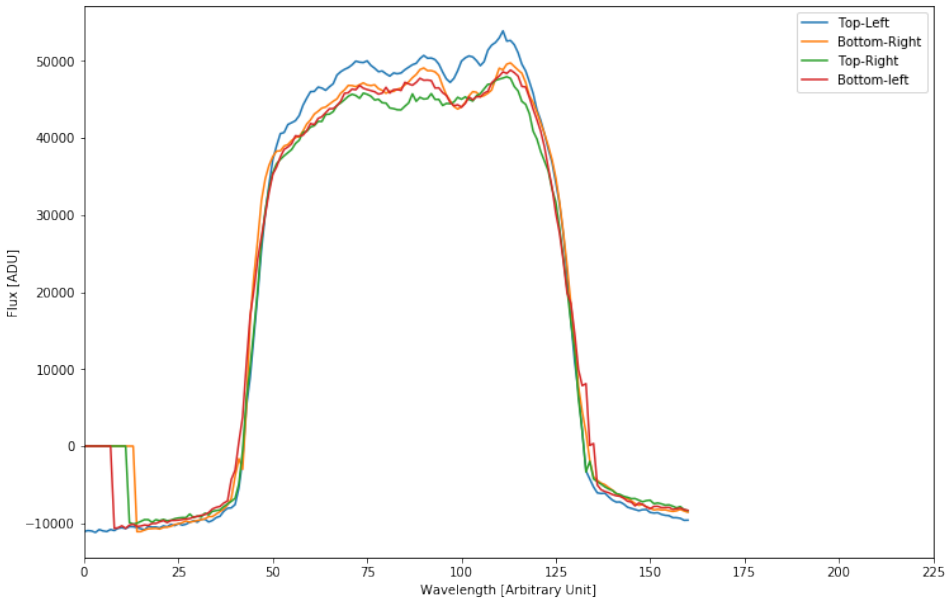
5.3.4 Step 3 - Extract the spectra from each thumbnail

In this step we will extract the specta from one of the source objects we found in Step 2.

[20]:
```
#Here you can enable "plot" to see where the traces were "found"
calibrated_data.source_list[0].extract_spectra(verbose=False)
```

```
/Users/maxwellmb/Dropbox (Personal)/Library/Python/wirc_drp/wirc_drp/utils/spec_utils.py:
↪553: RuntimeWarning: invalid value encountered in true_divide
  flux_opt_final = np.nan_to_num(sum1/sum3)
/Users/maxwellmb/Dropbox (Personal)/Library/Python/wirc_drp/wirc_drp/utils/spec_utils.py:
↪554: RuntimeWarning: invalid value encountered in true_divide
  variance_opt_final = np.nan_to_num(sumP/sum3)
/Users/maxwellmb/Dropbox (Personal)/Library/Python/wirc_drp/wirc_drp/utils/spec_utils.py:
↪457: RuntimeWarning: divide by zero encountered in true_divide
  P_0 = (data - background)/flux_0 #this is dividing each column (x) in data-background␣
↪by the sum in that column
/Users/maxwellmb/Dropbox (Personal)/Library/Python/wirc_drp/wirc_drp/utils/spec_utils.py:
↪457: RuntimeWarning: invalid value encountered in true_divide
  P_0 = (data - background)/flux_0 #this is dividing each column (x) in data-background␣
↪by the sum in that column
/Users/maxwellmb/Dropbox (Personal)/Library/Python/wirc_drp/wirc_drp/utils/spec_utils.py:
↪462: RuntimeWarning: invalid value encountered in less
  P_0[P_0 < 0] = 0
/Users/maxwellmb/Dropbox (Personal)/Library/Python/wirc_drp/wirc_drp/utils/spec_utils.py:
↪533: RuntimeWarning: invalid value encountered in less
  Mask = (data - background - flux_0*P_0)**2 < sig_clip**2*variance_opt
```

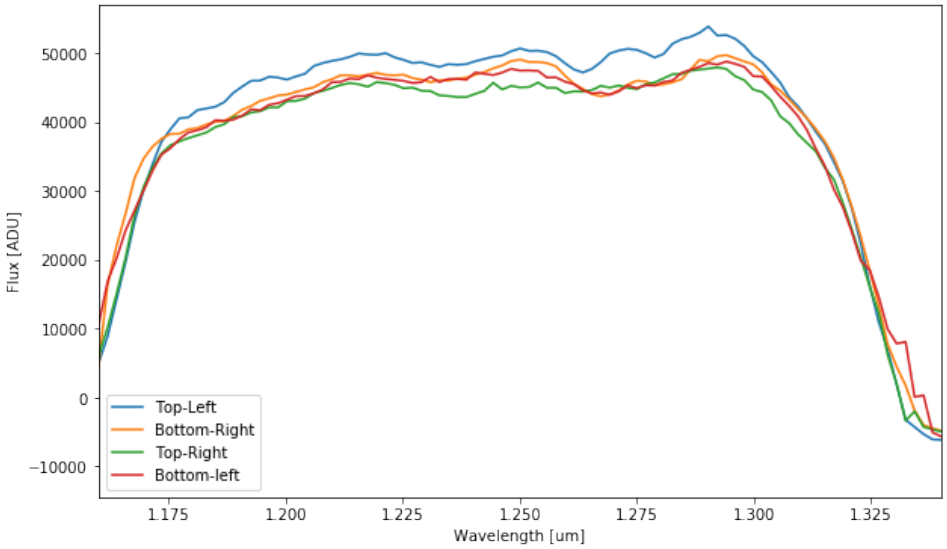
[21]:
```
#And we'll plot the data!
calibrated_data.source_list[0].plot_trace_spectra(figsize=(12,8))
```



nbsphinx-code-borderwhite

Now we'll do a rough wavelength calibration. Currently it uses the edge of the filters to shift and scale the spectra. This will eventually need to be refined as we're not super happy with it.

[22]:
```
## Now we'll do a rough wavelength calibration and plot the data again
calibrated_data.source_list[0].rough_lambda_calibration(method=2)
#Note method=1 is broken.

#And we'll plot the data again.
calibrated_data.source_list[0].plot_trace_spectra(figsize=(10,6),xlow=1.16,xhigh=1.34)
```



nbsphinx-code-borderwhite

5.3.5 Step 4 - Calculate the polarized spectra

We now combine the 4 spectra to calculate Stokes Q and U.

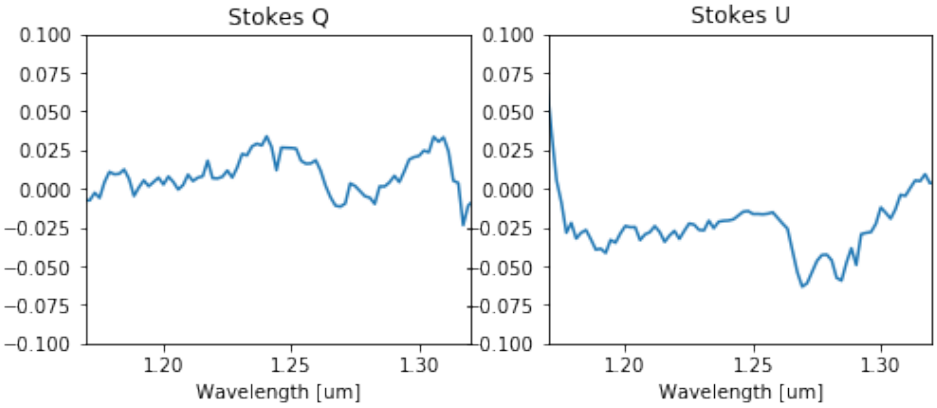
We'll note that at this point it is a "single-differenced" spectrum, and is subject to instrumental biases. These can be easily removed in data taken after Feb 2019 when we installed a modulator into WIRC+Pol. Tutorial 3 covers how to process data taken with the modulator (though don't skip Tutorial 2!).

These instrumental biases can also be calibrated, but to a worse precision, in data before Feb 2019 date using an unpolarized standard taken at a similar telescope pointing.

[23]:
```
calibrated_data.source_list[0].compute_polarization(cutmin=20, cutmax=150)
```

```
/Users/maxwellmb/Dropbox (Personal)/Library/Python/wirc_drp/wirc_drp/utils/spec_utils.py:
↪1751: RuntimeWarning: invalid value encountered in true_divide
  u = (Up-Um)/(Up+Um)
```

[24]:
```
calibrated_data.source_list[0].plot_Q_and_U(figsize=(8,3),xlow=1.17,xhigh=1.32,ylow=-0.1,
↪yhigh=0.1)
```



nbsphinx-code-borderwhite

5.3.6 Step 5 - Save the wirc object

We now save the new information and tables to a new file Note how it initiates the table columns even if the data hasn't been computed (for example, we haven't computed source_list[1] yet)

[26]:
```
calibrated_data.save_wirc_object(tutorial_dir+"calibrated.fits")
```

```
Saving a wirc_object to /Users/maxwellmb/Dropbox (Personal)/Library/Python/wirc_drp/wirc_
↪drp/../Tutorial/sample_data/Single_File_Tutorial/calibrated.fits
```

[ ]:

[ ]:

[ ]:

CHAPTER

SIX

TUTORIAL 2: DATASET REDUCTION TUTORIAL

Welcome to the WIRC+Pol dataset reduction pipeline (DRP) tutorial. The notebook will give you a basic run through of how to reduce a set of WIRC+Pol data. It assumes that you have already completed the **WIRC+Pol_Tutorial_1-Single_File_Spectral_Extraction** tutorial, which also goes through how to make calibration files.

6.1 This Tutorial

This tutorial will give an example of reducing a set of raw WIRC+Pol data images from start to finish, and will give you an idea of the implementation of the pipeline along the way. We'll note that many of the later data reduction steps (e.g. spectral extraction) are works in progress, and so the final data products should not yet be used for science. The purpose of this tutorial is more to demonstrate the features of the pipeline rather than to teach you how to fully calibrate your data.

In this tutorial we reduce two datasets: an unpolarized standard, and a brown dwarf. Each dataset has several files to read in and process.

6.2 The Recipe

There are several steps to reducing polarimetric data, some in common with typical photometry data reduction and some not. Here is a brief list of steps. Steps (1) through (4) are demonstrated for a single file in the *Single_File* tutorial. **If you have data from after March 2019, after step (5), you'll want to check out the "WIRC+Pol_Tutorial_3-Reducing_a_Modulation_Sequence" tutorial which goes over how to deal with data taken in sequence with the modulator.**

Here are the steps:

(1) Correct for detector dark current and pixel-to-pixel variation by subtracting dark and dividing flat.

(2) Source identification. The DRP has a capability to detect a source in the image. Alternatively, the pixel coordinates of the source can be manually entered. The DRP then grabs the 4 spectral traces associated with the source from the calibrated science image.

(3) Subtract background from the data. There are multiple ways to do this, but in this tutorial we will simply use the shifted science image for background subtraction.

(4) Spectral extraction. In this step we extract 1D spectra from the 4 spectral traces, which correspond to 4 different linear polarization angles. After finishing steps (2) to (4) for both the unpolarized standard star and the source, you can proceed to the next step.

(5) Stack spectra. We can now combine the whole observing sequence to increase SNR before we compute the polarization.

(6) Polarimetric calibration. Raw polarization measurements from WIRC+Pol are incorrect by several percent, but we can remove this offset by an observation of a known unpolarized star. We apply the correction in this step to measure polarization intrinsic to the source.

6.2.1 Step 0 - Read in the calibration files

The very first step will be to create your own master darks and master flats, for that see the *Single_File* tutorial. For the sake of this tutorial we well assume that you have done this already. Already-made master darks and flats have been provided in the tutorial directory.

[1]:
```
#Import the important things
%matplotlib inline
%load_ext autoreload
%autoreload 2
import wirc_drp.wirc_object as wo
from wirc_drp.utils import calibration, spec_utils as su, image_utils as iu
from astropy.io import fits
import matplotlib.pyplot as plt
import numpy as np
import astropy.io.ascii as asci
import os, copy
from scipy.ndimage import shift
import glob

import warnings; warnings.simplefilter('ignore')
```

[2]:
```
tutorial_dir = os.getcwd()
```

For this Tutorial, we will use pre-processed calibration files

[3]:
```
os.chdir('./sample_data/')
```

[4]:
```
flat_name = 'wirc0627_master_flat.fits'
bp_name = 'wirc0627_bp_map.fits'
dark_name1s_10 = 'wirc0648_master_dark.fits' #A dark with a 1s exposure time, 10-coadd
dark_name60s_1 = 'wirc0711_master_dark.fits' #A dark with a 60s exposure time, 1 coadd
```

6.2.2 Step 1 - Read in and calibrate your data

Assuming that you now have a master dark and master flat file, we can now read in a wirc_data file and perform the calibration.

[5]:
```
#First we'll set up all the directories and filenames:
unpol_filelist = 'HD11639_sci.list' #for the unpolarized standard
#source_filelist = 'SIMP0136_sci.list' #for the source, the brown dwarf here

#Read in the file names
unpol_fnames  = asci.read(unpol_filelist , format = 'fast_no_header')['col1']
#source_fnames = asci.read(source_filelist, format = 'fast_no_header')['col1']
```

First we deal with the unpolarized standard star. We will demonstrate the process for one file, then loop through and calibrate everything.

[6]:
```
#Choose dark with the correct exposure time here
dark_fn = dark_name1s_10
#Then select flat and bad pixels map
flat_fn = flat_name
bp_fn   = bp_name

#Now we'll create the wirc_data object, passing in the filenames for the master dark,␣
↪flat and bad pixel maps
raw_data = wo.wirc_data(raw_filename=unpol_fnames[0], flat_fn = flat_fn, dark_fn = dark_
↪fn, bp_fn = bp_fn,
                        ref_lib=sorted(glob.glob('/home/mmnguyen/Research/WIRC+POL/Data/
↪2018421/Sky/*.fits')))
```

```
Creating a new wirc_data object from file wirc0140.fits
Found a J-band filter in the header of file wirc0140.fits
```

Let's run the calibration step. It will subtract the dark, divide by the flat, and interpolate over the bad pixels.

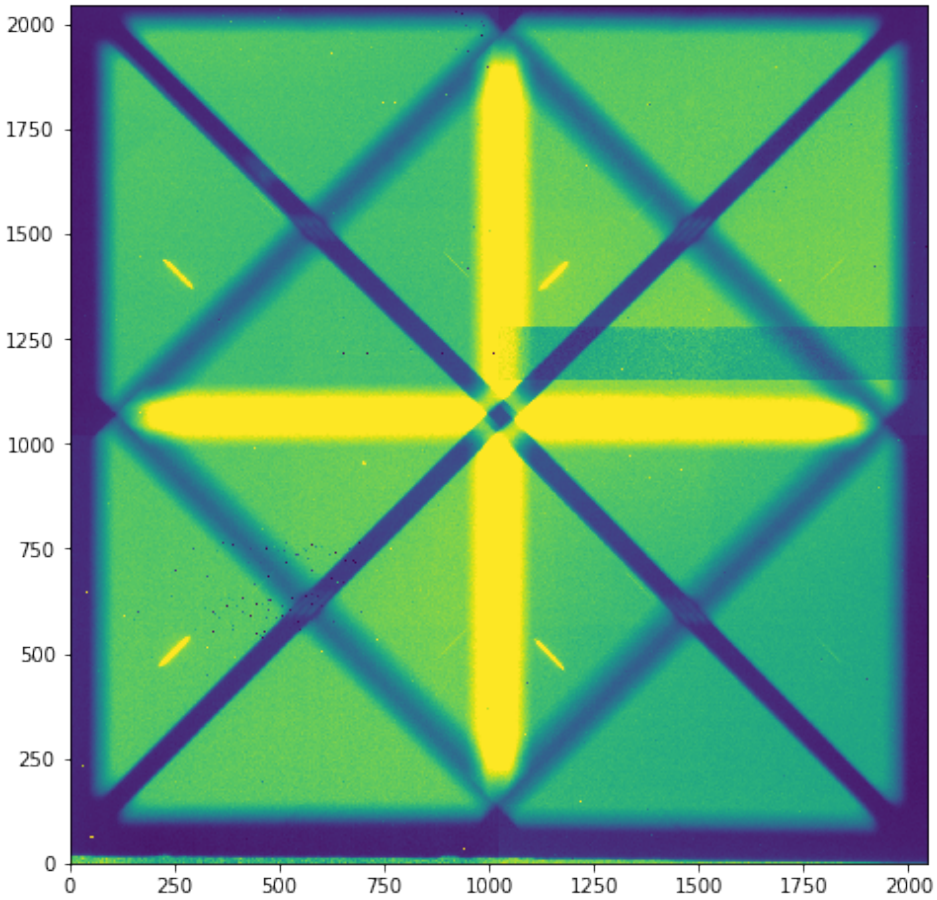
[7]:
```
raw_data.calibrate()
```

```
/Users/maxwellmb/Dropbox (Personal)/Library/Python/wirc_drp/wirc_drp/wirc_object.py:220:␣
↪RuntimeWarning: divide by zero encountered in true_divide
  self.full_image = self.full_image/master_flat
/Users/maxwellmb/Dropbox (Personal)/Library/Python/wirc_drp/wirc_drp/wirc_object.py:220:␣
↪RuntimeWarning: invalid value encountered in true_divide
  self.full_image = self.full_image/master_flat
/Users/maxwellmb/Dropbox (Personal)/Library/Python/wirc_drp/wirc_drp/utils/calibration.
↪py:734: RuntimeWarning: invalid value encountered in less_equal
  bad_pixel_map = np.logical_or(bad_pixel_map, redux_science <= 0)
```

What does it look like?

[8]:
```
plt.figure(figsize=(8,8))
plt.imshow(raw_data.full_image, vmin=0, vmax=5000, origin = 'lower')
```

[8]:
```
<matplotlib.image.AxesImage at 0x1c1f8569b0>
```



nbsphinx-code-borderwhite

There are a few remaining artifacts in the image, but overall, not bad!

Now we can loop through all the files and calibrate them!

[9]:
```
######Set the calibration output directory. Set this up for your own group!!!#####
calib_dir = tutorial_dir+'/sample_data/'

unpol_calibrated_list = np.array([]) #This is a list of the calibrated files.

#Read in the file names
unpol_fnames  = asci.read(unpol_filelist , format = 'fast_no_header')['col1']
#source_fnames = asci.read(source_filelist, format = 'fast_no_header')['col1']

for i, fn in enumerate(unpol_fnames):
    #open the file
    raw_data = wo.wirc_data(raw_filename=unpol_fnames[i], flat_fn = flat_fn, dark_fn =␣
↪dark_fn, bp_fn = bp_fn)
    #run the calibration
    raw_data.calibrate()
    #file name of the calibrated file
    outname = fn.split('.')[0]+'_calib.fits'
    #add to the list and save it
    unpol_calibrated_list = np.append(unpol_calibrated_list, calib_dir+outname)
    raw_data.save_wirc_object(calib_dir+outname)
```

(continues on next page)

(continued from previous page)

```
###You may want to save the list, for late use, here's an example how:
# np.save(calib_dir+'unpol_calibrated_list.npy', unpol_calibrated_list)
```

```
Creating a new wirc_data object from file wirc0140.fits
Found a J-band filter in the header of file wirc0140.fits
Saving a wirc_object to /Users/maxwellmb/Dropbox (Personal)/Library/Python/wirc_drp/
↪Tutorial/sample_data/wirc0140_calib.fits
Creating a new wirc_data object from file wirc0141.fits
Found a J-band filter in the header of file wirc0141.fits
Saving a wirc_object to /Users/maxwellmb/Dropbox (Personal)/Library/Python/wirc_drp/
↪Tutorial/sample_data/wirc0141_calib.fits
Creating a new wirc_data object from file wirc0142.fits
Found a J-band filter in the header of file wirc0142.fits
Saving a wirc_object to /Users/maxwellmb/Dropbox (Personal)/Library/Python/wirc_drp/
↪Tutorial/sample_data/wirc0142_calib.fits
```

6.2.3 Now it's your turn to set up a loop to calibrate the brown dwarf data!

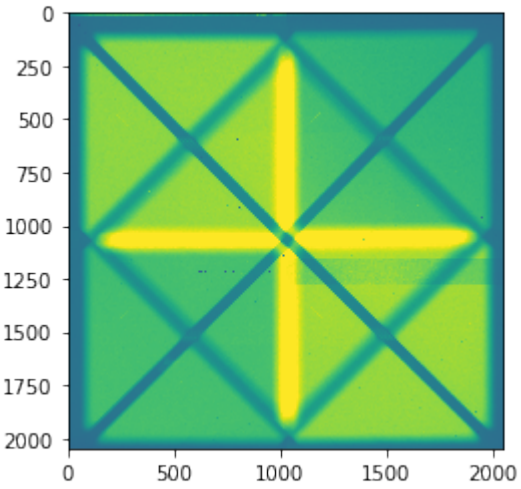
[10]:
```
################ Brown Dwarf calibraton goes here ################

BD_calib_dir = tutorial_dir+'/sample_data/'

BD_calibrated_list = np.array([]) #This is a list of the calibrated files.

#First we'll set up all the directories and filenames:
BD_filelist = 'SIMP0136_sci.list' #for the source, the brown dwarf here

#Read in the file names
BD_fnames  = asci.read(BD_filelist , format = 'fast_no_header')['col1']
#source_fnames = asci.read(source_filelist, format = 'fast_no_header')['col1']

for i, fn in enumerate(BD_fnames):
    #open the file
    raw_data = wo.wirc_data(raw_filename=BD_fnames[i], flat_fn = flat_fn, dark_fn = dark_
↪name60s_1, bp_fn = bp_fn)
    #run the calibration
    raw_data.calibrate(mask_bad_pixels=False)
    #file name of the calibrated file
    outname = fn.split('.')[0]+'_calib.fits'
    #add to the list and save it
    BD_calibrated_list = np.append(BD_calibrated_list, BD_calib_dir+outname)
    raw_data.save_wirc_object(BD_calib_dir+outname)

    plt.figure()
    plt.imshow(raw_data.full_image,vmax=10000)

###You may want to save the list, for late use, here's an example how:
# np.save(BD_calib_dir+'BD_calibrated_list.npy', BD_calibrated_list)
```

```
Creating a new wirc_data object from file wirc0158.fits
```

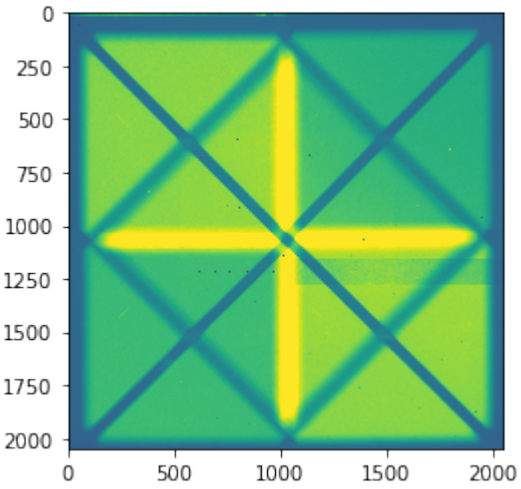
(continues on next page)

(continued from previous page)

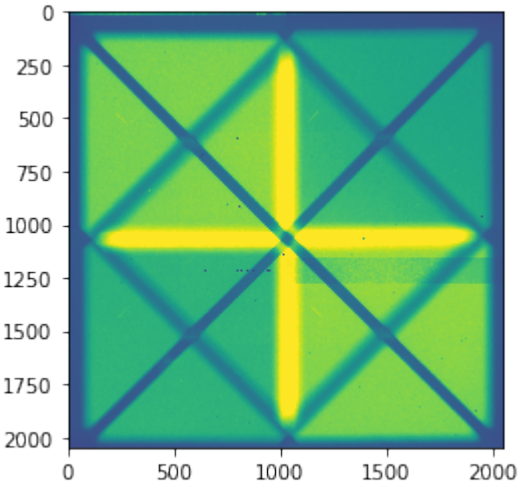
```
Found a J-band filter in the header of file wirc0158.fits
Saving a wirc_object to /Users/maxwellmb/Dropbox (Personal)/Library/Python/wirc_drp/
↪Tutorial/sample_data/wirc0158_calib.fits
Creating a new wirc_data object from file wirc0159.fits
Found a J-band filter in the header of file wirc0159.fits
Saving a wirc_object to /Users/maxwellmb/Dropbox (Personal)/Library/Python/wirc_drp/
↪Tutorial/sample_data/wirc0159_calib.fits
Creating a new wirc_data object from file wirc0160.fits
Found a J-band filter in the header of file wirc0160.fits
Saving a wirc_object to /Users/maxwellmb/Dropbox (Personal)/Library/Python/wirc_drp/
↪Tutorial/sample_data/wirc0160_calib.fits
```



nbsphinx-code-borderwhite



nbsphinx-code-borderwhite



nbsphinx-code-borderwhite

6.2.4 Step 2: Source identification

In this step we will create s wircpol_source object and attach it to each wirc_pol data object. Each source object will contain cutout thumbnails of each spectral trace, and eventually spectra and polarized spectra. We will then append the wircpol_source object to the wirc object's "source_list".

Again, we will show this step for one file first, then we will show steps 3-4 for this file. In the end we will loop through and do this for every file.

[11]:
```
#First we'll create a new data object, mostly just to demonstrate how to read in an␣
↪existing wirc_data object.
calibrated_data = wo.wirc_data(wirc_object_filename=BD_calibrated_list[1])
```

```
Loading a wirc_data object from file /Users/maxwellmb/Dropbox (Personal)/Library/Python/
↪wirc_drp/Tutorial/sample_data/wirc0159_calib.fits
```

Generate a background image by scaling the "calibrated" image from the _Single_File tutorial

[12]:
```
calibrated_data.generate_bkg(method="scaled_bkg",bkg_fns=[tutorial_dir+"/calibrated.fits
↪"],same_HWP=False)
```

Put in the coordinates of the source here, then add source to the wirc_data object

[13]:
```
source_coords = [700, 955] #X, Y position of the zeroth order of the source (undispersed␣
↪star in the center)

# This function add the source to our wirc_data object created in the last cell.
# update_w_chi2_shift flag centers the source
calibrated_data.add_source( source_coords[0], source_coords[1],
                slit_pos = "slitless", update_w_chi2_shift = True, verbose = True)

# We can access the source from the source list attribute
wp_source = calibrated_data.source_list[0]
```

```
Loading Template from /Users/maxwellmb/Dropbox (Personal)/Library/Python/wirc_drp/wirc_
↪drp/masks/single_trace_template2.fits
```

(continues on next page)

(continued from previous page)

```
Shfits are x,y =  [3.314453125, 1.751953125, 0.052734375, 0.052734375]
Applying shifts
```

```
/Users/maxwellmb/Dropbox (Personal)/Library/Python/image_registration/build/lib.macosx-
↪10.7-x86_64-3.7/image_registration/fft_tools/zoom.py:101: FutureWarning: Using a non-
↪tuple sequence for multidimensional indexing is deprecated; use `arr[tuple(seq)]`␣
↪instead of `arr[seq]`. In the future this will be interpreted as an array index,␣
↪`arr[np.array(seq)]`, which will result either in an error or a different result.
  outarr[ii] = outarr_d[dims]
```

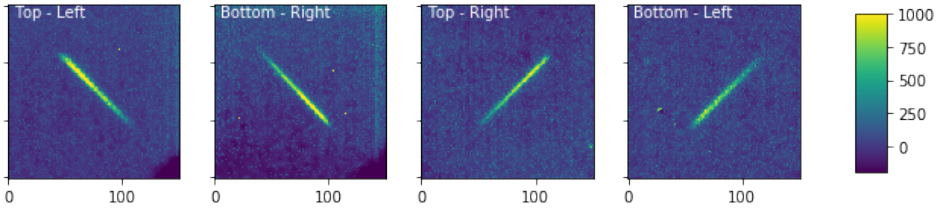
We'll now get the trace cutouts for this source

[14]:
```
wp_source.get_cutouts(calibrated_data.full_image, calibrated_data.DQ_image, calibrated_
↪data.filter_name,
                      calibrated_data.bkg_image, replace_bad_pixels = True, sub_bar =␣
↪True, cutout_size = 75)
```

```
Source 1's traces will hit the vertical bar of doom, compensating by subtracting the␣
↪median of the edges of each row
Source 1's traces will hit the vertical bar of doom, compensating by subtracting the␣
↪median of the edges of each row
```

Plot the cutouts

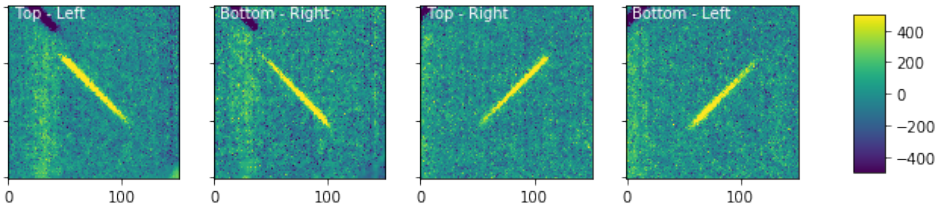
[15]:
```
wp_source.plot_cutouts(figsize=(10,6),vmin=-200,vmax=1000)
```



nbsphinx-code-borderwhite

Plot the background-subtracted cutouts

[16]:
```
wp_source.plot_cutouts(figsize=(10,6),plot_bkg_sub=True,vmin=-500,vmax=500)
```



nbsphinx-code-borderwhite

This background subtraction has clearly introduced some features into the image that aren't ideal, but in a real observation set you will have taken more appropriate background images. There are also several different background subgraction options that you can experiment with within the **wircpol_data.generate_bkg()** functions that may improve your results.

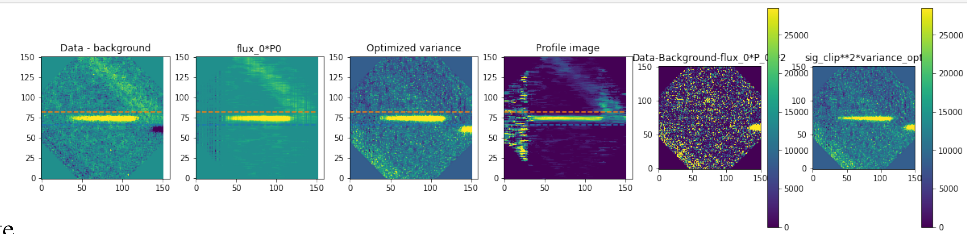
6.2.5 Step 3-4: Background subtraction and spectral extraction

In this step we will subtract background and extract the specta from one of the source objects we found in Step 2. Options for the spectral extraction function are pretty extensive, but here are all you need to care about:

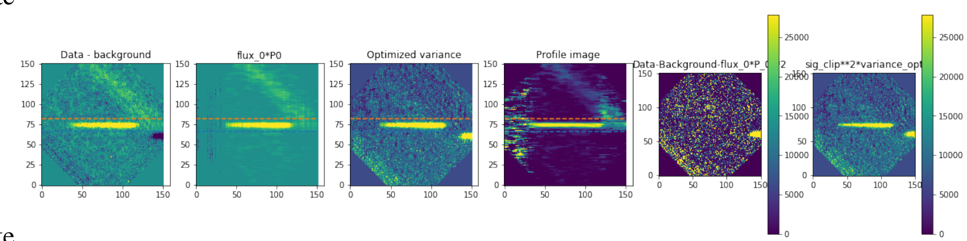
method is how we exactly extract the spectra. We are using optimal extraction here and you can read about it in Horne 1986. **spatial_sigma** is how far in the wings of the spectral trace we want to extract.

Here we'll plot some of the internal workings for each trace. Each trace goes through 2 iterations of the optimal extraction algorithm. The orange and blue lines show the regions defined by the **spatial_sigma** parameter.

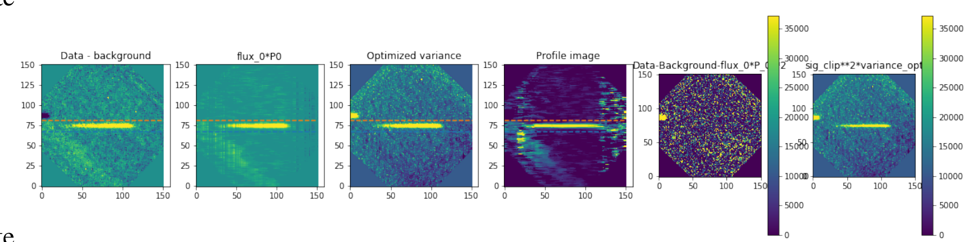
[17]:
```
calibrated_data.source_list[0].extract_spectra(method="optimal_extraction",
                                               spatial_sigma=4,
                                               bad_pix_masking = 1,
                                               plot_optimal_extraction = True,
                                               plot_findTrace = False,
                                               verbose=False)
```



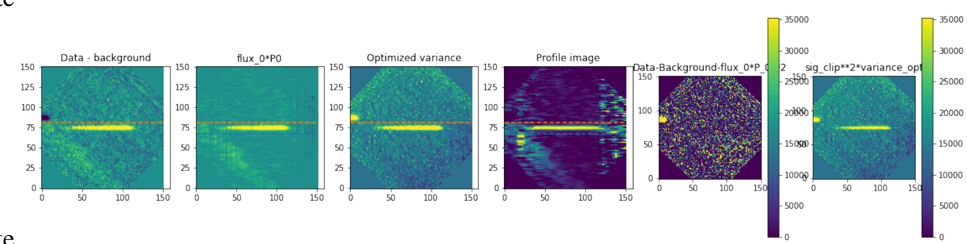
nbsphinx-code-borderwhite



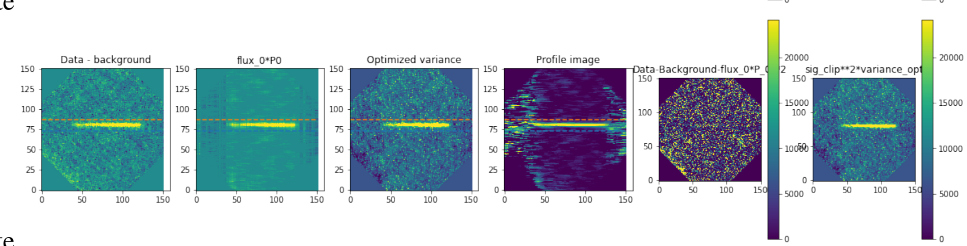
nbsphinx-code-borderwhite



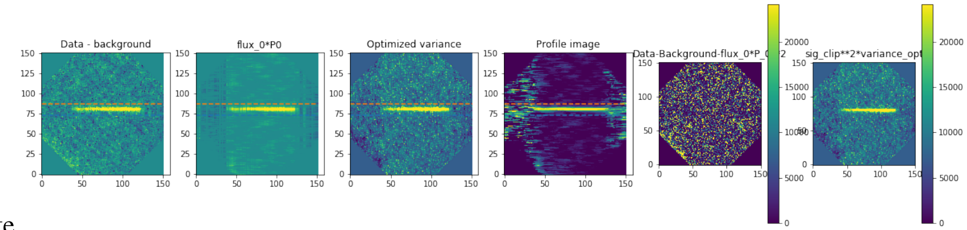
nbsphinx-code-borderwhite



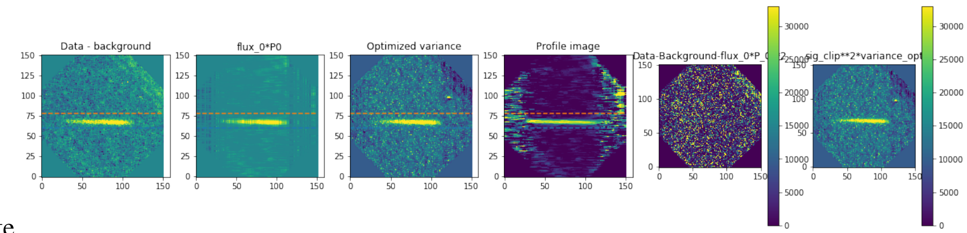
nbsphinx-code-borderwhite



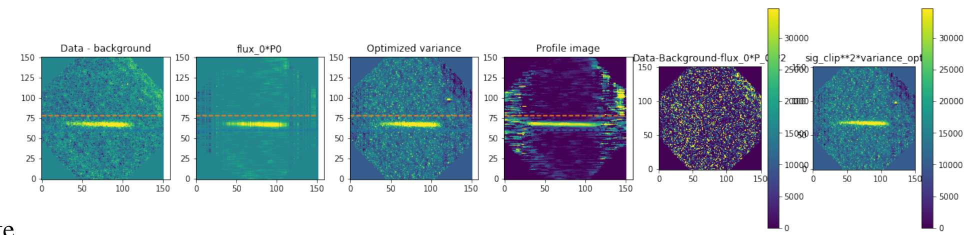
nbsphinx-code-borderwhite



nbsphinx-code-borderwhite



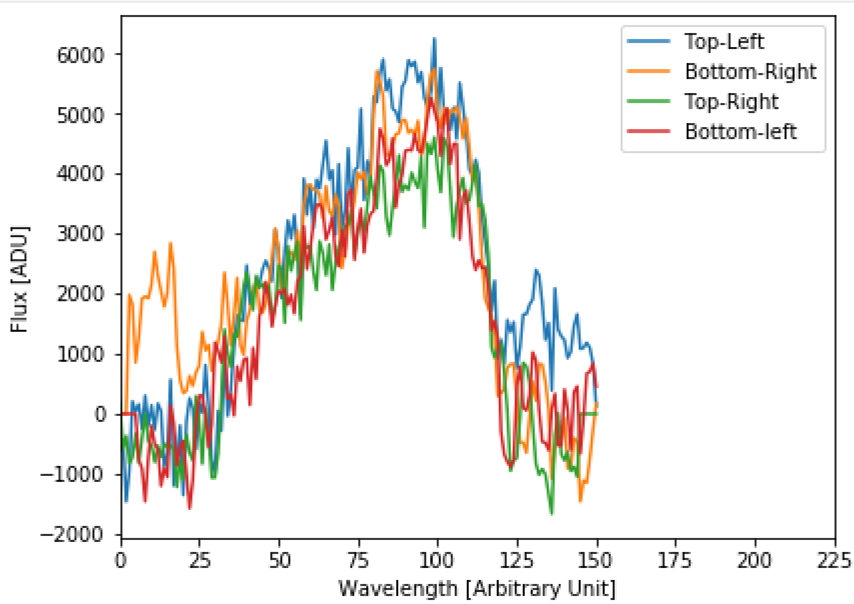
nbsphinx-code-borderwhite



nbsphinx-code-borderwhite

Now let's plot the spectra:

[18]:
```
calibrated_data.source_list[0].plot_trace_spectra()
```



nbsphinx-code-borderwhite

It is clear that these spectra are messed up, and it's related to the background levels. This is a work in progress (and one of the reasons we suggest that you don't use your data for science without talking to the instrument team).

6.2.6 Now with Steps 2-4 demonstrated for one file, let's run it all on all files: First with the unpolarized standard star

[19]:
```
source_list = []
all_spec_cube = []
counter = 0
for i,fn in enumerate(unpol_calibrated_list):
    try:
        print("Extracting from file {} of {}".format(counter+1, len(unpol_calibrated_
↪list)))
        data = wo.wirc_data(wirc_object_filename=fn, verbose=False)
        im = np.array(data.full_image).copy()

        data.generate_bkg()

        #Clear the source list
        data.source_list = []
        data.n_sources = 0
        data.add_source( source_coords[0], source_coords[1], slit_pos = "slitless",
                        update_w_chi2_shift = True, verbose = False)

        #Let's save ourselves from million plots
        do_plot = False
        #Get cutouts of the 4 spectra and plot them for every 10 frames
        data.source_list[0].get_cutouts(data.full_image, data.DQ_image, data.filter_name,
↪ data.bkg_image,
                                        replace_bad_pixels = True,
                                        sub_bar = True, cutout_size = 75)
#         if do_plot == True:
#             data.source_list[0].plot_cutouts(origin ='lower')
        #Now extract the spectra
        data.source_list[0].extract_spectra(method="optimal_extraction",
                                    bad_pix_masking =1,
                                    plot_optimal_extraction = False,
                                     plot_findTrace = do_plot,
                                    verbose=False, spatial_sigma=10)
        #Save the spectra in this array
        all_spec_cube.append(data.source_list[0].trace_spectra)
        #Save the file
        data.save_wirc_object(fn)
        source_list += [data.source_list[0]]
    except Exception as e:
        print("Some sort of Error, skipping file {}".format(fn))
        print("Error {}".format(e))
    counter += 1
# trace_info = iu.locate_traces(im,sky, plot=False, verbose=True)

source_list = copy.deepcopy(source_list)
up_all_spec_cube = np.array(all_spec_cube)
```

```
Extracting from file 1 of 3
Subtracting background using shift and subtract method.
Applying shifts
```

(continues on next page)

(continued from previous page)

```
Source 1's traces will hit the vertical bar of doom, compensating by subtracting the␣
↪median of the edges of each row
Source 1's traces will hit the vertical bar of doom, compensating by subtracting the␣
↪median of the edges of each row
Saving a wirc_object to /Users/maxwellmb/Dropbox (Personal)/Library/Python/wirc_drp/
↪Tutorial/sample_data/wirc0140_calib.fits
```

```
WARNING: VerifyWarning: Card is too long, comment will be truncated. [astropy.io.fits.
↪card]
```

```
Extracting from file 2 of 3
Subtracting background using shift and subtract method.
Applying shifts
Source 1's traces will hit the vertical bar of doom, compensating by subtracting the␣
↪median of the edges of each row
Source 1's traces will hit the vertical bar of doom, compensating by subtracting the␣
↪median of the edges of each row
Saving a wirc_object to /Users/maxwellmb/Dropbox (Personal)/Library/Python/wirc_drp/
↪Tutorial/sample_data/wirc0141_calib.fits
Extracting from file 3 of 3
Subtracting background using shift and subtract method.
Applying shifts
Source 1's traces will hit the vertical bar of doom, compensating by subtracting the␣
↪median of the edges of each row
Source 1's traces will hit the vertical bar of doom, compensating by subtracting the␣
↪median of the edges of each row
Saving a wirc_object to /Users/maxwellmb/Dropbox (Personal)/Library/Python/wirc_drp/
↪Tutorial/sample_data/wirc0142_calib.fits
```

6.2.7 Then the brown dwarf

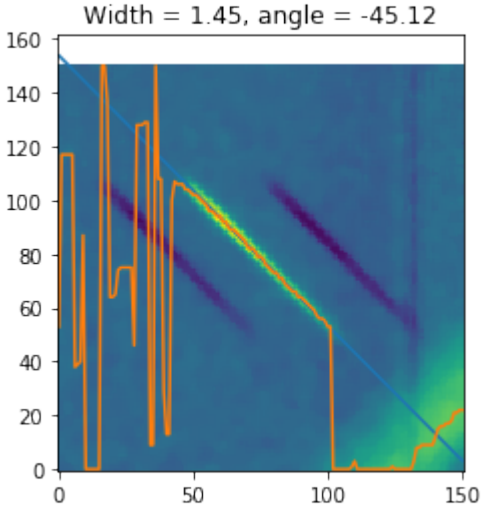
[20]:
```
source_list = []
BD_all_spec_cube = []
counter = 0
for i,fn in enumerate(BD_calibrated_list):
    try:
        print("Extracting from file {} of {}".format(counter+1, len(BD_calibrated_list)))
        data = wo.wirc_data(wirc_object_filename=fn, verbose=False)
        data.generate_bkg()
        im = np.array(data.full_image).copy()
        #Clear the source list
        data.source_list = []
        data.n_sources = 0
        data.add_source( source_coords[0], source_coords[1], slit_pos = "slitless",
                        update_w_chi2_shift = True, verbose = False)
        #Let's save ourselves from million plots
        if i%10 == 0:
            do_plot = True
        else:
            do_plot = False
```

(continues on next page)

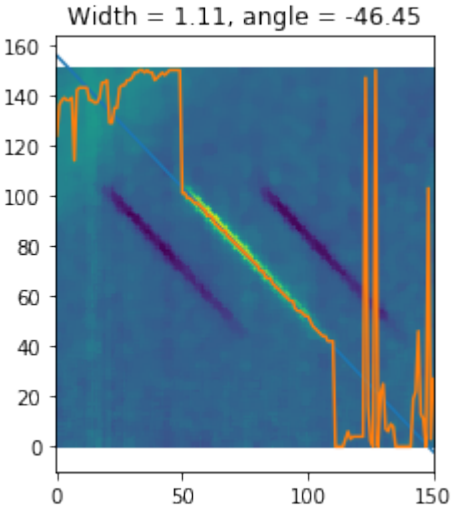
(continued from previous page)

```
        #Get cutouts of the 4 spectra and plot them for every 10 frames
        data.source_list[0].get_cutouts(data.full_image, data.DQ_image, data.filter_name,
↪ data.bkg_image,
                                        replace_bad_pixels = True,
                                        sub_bar = True, cutout_size = 75)
#         if do_plot == True:
#             data.source_list[0].plot_cutouts(origin ='lower')
        #Now extract the spectra
        data.source_list[0].extract_spectra(method="optimal_extraction",
                                    bad_pix_masking =1,
                                    plot_optimal_extraction = False,
                                     plot_findTrace = do_plot,
                                    verbose=False, spatial_sigma=10)
        #Save the spectra in this array
        BD_all_spec_cube.append(data.source_list[0].trace_spectra)
        #Save the file
        data.save_wirc_object(fn)
        source_list += [data.source_list[0]]
    except Exception as e:
        print("Some sort of Error, skipping file {}".format(fn))
        print("Error {}".format(e))
    counter += 1
# trace_info = iu.locate_traces(im,sky, plot=False, verbose=True)

source_list = copy.deepcopy(source_list)
BD_up_all_spec_cube = np.array(BD_all_spec_cube)
```

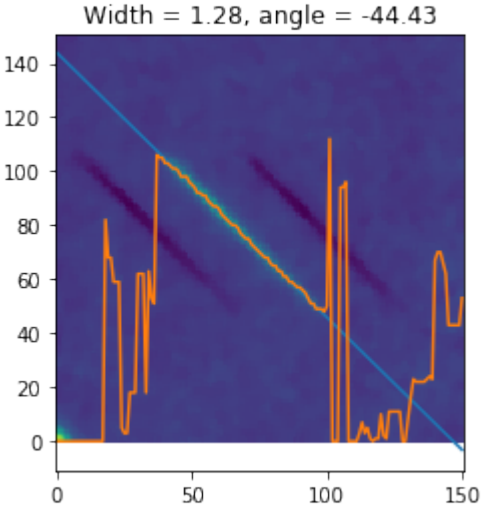
```
Extracting from file 1 of 3
Subtracting background using shift and subtract method.
Applying shifts
Source 1's traces will hit the vertical bar of doom, compensating by subtracting the␣
↪median of the edges of each row
Source 1's traces will hit the vertical bar of doom, compensating by subtracting the␣
↪median of the edges of each row
```



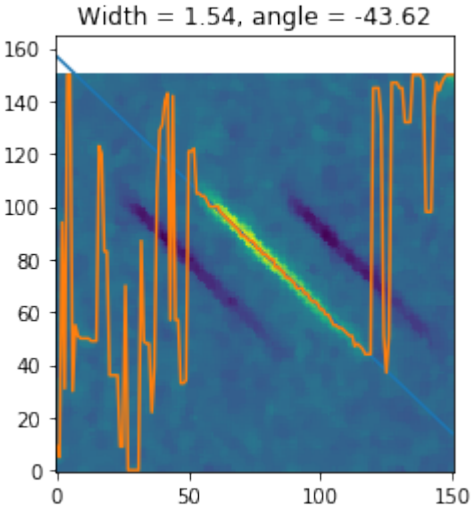
nbsphinx-code-borderwhite



nbsphinx-code-borderwhite



nbsphinx-code-borderwhite



nbsphinx-code-borderwhite

```
Saving a wirc_object to /Users/maxwellmb/Dropbox (Personal)/Library/Python/wirc_drp/
↪Tutorial/sample_data/wirc0158_calib.fits
```

(continues on next page)

(continued from previous page)

```
Extracting from file 2 of 3
Subtracting background using shift and subtract method.
Applying shifts
Source 1's traces will hit the vertical bar of doom, compensating by subtracting the␣
↪median of the edges of each row
Source 1's traces will hit the vertical bar of doom, compensating by subtracting the␣
↪median of the edges of each row
Saving a wirc_object to /Users/maxwellmb/Dropbox (Personal)/Library/Python/wirc_drp/
↪Tutorial/sample_data/wirc0159_calib.fits
Extracting from file 3 of 3
Subtracting background using shift and subtract method.
Applying shifts
Source 1's traces will hit the vertical bar of doom, compensating by subtracting the␣
↪median of the edges of each row
Source 1's traces will hit the vertical bar of doom, compensating by subtracting the␣
↪median of the edges of each row
Saving a wirc_object to /Users/maxwellmb/Dropbox (Personal)/Library/Python/wirc_drp/
↪Tutorial/sample_data/wirc0160_calib.fits
```

6.2.8 Step 5 - Stack the spectra

Here we simply align spectra from all the exposures, scale and median combine them.

[21]:
```
#Align the spectra.
up_aligned_cube = su.align_spectral_cube(up_all_spec_cube)

#Scale the spectra to compensate for atmospheric variation - you may not want to do this␣
↪all the time
up_scaled_cube = su.scale_and_combine_spectra(up_aligned_cube, return_scaled_cube = True)
```

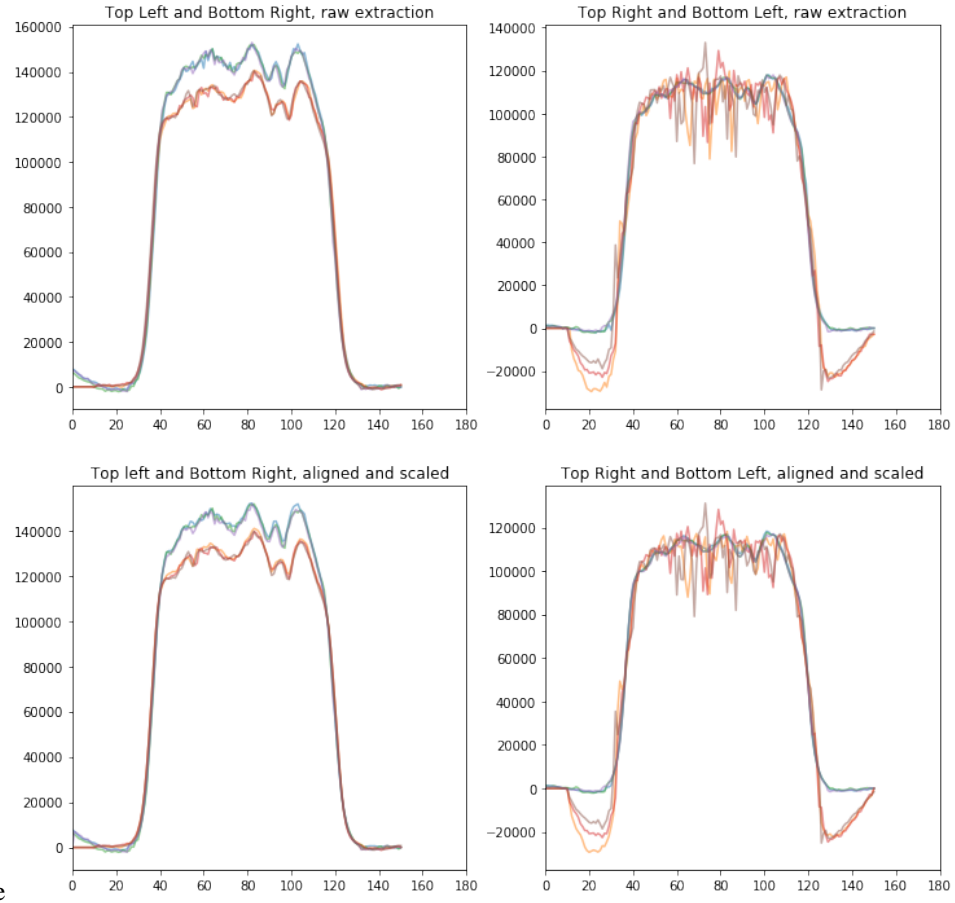
[22]:
```
#Plot them. The top plots are raw extraction, bottom are aligned and scaled.
#The left plots are upper left, lower right (U pair), and right plots are Q pair.
### What do the spectra look like
fig, ax = plt.subplots(2,2,figsize=(12,12))
low_ind =  0
high_ind = 180

cm1 = plt.get_cmap('bwr')
cm2 = plt.get_cmap('copper')
alpha = 0.5

for i in range(up_all_spec_cube.shape[0]):
    ax[0,0].plot(up_all_spec_cube[i,0,1,:], alpha = alpha) #top left
    ax[0,0].plot(up_all_spec_cube[i,1,1,:], alpha = alpha) #bottom right
    ax[0,1].plot(up_all_spec_cube[i,2,1,:], alpha = alpha) #top right
    ax[0,1].plot(up_all_spec_cube[i,3,1,:], alpha = alpha) #bottom left

    #aligned and scaled cube
    ax[1,0].plot(up_scaled_cube[i,0,1,:], alpha = alpha) #top left
    ax[1,0].plot(up_scaled_cube[i,1,1,:], alpha = alpha) #bottom right
```

(continues on next page)

(continued from previous page)

```
    ax[1,1].plot(up_scaled_cube[i,2,1,:], alpha = alpha) #top right
    ax[1,1].plot(up_scaled_cube[i,3,1,:], alpha = alpha) #bottom left

ax[0,0].set_xlim([low_ind, high_ind])
ax[0,1].set_xlim([low_ind, high_ind])
ax[1,0].set_xlim([low_ind, high_ind])
ax[1,1].set_xlim([low_ind, high_ind])
# ax[0,0].set_ylim([0, 180000])
# ax[0,1].set_ylim([0, 150000])
# ax[1,0].set_ylim([0, 180000])
# ax[1,1].set_ylim([0, 150000])


ax[0,0].set_title('Top Left and Bottom Right, raw extraction')
ax[0,1].set_title('Top Right and Bottom Left, raw extraction')
ax[1,0].set_title('Top left and Bottom Right, aligned and scaled')
ax[1,1].set_title('Top Right and Bottom Left, aligned and scaled')
```

[22]:
```
Text(0.5, 1.0, 'Top Right and Bottom Left, aligned and scaled')
```



nbsphinx-code-borderwhite

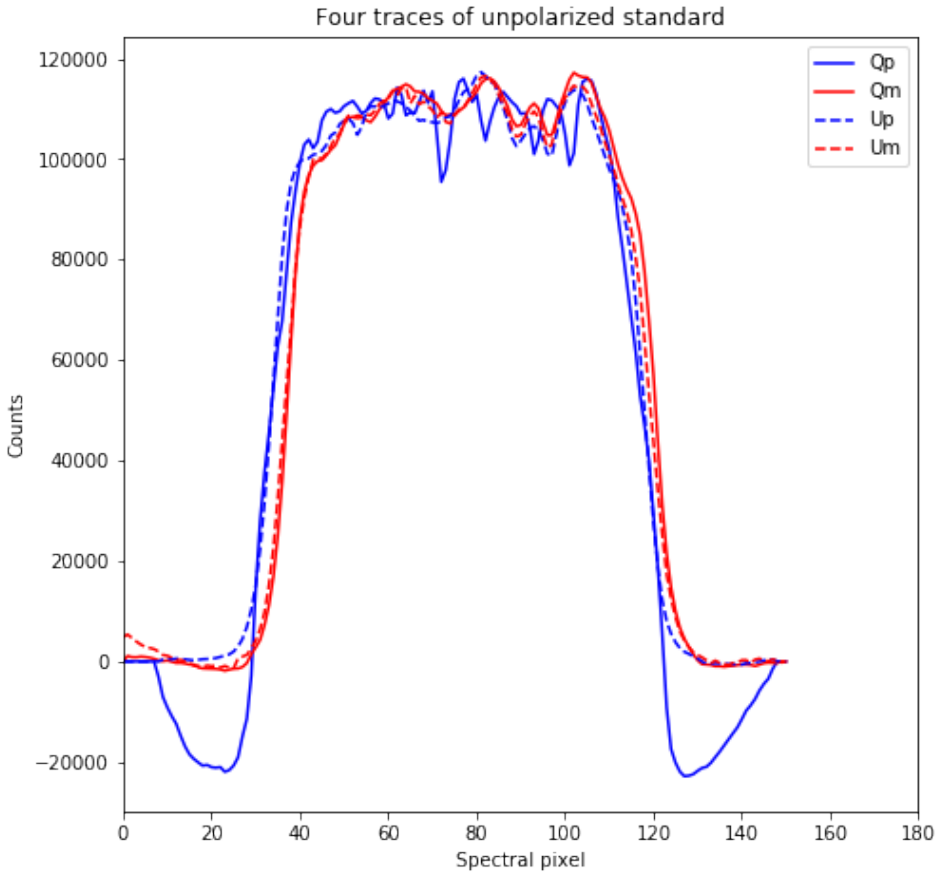
We can now median combine and compute standard deviation of the measurement.

The most manual step here is to make sure the 4 spectra are aligned in wavelength, this is caused by the tilt of the J-band filter. The offsets given below are good, but let's plot them to check.

[23]:
```
sshift_um = 0
sshift_up = -2.2
sshift_qm = +0.7
sshift_qp = -2.5

unpol_um_med = shift(np.nanmedian(up_scaled_cube[:,0,1,:], axis=0),sshift_um, order = 1)
unpol_up_med = shift(np.nanmedian(up_scaled_cube[:,1,1,:], axis=0),sshift_up, order = 1)
unpol_qm_med = shift(np.nanmedian(up_scaled_cube[:,2,1,:], axis=0),sshift_qm, order = 1)
unpol_qp_med = shift(np.nanmedian(up_scaled_cube[:,3,1,:], axis=0),sshift_qp, order = 1)

unpol_um_std = shift(np.nanstd(up_scaled_cube[:,0,1,:], axis=0)/np.sqrt(up_scaled_cube.
↪shape[0]),sshift_um, order = 1)
unpol_up_std = shift(np.nanstd(up_scaled_cube[:,1,1,:], axis=0)/np.sqrt(up_scaled_cube.
↪shape[0]),sshift_up, order = 1)
unpol_qm_std = shift(np.nanstd(up_scaled_cube[:,2,1,:], axis=0)/np.sqrt(up_scaled_cube.
↪shape[0]),sshift_qm, order = 1)
unpol_qp_std = shift(np.nanstd(up_scaled_cube[:,3,1,:], axis=0)/np.sqrt(up_scaled_cube.
↪shape[0]),sshift_qp, order = 1)
```

[24]:
```
plt.figure(figsize = (7.5,7.5))
vector = [unpol_qp_med,unpol_qm_med,unpol_up_med,unpol_um_med]
labels = ['Qp','Qm','Up','Um']
factors = np.sum(unpol_qp_med[50:110])/np.array([np.sum(x[50:110]) for x in vector])
factors[1] = factors[1]*1.015
factors[2] = factors[2]*0.995
factors[3] = factors[3]*1.005
fmts =   ['-b','-r','--b','--r']
for i in range(len(vector)):
    plt.plot(vector[i]*factors[i],fmts[i], label = labels[i])
# plt.xlim([low_ind, high_ind])
plt.xlim([low_ind, high_ind])
plt.xlim([0,180])
#plt.ylim([100000,140000])
plt.xlabel('Spectral pixel')
plt.ylabel('Counts')
plt.legend()
plt.title('Four traces of unpolarized standard')
```

[24]:
```
Text(0.5, 1.0, 'Four traces of unpolarized standard')
```



nbsphinx-code-borderwhite

Each trace corresponds to Stokes parameters as following: Qp – Bottom left; Qm – Top right; Up – Bottom right; Um – Top left.

These are such that normalized Stokes parameters are q = (Qp - Qm)/(Qp + Qm) and u = (Up - Um)/(Up + Um)

6.2.9 Now do the same for the brown dwarf

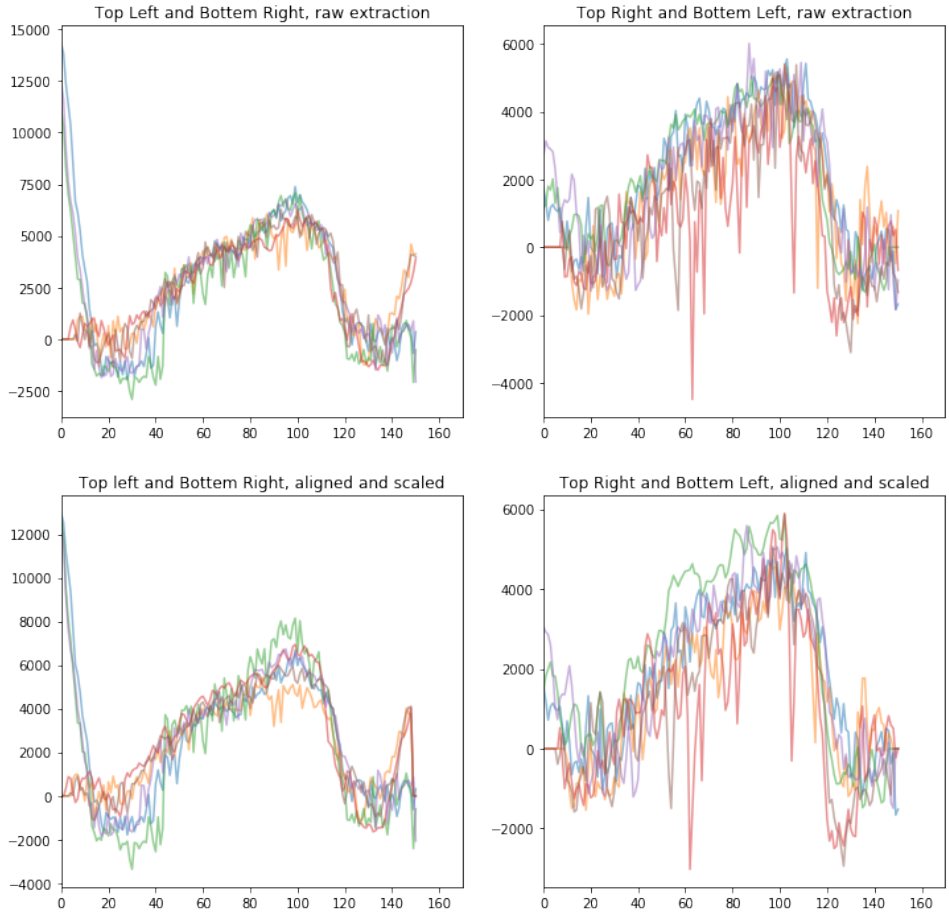
[25]:
```
BD_up_aligned_cube = su.align_spectral_cube(BD_up_all_spec_cube)
BD_up_scaled_cube = su.scale_and_combine_spectra(BD_up_aligned_cube, return_scaled_cube␣
↪= True)

fig1, ax1 = plt.subplots(2,2,figsize=(12,12))
low_ind =  0
high_ind = 170

cm1 = plt.get_cmap('bwr')
cm2 = plt.get_cmap('copper')
alpha = 0.5

for i in range(BD_up_all_spec_cube.shape[0]):
    ax1[0,0].plot(BD_up_all_spec_cube[i,0,1,:], alpha = alpha) #top left
    ax1[0,0].plot(BD_up_all_spec_cube[i,1,1,:], alpha = alpha) #bottom right
    ax1[0,1].plot(BD_up_all_spec_cube[i,2,1,:], alpha = alpha) #top right
```

(continues on next page)

(continued from previous page)

```
    ax1[0,1].plot(BD_up_all_spec_cube[i,3,1,:], alpha = alpha) #bottom left

    #aligned and scaled cube
    ax1[1,0].plot(BD_up_scaled_cube[i,0,1,:], alpha = alpha) #top left
    ax1[1,0].plot(BD_up_scaled_cube[i,1,1,:], alpha = alpha) #bottom right
    ax1[1,1].plot(BD_up_scaled_cube[i,2,1,:], alpha = alpha) #top right
    ax1[1,1].plot(BD_up_scaled_cube[i,3,1,:], alpha = alpha) #bottom left

ax1[0,0].set_xlim([low_ind, high_ind])
ax1[0,1].set_xlim([low_ind, high_ind])
ax1[1,0].set_xlim([low_ind, high_ind])
ax1[1,1].set_xlim([low_ind, high_ind])
#ax1[0,0].set_ylim([0, 180000])
#ax1[0,1].set_ylim([0, 150000])
#ax1[1,0].set_ylim([0, 180000])
#ax1[1,1].set_ylim([0, 150000])


ax1[0,0].set_title('Top Left and Bottem Right, raw extraction')
ax1[0,1].set_title('Top Right and Bottem Left, raw extraction')
ax1[1,0].set_title('Top left and Bottem Right, aligned and scaled')
ax1[1,1].set_title('Top Right and Bottem Left, aligned and scaled')
```

[25]:
```
Text(0.5, 1.0, 'Top Right and Bottem Left, aligned and scaled')
```



nbsphinx-code-borderwhite

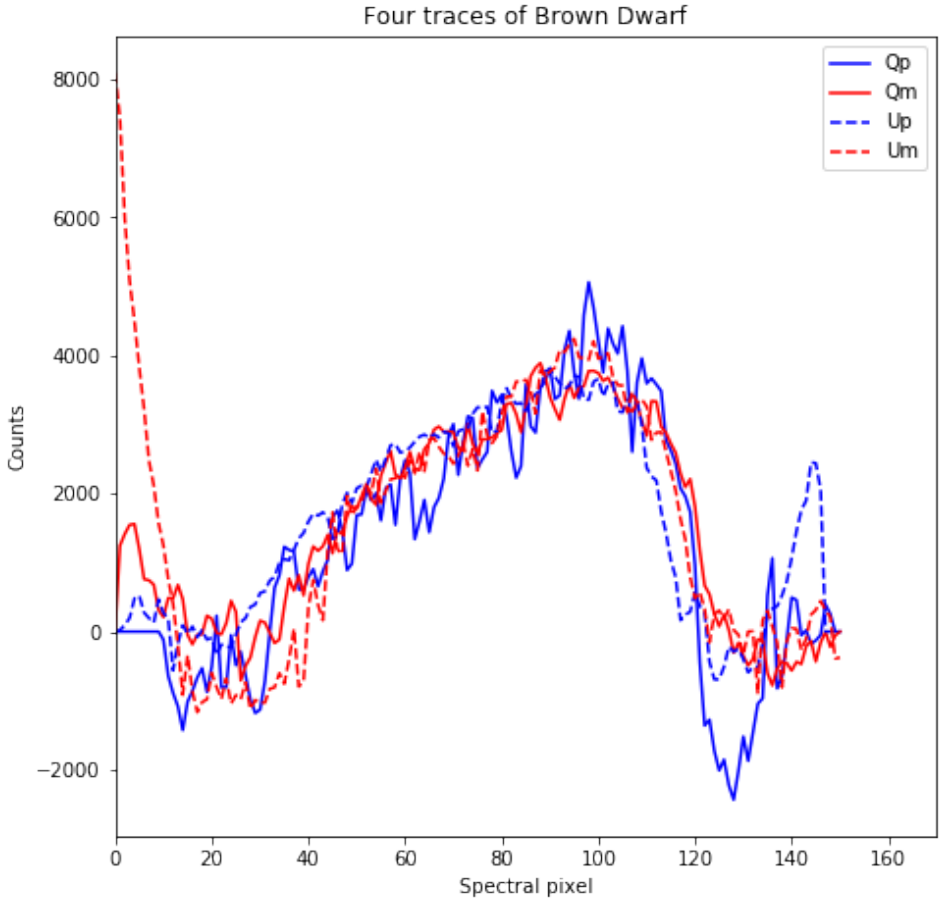
[26]:
```
#now shifts for the BD spectra
sshift_um = 0
sshift_up = -2.2
sshift_qm = +0.7
sshift_qp = +1

BD_um_med = shift(np.nanmedian(BD_up_scaled_cube[:,0,1,:], axis=0),sshift_um, order = 1)
BD_up_med = shift(np.nanmedian(BD_up_scaled_cube[:,1,1,:], axis=0),sshift_up, order = 1)
BD_qm_med = shift(np.nanmedian(BD_up_scaled_cube[:,2,1,:], axis=0),sshift_qm, order = 1)
BD_qp_med = shift(np.nanmedian(BD_up_scaled_cube[:,3,1,:], axis=0),sshift_qp, order = 1)

BD_um_std = shift(np.nanstd(BD_up_scaled_cube[:,0,1,:], axis=0)/np.sqrt(BD_up_scaled_
↪cube.shape[0]),sshift_um, order = 1)
BD_up_std = shift(np.nanstd(BD_up_scaled_cube[:,1,1,:], axis=0)/np.sqrt(BD_up_scaled_
↪cube.shape[0]),sshift_up, order = 1)
BD_qm_std = shift(np.nanstd(BD_up_scaled_cube[:,2,1,:], axis=0)/np.sqrt(BD_up_scaled_
↪cube.shape[0]),sshift_qm, order = 1)
BD_qp_std = shift(np.nanstd(BD_up_scaled_cube[:,3,1,:], axis=0)/np.sqrt(BD_up_scaled_
↪cube.shape[0]),sshift_qp, order = 1)

plt.figure(figsize = (7.5,7.5))
vector = [BD_qp_med,BD_qm_med,BD_up_med,BD_um_med]
labels = ['Qp','Qm','Up','Um']
```

(continues on next page)

(continued from previous page)

```
factors = np.sum(BD_qp_med[40:120])/np.array([np.sum(x[40:120]) for x in vector])
factors[1] = factors[1]*1.015
factors[2] = factors[2]*0.995
factors[3] = factors[3]*1.005
fmts =   ['-b','-r','--b','--r']
for i in range(len(vector)):
    plt.plot(vector[i]*factors[i],fmts[i], label = labels[i])
# plt.xlim([low_ind, high_ind])
plt.xlim([low_ind, high_ind])
#plt.xlim([160,180])
#plt.ylim([100000,140000])
plt.xlabel('Spectral pixel')
plt.ylabel('Counts')
plt.legend()
plt.title('Four traces of Brown Dwarf')
```

[26]:
```
Text(0.5, 1.0, 'Four traces of Brown Dwarf')
```



nbsphinx-code-borderwhite

Let's now save out spectra to a numpy file.

[27]:
```
np.save("BD_Spectra",BD_up_scaled_cube)
```

6.2.10 Step 6: Polarization calibration and calculation

Note, that if you're using data from after March 2019 that uses the new awesome half-wave plate, then instead of reading this section, you should move on to the other tutorial: WIRC+Pol_Tutorial_3-Reducing_a_Modulation_Sequence

Suggestions: The most crucial part here is to get wavelength to line up. This is a bit tricky since WIRC+Pol is slitless and there's no easy way to get wavelength solution directly from the data.

We suggest that you align the brown dwarf spectra to those of the standard star **trace by trace** relying on the filter cutoffs. Note that different traces have difference filter cutoff.

From the last part, you would have the median spectra: unpol_qp_med, unpol_qm_med and pol_qp_med, pol_qm_med from the standard and the brown dwarf respectively.

[28]:
```
#First compute q, u from the unpolarized star. This is the instrumental polarization
import math
import numpy as np

q_inst = (unpol_qp_med - unpol_qm_med)/(unpol_qp_med + unpol_qm_med)
u_inst = (unpol_up_med - unpol_um_med)/(unpol_up_med + unpol_um_med)

# Uncertainties, fill this in!

q_inst_err = 2/(unpol_qp_med+unpol_qm_med)**2*np.sqrt((unpol_qp_med*unpol_qm_std)**2 +␣
↪(unpol_qm_med*unpol_qp_std)**2 )
u_inst_err = 2/(unpol_up_med+unpol_um_med)**2*np.sqrt((unpol_up_med*unpol_um_std)**2 +␣
↪(unpol_um_med*unpol_up_std)**2 )
```

```
/Users/maxwellmb/Anaconda3/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:5:
↪ RuntimeWarning: invalid value encountered in true_divide
  """
/Users/maxwellmb/Anaconda3/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:6:
↪ RuntimeWarning: invalid value encountered in true_divide

/Users/maxwellmb/Anaconda3/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:
↪10: RuntimeWarning: divide by zero encountered in true_divide
  # Remove the CWD from sys.path while we load stuff.
/Users/maxwellmb/Anaconda3/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:
↪10: RuntimeWarning: invalid value encountered in multiply
  # Remove the CWD from sys.path while we load stuff.
/Users/maxwellmb/Anaconda3/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:
↪11: RuntimeWarning: divide by zero encountered in true_divide
  # This is added back by InteractiveShellApp.init_path()
/Users/maxwellmb/Anaconda3/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:
↪11: RuntimeWarning: invalid value encountered in multiply
  # This is added back by InteractiveShellApp.init_path()
```

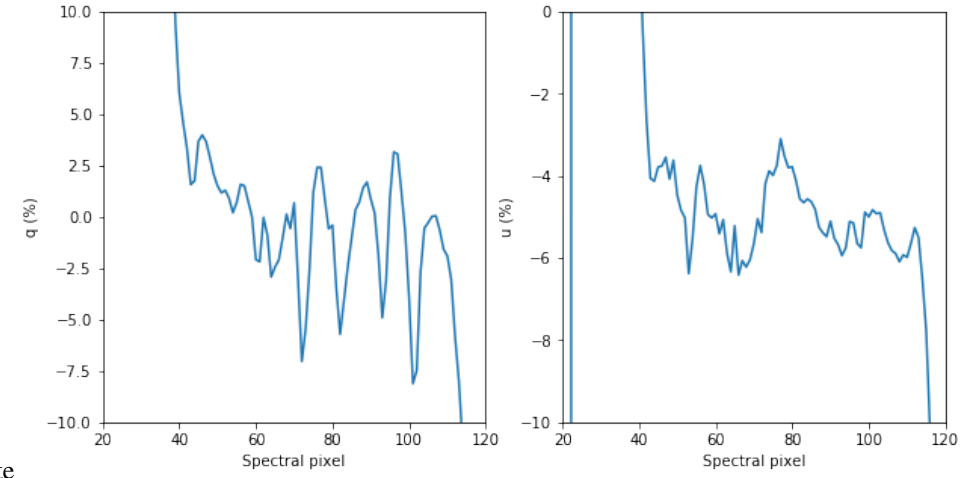
[29]:
```
#show this

fig, ax = plt.subplots(1,2, figsize = (10,5))
ax[0].plot(q_inst*100)
ax[1].plot(u_inst*100)

ax[0].set_xlim([20,120])
```

(continues on next page)

(continued from previous page)

```
ax[1].set_xlim([20,120])
ax[0].set_ylim([-10,10])
ax[1].set_ylim([-10,0])

ax[0].set_xlabel('Spectral pixel')
ax[1].set_xlabel('Spectral pixel')

ax[0].set_ylabel('q (%)')
ax[1].set_ylabel('u (%)')
```

[29]:
```
Text(0, 0.5, 'u (%)')
```



nbsphinx-code-borderwhite

[30]:
```
#### Do the same for the Brown Dwarf ####
```

[31]:
```
q_BD_raw = (BD_qp_med - BD_qm_med)/(BD_qp_med + BD_qm_med)
u_BD_raw = (BD_up_med - BD_um_med)/(BD_up_med + BD_um_med)

# Uncertainties, fill this in!

q_BD_raw_err = 2/(BD_qp_med+BD_qm_med)**2*np.sqrt((BD_qp_med*BD_qm_std)**2 + (BD_qm_
↪med*BD_qp_std)**2 )
u_BD_raw_err = 2/(BD_up_med+BD_um_med)**2*np.sqrt((BD_up_med*BD_um_std)**2 + (BD_um_
↪med*BD_up_std)**2 )

#Now subtract the instrumental polarization
q_BD_corrected =  q_BD_raw - q_inst
u_BD_corrected =  u_BD_raw - u_inst

q_BD_corrected_err = np.sqrt(q_BD_raw_err**2 + q_inst_err**2)
u_BD_corrected_err = np.sqrt(u_BD_raw_err**2 + u_inst_err**2)
```

```
/Users/maxwellmb/Anaconda3/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:1:
↪ RuntimeWarning: invalid value encountered in true_divide
  """Entry point for launching an IPython kernel.
/Users/maxwellmb/Anaconda3/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:6:
↪ RuntimeWarning: divide by zero encountered in true_divide
```

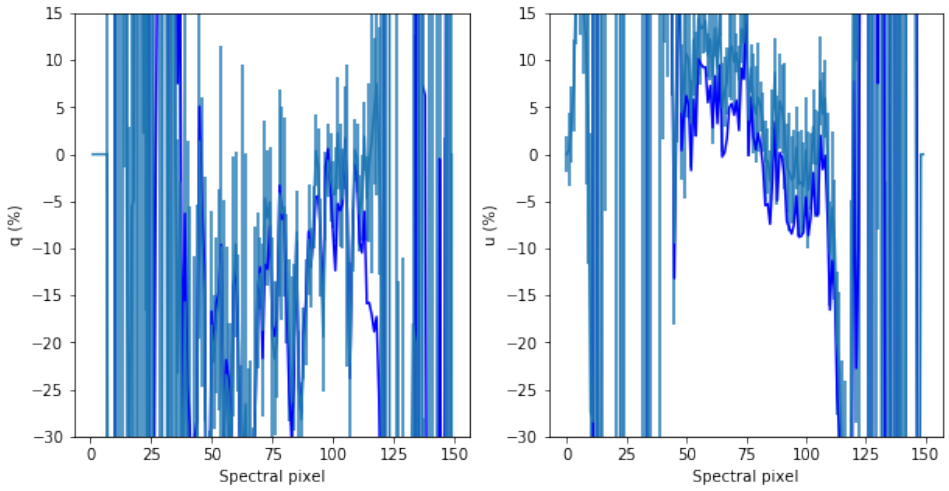
(continues on next page)

(continued from previous page)

```
/Users/maxwellmb/Anaconda3/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:6:
↪ RuntimeWarning: invalid value encountered in multiply
```

[32]:
```
### Plot q_BD_raw, q_BD_corrected here

fig, ax = plt.subplots(1,2, figsize = (10,5))

x = range(len(q_BD_raw))
ax[0].plot(x,q_BD_raw*100,'-b')
#ax[0].plot(q_BD_corrected*100,'-r')

ax[1].plot(x,u_BD_raw*100,'b-')
#ax[1].plot(u_BD_corrected*100,'-r')

#ax[0].set_xlim([100,200])
#ax[1].set_xlim([100,200])
ax[0].set_ylim([-30,15])
ax[1].set_ylim([-30,15])

ax[0].set_xlabel('Spectral pixel')
ax[1].set_xlabel('Spectral pixel')

ax[0].set_ylabel('q (%)')
ax[1].set_ylabel('u (%)')

#q here
ax[0].errorbar(x,q_BD_corrected*100,yerr=q_BD_corrected_err*100)
#u here
ax[1].errorbar(x,u_BD_corrected*100,yerr=u_BD_corrected_err*100)
```

[32]:
```
<ErrorbarContainer object of 3 artists>
```



nbsphinx-code-borderwhite

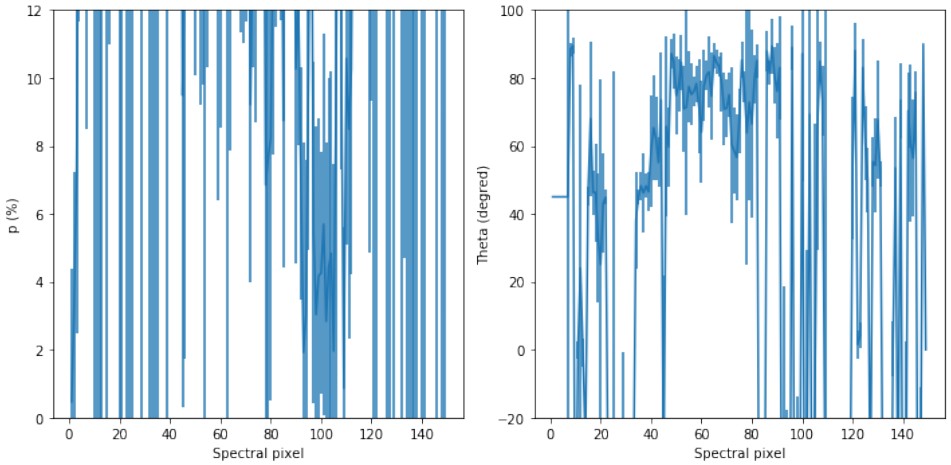
[33]:
```
#### Finally, compute degree and angle of polarization

p = np.sqrt(q_BD_corrected**2 + u_BD_corrected**2)
```

(continues on next page)

(continued from previous page)

```
theta = 0.5*np.arctan2(u_BD_corrected, q_BD_corrected)

#uncertainties

p_err = (1/p)*np.sqrt((q_BD_corrected*q_BD_corrected_err)**2 + (u_BD_corrected*u_BD_
↪corrected_err)**2)
# theta_err = (0.5/(1+(u_BD_corrected/q_BD_corrected)**2))*np.sqrt((u_BD_corrected_err/q_
↪BD_corrected)**2 + (u_BD_corrected*q_BD_corrected_err/(q_BD_corrected**2))**2)
theta_err = (0.5/p**2)*np.sqrt((q_BD_corrected*u_BD_corrected_err)**2 + (u_BD_
↪corrected*q_BD_corrected_err)**2)

#plot these!

fig, ax = plt.subplots(1,2, figsize = (10,5))

x = range(len(q_BD_raw))
# ax[0].plot(x,p*100,'-b')
# ax[1].plot(x,theta,'b-')
#ax[1].plot(u_BD_corrected*100,'-r')

#ax[0].set_xlim([100,200])
#ax[1].set_xlim([100,200])
ax[0].set_ylim([-0,12])
ax[1].set_ylim([-20,100])

ax[0].set_xlabel('Spectral pixel')
ax[1].set_xlabel('Spectral pixel')

ax[0].set_ylabel('p (%)')
ax[1].set_ylabel('Theta (degred)')

#q here
ax[0].errorbar(x,p*100,yerr=p_err*100)
#u here
ax[1].errorbar(x,np.degrees(theta),yerr=np.degrees(theta_err))
plt.tight_layout()
```

```
/Users/maxwellmb/Anaconda3/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:8:
↪ RuntimeWarning: divide by zero encountered in true_divide

/Users/maxwellmb/Anaconda3/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:8:
↪ RuntimeWarning: invalid value encountered in multiply

/Users/maxwellmb/Anaconda3/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:
↪10: RuntimeWarning: divide by zero encountered in true_divide
  # Remove the CWD from sys.path while we load stuff.
/Users/maxwellmb/Anaconda3/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:
↪10: RuntimeWarning: invalid value encountered in multiply
  # Remove the CWD from sys.path while we load stuff.
```



nbsphinx-code-borderwhite

6.3 Acknowledgement

The data used in this Tutorial are taken by the Ay122a class in Fall 2018 with Prof. Dimitri Mawet. Class members are Aida, Anise, Anusha, Luiza, Nitika, Ryan, Yuhan, Yuping, Zhihui, and TA Jackie Pezzato. They contributed to making this Tutorial as well.

[ ]:

[ ]:

CHAPTER

SEVEN

TUTORIAL 3: MODULATION SEQUENCE TUTORIAL

This tutorial is to show you how to reduce WIRC+Pol data obtained after March 2019 using the new awesome modulator to obtain q,u,p and theta.

In this tutorial we assume that you have already been through the two other tutorials, named WIRC+Pol_Tutorial_1-Single_File_Spectral_Extraction and WIRC+Pol_Tutorial_2-Reducing_a_Dataset. These tutorials will touch on installation, give you a good instroduction to the formats of WIRC+Pol data and demonstrate how to extract the spectra for a larger dataset.

Here we assume that you already have run the pipeline to extract spectra on a dataset, but have yet to combine them via single differencing (in time) or double-differencing.

[4]:
```
import numpy as np
import glob
import wirc_drp
import wirc_drp.wirc_object as wo
from wirc_drp.utils import source_utils
from wirc_drp.utils import calibration as wc
import matplotlib.pyplot as plt
%load_ext autoreload
%autoreload 2
%matplotlib inline

import warnings; warnings.simplefilter('ignore')
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

7.1 Step 1: Read in the files

7.1.1 Read in the files - Tutorial Dataset

For the purpose of this tutorial, we are providing you with spectra and hwp angles in .np files. That we read in here. Below we show a commented out example of how you would get these same inputs from a WIRC+Pol dataset.

We assume here that the group of spectra have already been aligned (see the Dataset tutorial).

[6]:
```
wircpol_dir = wirc_drp.__path__[0]+"/../"
tutorial_dir = wircpol_dir + "Tutorial/"

all_spec_cube = np.load(tutorial_dir+"sample_data/Elia2-14_HWP_Spectra.npy") #The␣
```

(continues on next page)

(continued from previous page)

```
↪extracted spectra
hwp_ang = np.load(tutorial_dir+"sample_data/Elia2-14_HWP_Angles.npy") #The HWP angles
wvs = all_spec_cube[0,0,0,:] #Grab the wavelengths from the first cube
nfiles = all_spec_cube.shape[0]
```

7.1.2 Read in the files - Normal Operations

The following three commented out boxes give you an example of how to read in the data for this tutorial from normal wircpol_object fits files. You would do something like this when reducing your own data.

[7]:
```
### Read in the files.
#fnames = np.sort(glob.glob("/path_to_your_files_here/*.fits"))
#nfiles = len(fnames)
#print("Found {} files".format(nfiles))
```

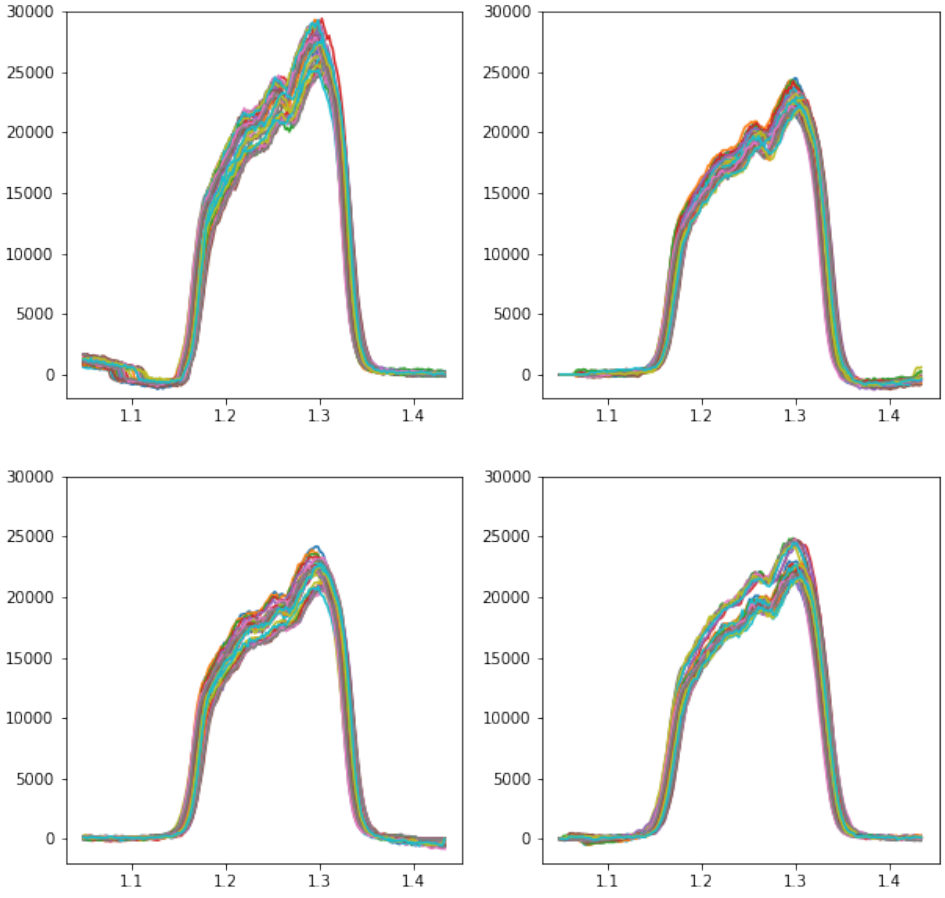
[8]:
```
### Read in the first one to get the spectra size
#test= wo.wirc_data(wirc_object_filename=fnames[0],verbose=False)

### Create a list to read the spectra into
# all_spec_cube = []

### And the HWP angles
#hwp_ang = np.zeros([nfiles])
#time = np.zeros([nfiles])
```

[9]:
```
# all_spec_cube = []
# for im in np.arange(nfiles):
#     wirc_object = wo.wirc_data(wirc_object_filename=fnames[im],verbose=False)
#     wirc_object.source_list[0].get_broadband_polarization(mode="aperture_photometry")
#     hwp_ang[im] = wirc_object.header['HWP_ANG']

#     #Shift the spectra to the side because of the filter tilt.
#     #The value of -3 used here might not be ideal, but we'll see.
#     spectra[im,1,:] = np.roll(spectra[im,1,:],-3)
#     spectra[im,3,:] = np.roll(spectra[im,3,:],-3)

#     wirc_object.source_list[0].trace_spectra[1,1,:] = np.roll(wirc_object.source_
↪list[0].trace_spectra[1,1,:],-3)
#     wirc_object.source_list[0].trace_spectra[3,1,:] = np.roll(wirc_object.source_
↪list[0].trace_spectra[3,1,:],-3)

#     all_spec_cube.append(wirc_object.source_list[0].trace_spectra)

# all_spec_cube = np.array(all_spec_cube)
```

7.2 Step 2: Plot the spectra and make sure they're aligned and ready to go

[10]:
```
fig,axes = plt.subplots(2,2,figsize=(10,10))

for im in np.arange(nfiles):
    axes[0,0].plot(wvs,all_spec_cube[im,0,1,:])
    axes[0,1].plot(wvs,all_spec_cube[im,1,1,:])
    axes[1,0].plot(wvs,all_spec_cube[im,2,1,:])
    axes[1,1].plot(wvs,all_spec_cube[im,3,1,:])

axes[0,0].set_ylim(-2000,30000)
axes[0,1].set_ylim(-2000,30000)
axes[1,0].set_ylim(-2000,30000)
axes[1,1].set_ylim(-2000,30000)
```

[10]:
```
(-2000, 30000)
```



nbsphinx-code-borderwhite

Things look good here, so let's move on. If the spectra aren't aligned, then go back to the last tutorial and remind yourself how to align a spectral cube.

7.3 Step 3: Calculate *q* and *u* for each pair of HWP positions.

Here we create an array of q and u by using the "Flux Ratio" method". The following function automatically sorts the input data based on the HWP angles that you pass to it and subtracts the appropriate spectra from each other (i.e. 45 degrees from 0 degrees, and 67.5 degrees from 22.5 degrees).

[15]:
```
q,u,q_err,u_err,q_ind,u_ind = source_utils.compute_qu_for_obs_sequence(all_spec_cube,hwp_
↪ang)
```

The q and u arrays that are output contain the q and u measurements for each pair of HWP positions. Recall that WIRC+Pol obtains a measurement of both q and u for a given pair of HWP positions. The dimensions of the q and u (also q_err and u_err) are [n_measurements/2, wavelength]. To know the HWP position for a given q or u measurement was made, you can look at the q_ind parameter.

[13]:
```
print(q_ind)
```

```
[0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1]
```

An index of '0' indicates that q measurement was obtained with the HWP at angles 0 and 45 degrees (i.e. the measurement was made with the Top-Left and Bottom-Right spectral trace pair). An index of '1' indicates that the q measurement was obtained with HWP angles of 45 and 67.5 degrees (i.e. the measurement was made with the Top-Right and Bottom-Left spectral trace).

7.3.1 Now calculate *p* and *:math:`theta`*

[14]:
```
p = np.sqrt(q**2+u**2)
theta = 0.5*np.degrees(np.arctan2(u,q))
theta[theta<0] += 180
```

Now we can plot our results

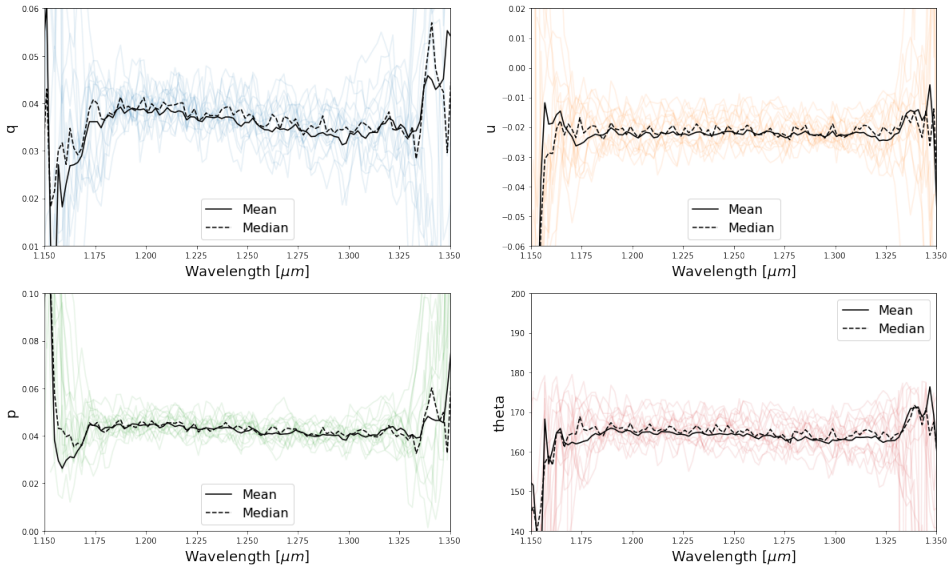
[31]:
```
fig,axes = plt.subplots(2,2,figsize=(20,12))

n_q = q.shape[0]

q_mean = np.nanmean(q[:,:],axis=(0))
u_mean = np.nanmean(u[:,:],axis=(0))
q_mean_err = np.sqrt(np.nanmean(q_err))

p_mean = np.sqrt(q_mean**2+u_mean**2)
theta_mean = 0.5*np.degrees(np.arctan2(u_mean,q_mean))
theta_mean[theta_mean<0] += 180


q_median = np.nanmedian(q[:,:],axis=(0))
u_median = np.nanmedian(u[:,:],axis=(0))
p_median = np.sqrt(q_median**2+u_median**2)
theta_median = 0.5*np.degrees(np.arctan2(u_median,q_median))
theta_median[theta_median<0] += 180

for i in range(15):
```

(continues on next page)

(continued from previous page)

```
    #Plot Q
    axes[0,0].plot(wvs,q[i,:], 'C0', alpha=2/n_q)

    #Plot U
    axes[0,1].plot(wvs,u[i,:], 'C1', alpha=2/n_q)

    #Plot p
    axes[1,0].plot(wvs,p[i,:], 'C2', alpha=2/n_q)

    #Plot theta
    axes[1,1].plot(wvs,theta[i,:], 'C3', alpha=2/n_q)

axes[0,0].plot(wvs,q_mean,'k',label="Mean")
axes[0,1].plot(wvs,u_mean,'k',label="Mean")
axes[1,0].plot(wvs,p_mean,'k',label="Mean")
axes[1,1].plot(wvs,theta_mean,'k', label='Mean')

axes[0,0].plot(wvs,q_median,'k--',label="Median")
axes[0,1].plot(wvs,u_median,'k--',label="Median")
axes[1,0].plot(wvs,p_median,'k--',label="Median")
axes[1,1].plot(wvs,theta_median,'k--',label="Median")

axes[0,0].legend(fontsize=16)
axes[0,1].legend(fontsize=16)
axes[1,0].legend(fontsize=16)
axes[1,1].legend(fontsize=16)

axes[0,0].set_xlim(1.15,1.35)
axes[0,1].set_xlim(1.15,1.35)
axes[1,0].set_xlim(1.15,1.35)
axes[1,1].set_xlim(1.15,1.35)

axes[0,0].set_ylim(0.01,0.06)
axes[0,1].set_ylim(-0.06,0.02)
axes[1,0].set_ylim(0,0.1)
axes[1,1].set_ylim(140,200)

axes[0,0].set_ylabel("q",fontsize=18)
axes[0,1].set_ylabel("u",fontsize=18)
axes[1,0].set_ylabel("p",fontsize=18)
axes[1,1].set_ylabel("theta",fontsize=18)


axes[0,0].set_xlabel(r"Wavelength [$\mu m$]",fontsize=18)
axes[0,1].set_xlabel(r"Wavelength [$\mu m$]",fontsize=18)
axes[1,0].set_xlabel(r"Wavelength [$\mu m$]",fontsize=18)
axes[1,1].set_xlabel(r"Wavelength [$\mu m$]",fontsize=18)
```

[31]:
```
Text(0.5, 0, 'Wavelength [$\\mu m$]')
```



nbsphinx-code-borderwhite

7.4 Step 4: Calibrate q and u

Before using this data, we want to apply our instrument calibration that accounts for non-ideal polarimetric efficiencies and crosstalks. The details of how this correction was determined will be published in an upcoming paper.

The correction was calculated by treating each trace pair (i.e. [top left,bottom right] and [top right,bottom left]) as independent polarimeters, so we first need to split them up to do the correction.

[16]:
```
#Well apply the correction only to the mean values.

#For the first trace pair
q_mean0 = np.nanmean(q[q_ind==0],axis=(0))
u_mean0 = np.nanmean(u[u_ind==0],axis=(0))
q_mean_err0 = np.sqrt(np.nanmean(q_err[q_ind==0]**2,axis=0))
u_mean_err0 = np.sqrt(np.nanmean(u_err[u_ind==0]**2,axis=0))

p_mean0 = np.sqrt(q_mean0**2+u_mean0**2)
theta_mean0 = 0.5*np.degrees(np.arctan2(u_mean0,q_mean0))
theta_mean0[theta_mean0 < 10] += 180

#For the second trace pair
q_mean1 = np.nanmean(q[q_ind==1],axis=(0))
u_mean1 = np.nanmean(u[u_ind==1],axis=(0))
q_mean_err1 = np.sqrt(np.nanmean(q_err[q_ind==1]**2,axis=0))
u_mean_err1 = np.sqrt(np.nanmean(u_err[u_ind==1]**2,axis=0))

p_mean1 = np.sqrt(q_mean1**2+u_mean1**2)
theta_mean1 = 0.5*np.degrees(np.arctan2(u_mean1,q_mean0))
theta_mean1[theta_mean1 < 10] += 180


#We'll just apply it to the mean q and u
q_cal0,u_cal0,q_err0,u_err0 = wc.calibrate_qu(wvs,q_mean0,u_mean0,q_mean_err0,u_mean_
```

(continues on next page)

(continued from previous page)

```
↪err0,0)
q_cal1,u_cal1,q_err1,u_err1 = wc.calibrate_qu(wvs,q_mean1,u_mean1,q_mean_err1,u_mean_
↪err1,1)
#The last argument in the above function corresponds to which qind (i.e. trace pair) you
↪'re
#currently calibrating

#Calculat the calibrated p and theta
p_cal0 = np.sqrt(q_cal0**2+u_cal0**2)
theta_cal0 = 0.5*np.degrees(np.arctan2(u_cal0,q_cal0))
theta_cal0[theta_cal0<10] += 180

p_cal1 = np.sqrt(q_cal1**2+u_cal1**2)
theta_cal1 = 0.5*np.degrees(np.arctan2(u_cal1,q_cal1))
theta_cal1[theta_cal1<10] += 180
```

Now let's compare the before and after and include the expected polarization.

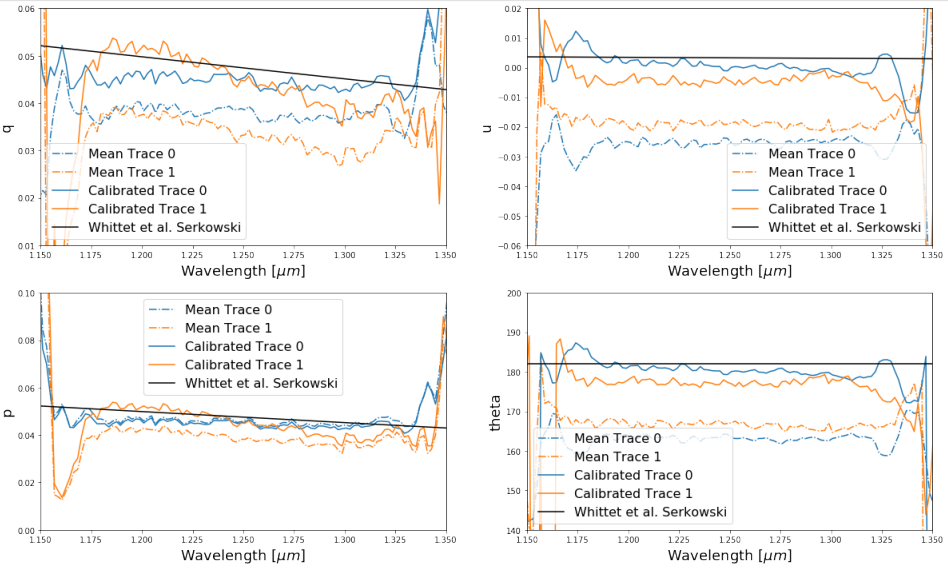
[49]:
```
#Get the Serkowski law for Elias 2-14

p_elias,q_elias,u_elias = source_utils.serkowski_polarization(wvs,0.74,0.0656,1.17,
↪theta=182)


fig,axes = plt.subplots(2,2,figsize=(20,12))

axes[0,0].plot(wvs,q_mean0,'-.',label="Mean Trace 0")
axes[0,1].plot(wvs,u_mean0,'-.',label="Mean Trace 0")
axes[1,0].plot(wvs,p_mean0,'-.',label="Mean Trace 0")
axes[1,1].plot(wvs,theta_mean0,'-.', label='Mean Trace 0')

axes[0,0].plot(wvs,q_mean1,'-.',label="Mean Trace 1")
axes[0,1].plot(wvs,u_mean1,'-.',label="Mean Trace 1")
axes[1,0].plot(wvs,p_mean1,'-.',label="Mean Trace 1")
axes[1,1].plot(wvs,theta_mean1,'-.', label='Mean Trace 1')

axes[0,0].plot(wvs,q_cal0,'C0',label="Calibrated Trace 0")
axes[0,1].plot(wvs,u_cal0,'C0',label="Calibrated Trace 0")
axes[1,0].plot(wvs,p_cal0,'C0',label="Calibrated Trace 0")
axes[1,1].plot(wvs,theta_cal0,'C0',label="Calibrated Trace 0")

axes[0,0].plot(wvs,q_cal1,'C1',label="Calibrated Trace 1")
axes[0,1].plot(wvs,u_cal1,'C1',label="Calibrated Trace 1")
axes[1,0].plot(wvs,p_cal1,'C1',label="Calibrated Trace 1")
axes[1,1].plot(wvs,theta_cal1,'C1',label="Calibrated Trace 1")

axes[0,0].plot(wvs,q_elias,'k',label="Whittet et al. Serkowski")
axes[0,1].plot(wvs,u_elias,'k',label="Whittet et al. Serkowski")
axes[1,0].plot(wvs,p_elias,'k',label="Whittet et al. Serkowski")
axes[1,1].plot(wvs,wvs*0+182,'k',label="Whittet et al. Serkowski")
```

(continues on next page)

(continued from previous page)

```
axes[0,0].legend(fontsize=16)
axes[0,1].legend(fontsize=16)
axes[1,0].legend(fontsize=16)
axes[1,1].legend(fontsize=16)

axes[0,0].set_xlim(1.15,1.35)
axes[0,1].set_xlim(1.15,1.35)
axes[1,0].set_xlim(1.15,1.35)
axes[1,1].set_xlim(1.15,1.35)

axes[0,0].set_ylim(0.01,0.06)
axes[0,1].set_ylim(-0.06,0.02)
axes[1,0].set_ylim(0,0.1)
axes[1,1].set_ylim(140,200)

axes[0,0].set_ylabel("q",fontsize=18)
axes[0,1].set_ylabel("u",fontsize=18)
axes[1,0].set_ylabel("p",fontsize=18)
axes[1,1].set_ylabel("theta",fontsize=18)


axes[0,0].set_xlabel(r"Wavelength [$\mu m$]",fontsize=18)
axes[0,1].set_xlabel(r"Wavelength [$\mu m$]",fontsize=18)
axes[1,0].set_xlabel(r"Wavelength [$\mu m$]",fontsize=18)
axes[1,1].set_xlabel(r"Wavelength [$\mu m$]",fontsize=18)
```

[49]:
```
Text(0.5, 0, 'Wavelength [$\\mu m$]')
```



nbsphinx-code-borderwhite

With this dataset we're not exactly matching as well as we should be to the Whittet et al values, but we're much closer! The mismatch is perhaps related to the background subtraction used to extract the data or possibly the quality of our current instrument mode – a work in progress!

If you have a set of polarized standard measurements you can create your own calibration using the function: `wirc_drp.utils.calibration.make_instrument_calibration()`

Congrats on making it through the tutorial!

CHAPTER

EIGHT

TUTORIAL 4: CONVENIENCE FUNCTIONS TUTORIAL

This tutorial shows you how to use some of our convenience functions that package all the previous tutorials into just a few lines. Since a lot of the action will be going on behind the scenes we highly recommend that you complete the first three tutorials before using these functions. Even after that, we recommend you dig into the code a bit to see what it's doing and understand some of the default options better.

There are two kinds of convenience functions: dataset reduction functions and plotting functions.

8.1 Dataset Reduction Functions

Dataset reduction functions are here to help you reduce a complete dataset with ease. This requires the dataset to be generally well behaved, and overall this best works when you have good background frames.

The functions generally follow the same form: they take input file lists, extract the spectra for a single source and then save a new wirc object fits file for each input file to the output directory. Each background method that you choose to use will create its own subdirectory in the output directory. Within that subdirectory you'll also find a directory that holds an extraction summary image for each file. These images can help diagnose extraction issues.

The basic usage of these functions is displayed below, but dig in to the code options a bit more for advanced usage.

8.1.1 reduce_dataset

The reduce_datset function accepts a filelist and source position as input and automatically extracts the spectra

Here's an example use:

[ ]:
```
import numpy as np
import wirc_drp.dataset as wd

#This is a very simple function just to help gather your files.
def make_list(start, stop, path, prefix):
    file_list = [path+prefix+str(x).zfill(4)+'_cal.fits' for x in np.arange(start,␣
↪stop+1)]
    return np.array(file_list)

#Generate your file list
filelist = make_list(2334, 2373, '/scr/data/Calibrated Files/20190319/Auto_Reduced/',
↪'wirc')


wd.reduce_dataset(filelist, #The file list
```

(continues on next page)

(continued from previous page)

```
                [758,1093], #The detector coordinates of the 0th order of your source [x,
↪y] in pixels
                bkg_fnames = None, #In this case we don't have a list of background␣
↪files to input
                output_path = "/scr/mblanchaer/Data/WIRC+Pol/Elia2-25/190319/", #The␣
↪output data path
                verbose=False, #We don't want ALL of the status outputs
                less_verbose=True, #But we want some
                bkg_methods = ["cutout_median"], #We'll just use the cutout_median␣
↪background subtraction here
                in_slit=False, #The source is not in the slit
                parallel=True, #Let's reduce the files in parallel to make things go␣
↪faster
                n_processes=None, #Let's use the maximum number of processes (minues 1)␣
↪that our machine allows
                fixed_width=7) #We'll use a fixed width for the extraction of each file -
↪> this improves stability
```

You could also use it with a set of background files:

[ ]:
```
#Generate your file list
filelist = make_list(2334, 2373, '/scr/data/Calibrated Files/20190319/Auto_Reduced/',
↪'wirc')

#Generate your background file list
bkg_filelist = make_list(2329, 2333, '/scr/data/Calibrated Files/20190319/Auto_Reduced/',
↪ 'wirc')

wd.reduce_dataset(filelist,
                [758,1093],
                bkg_fnames = bkg_filelist, #This time we include the background file list
                output_path = "/scr/mblanchaer/Data/WIRC+Pol/Elia2-25/190319/",
                verbose=False,
                less_verbose=True,
                bkg_methods = ["cutout_median","scaled_bkg"], #There are now more␣
↪background subtraction options open
                                                              #to us since we have the␣
↪backgroud files
                in_slit=False,
                parallel=True,
                n_processes=None,
                nclosest = 10, #This allows you to set a limit on the number of␣
↪background frames you want to use,
                               #and picks the closest in time
                fixed_width=7)
```

8.1.2 reduce_ABAB_dataset

This function is very much like reduce_dataset, except that it doesn't accept a list of background files, but instead assumes that you observed in an ABAB dither pattern.

Here's an example use:

```
[ ]: filelist = make_list(72, 80, '/scr/data/Calibrated Files/20200807/Auto_Reduced/', 'image
     ↪')

     wd.reduce_ABAB_dataset(filelist, #The file list
                            [720,1100],#The detector coordinates of the 0th order of your␣
     ↪source [x,y] in the A position
                            [754,1102],#The detector coordinates of the 0th order of your␣
     ↪source [x,y] in the B position
                            output_path = "/scr/mblanchaer/Data/WIRC+Pol/Elia2-25/200807/",
     ↪#The output data path
                            verbose=False, #We don't want ALL of the status outputs
                            less_verbose=True, #But we want some
                            bkg_methods = ["scaled_bkg","cutout_median"],
                            parallel=True,
                            fixed_width=7)
```

8.1.3 reduce_dataset_distance

In this version the pipeline automatically finds your source (based on your initial guess) and uses any frame that is a set number of pixels away as part of the background library.

```
[ ]: filelist = make_list(72, 80, '/scr/data/Calibrated Files/20200807/Auto_Reduced/', 'image
     ↪')

     wd.reduce_dataset_distance(filelist, #The file list
                            [720,1100],#The average detector coordinates of the 0th order of␣
     ↪your source
                            output_path = "/scr/mblanchaer/Data/WIRC+Pol/Elia2-25/200807/",
     ↪#The output data path
                            verbose=False, #We don't want ALL of the status outputs
                            less_verbose=True, #But we want some
                            bkg_methods = ["scaled_bkg","cutout_median"],
                            parallel=True,
                            fixed_width=7)
```

8.2 Plotting Functions

There are also a few built-in functions that can be used to give you a quick look at some of your data.

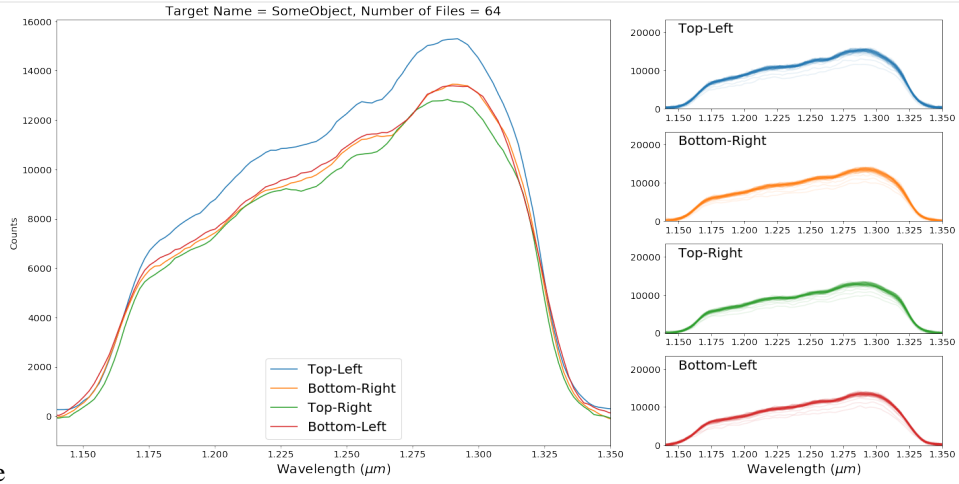
8.2.1 Extraction Sumary

The first summary plot shows you the mean extracted spectra from each of the four traces as well as a set of plots for each trace showing all of the extracted spectra. This can help you spot a significant number of outliers in your dataset. Often having just a few doesn't skew your results a lot.

[7]:
```
data_directory = "/scr/mblanchaer/Data/WIRC+Pol/CFHT_BD4/200905/scaled_bkg/"
```

[9]:
```
import wirc_drp.dataset as wd
wd.plot_dataset_extraction_summary(data_directory,save=False,verbose=True,target_name=
↪"SomeObject")
```

```
64
Found 64 files
```



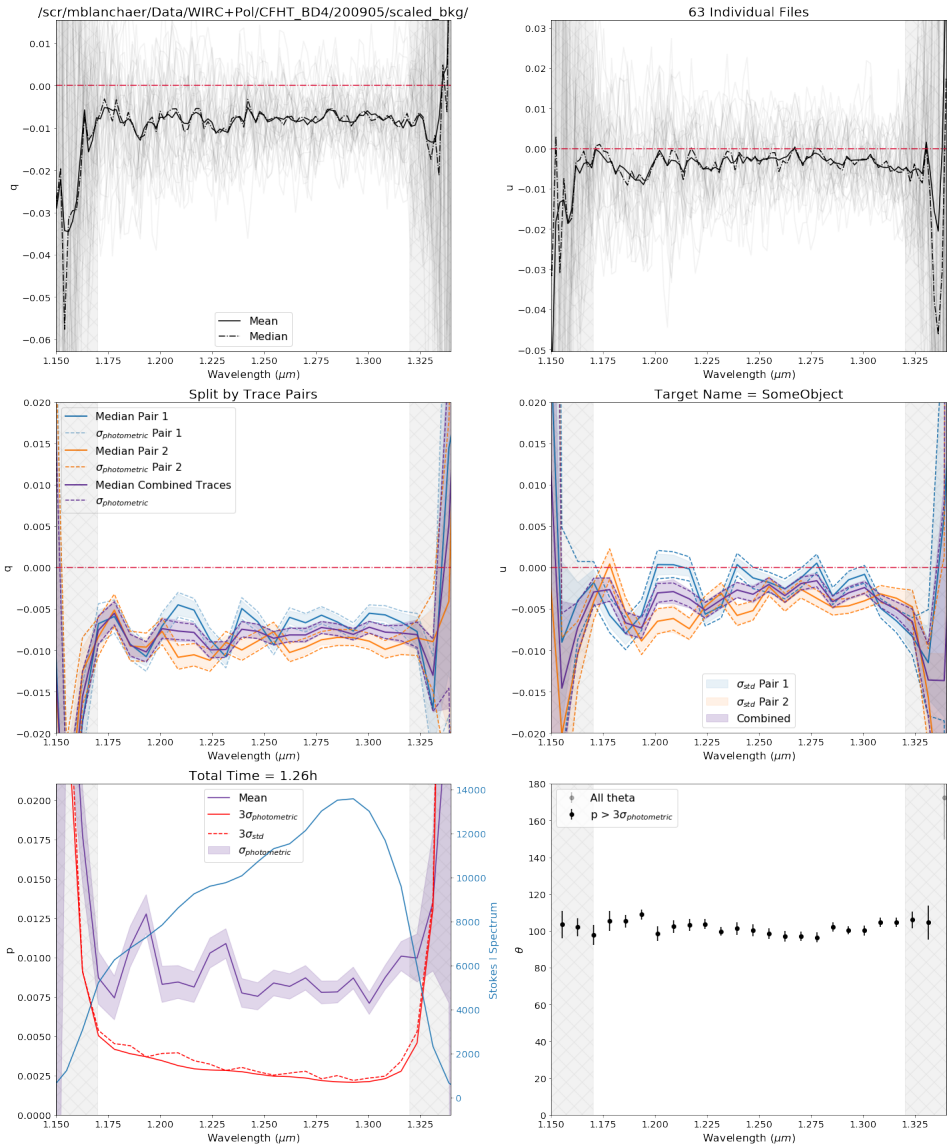
nbsphinx-code-borderwhite

8.2.2 Polarization Summary

This plot provides a summary of the polarized spectrum measured from your dataset. The top row shows all of the individual q and u measurements, as well as the mean and median. The middle row shows the median values for each of the trace pairs individually and combined, and displays the 1-sigma error ranges based on the standard error on the mean of each wavelength bin. The bottom row shows p and theta for the combined.

Note: by default this is uncalibrated data (i.e. no system Mueller matrix has been applied). You can turn on the calibration by setting `calibrate=True` as a keyword argument, but this is currently an experimental feature.

[11]:
```
wd.plot_dataset_polarization_summary(data_directory,save=False,verbose=True,target_name=
↪"SomeObject")
```



nbsphinx-code-borderwhite

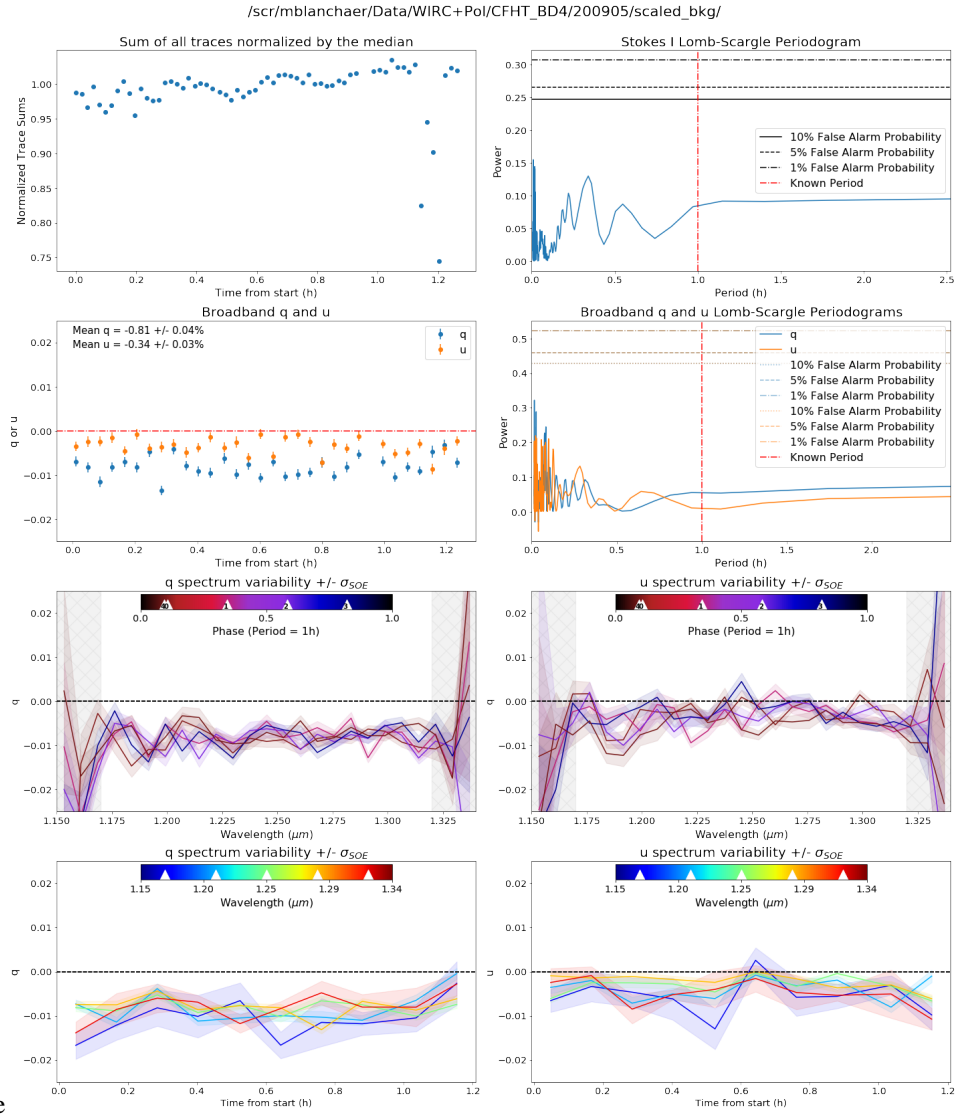
8.2.3 Variability Summary

The variability summary plot looks for variability in your dataset. In the top row we look at the sum of your trace spectra over time and take a lomb-scargle. In the second row, we do the same for q and u (i.e. the broadband value for q and u). In the third row, we bin together your spectra in time into `n_time_bins` bins. In the fourth row we bin together your spectra in spectral bins and plot them against time.

Among other input arguments, you can also include a known period in hours if your object is known to be variable.

[18]:
```
known_period=1
wd.plot_dataset_variability_summary(data_directory,save=False,known_period=known_period,
                                    n_time_bins=5,target_name="SomeTarget")
```

```
(63,)
3
```

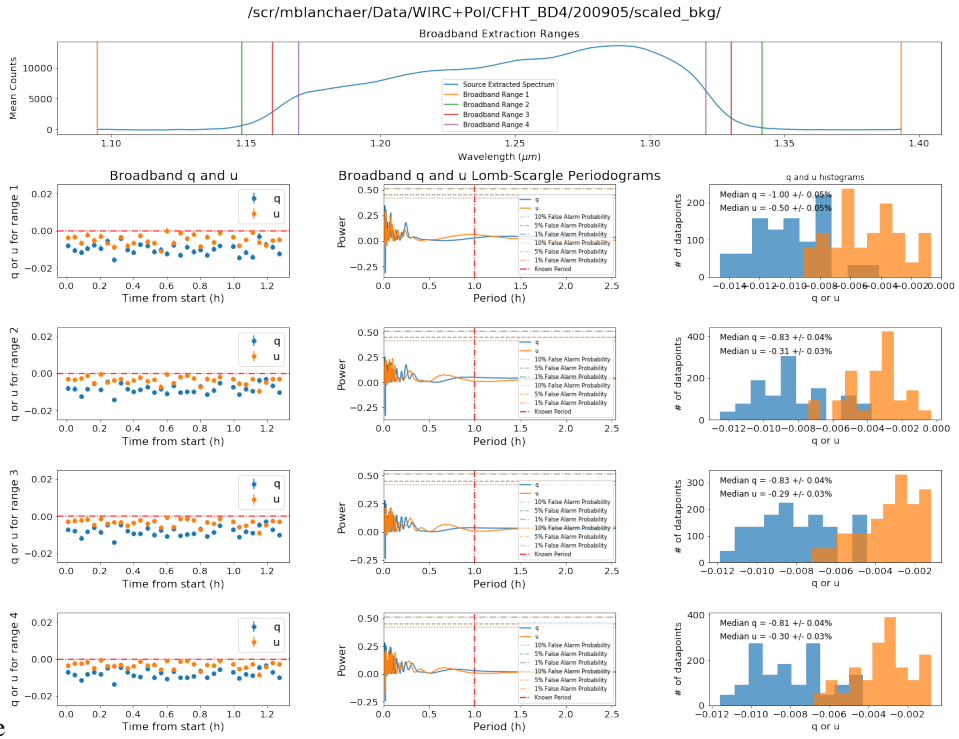


nbsphinx-code-borderwhite

8.2.4 Broadband Summary

We can also examine the different results one gets when varying the location of where you start to sum your flux for broadband measurements.

[22]:
```
wd.plot_dataset_broadband_summary(data_directory,save=False,known_period=known_period,
↪target_name="Some Object",
                                  calibrate=False)
```



nbsphinx-code-borderwhite

[ ]:

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search