MOSFIRE Spectroscopy Data Reduction Pipeline:

User Manual

The MOSFIRE DRP development team

August 1st, 2015

# Changes in version 2015A

## New features

* Reduction of long2pos and long2pos\_specphot
* Reduction of longslit data
* Automatic generation of driver file
* Logging and diagnostic information on screen and on disk
* Package-style installation as a Ureka sub-package
* Support for Ureka 1.5.1

## Improvements and bug fixes

* Fix incorrect determination of the slit parameters which prevented the use of large slits
* Fix incorrect determination of the average wavelength dispersion for the long2pos mode
* Added ability of specifying the output name of the files
* Improved robustness of non-interactive wavelength solution, added possibilty of switching from interactive to non-interactive during the reduction, added k-sigma clipping of the sky or arc lines
* Fixed the problem with the interactive wavelength window not closing at the end of the fit
* Fixed the problem with the interactive fitting window showing up empty on the first fit (no need to use the x key to unzoom)
* Added procedure to fix the header of old images acquired with an outdated version of long2pos
* Disabled cosmic ray rejection for the case of less than 5 exposures
* There is no need to specify one of the observations in Rectify: Rectify will use the first of the files listed in the Offset files.

# Preface

This manual describes the installation and usage of the MOSFIRE data reduction pipeline on a unix-like computer. Although primarily tested and developed on a Mac, the pipeline operates on both OSX and Linux systems. In the section 4, we describe an installation procedure for a Mac OSX system. Later sections describe code usage, execution, and outputs.

The MOSFIRE spectrograph data reduction pipeline was architected by the MOSFIRE commissioning team and written by Nick Konidaris with extensive checking and feedback from Chuck Steidel and other MOSFIRE team members. The pipeline is maintained on an online code repository <http://mosfire-datareductionpipeline.github.io/MosfireDRP/>. Please use this website to track issues and and submit requests.

# Installing the Data Reduction Pipeline

As of version 2015A, the pipeline can be installed as Ureka package. If you prefer to use this installation procedure, download the pipeline tar file and unpack it. Then execute:

ur\_setup

Go to the mosfire python directory where the setup.py file is located (it is the main directory above the MOSFIRE and apps directories). Run:

python setup.py install

This will copy the MOSFIRE pipeline as a python package located into your ureka enviornment.

Alternatively, if you would like to modify the MOSFIRE pipeline files, run the following instead:

python setup.py develop

This will make symoblic links to the MOSFIRE files instead of copying them. Your changes to the pipeline file will now automatically be used when loading the MOSFIRE package.

Directories created by you

* DATA – sub directory in which you can store your data. This is not a necessary sub-directory but may help you manage files. Raw data may be stored in other areas on your disk.
* REDUX – sub directory where reductions will be stored. Also not critical, but helpful.

From now on, if you want to run any pipeline commands, you will always execute “mospy” as seen in our first step in section 5 below. Remember to run ur\_setup before running the MOSFIRE pipleine.

## Alternate Installation Method

If you prefer the previous installation method, follow these instructions:

1. The pipeline relies on the Ureka Python distribution produced by STScI and Gemini Observatory.

<http://ssb.stsci.edu/ureka/>

The DRP was developed using UREKA version 1.0. Navigate to the 1.0 distribution using the url listed above. Follow the instructions at the links to install the package. The UREKA instructions indicate that you need to run ur\_setup to put ureka in the path. This is automatically completed when you run the drp and it is found in the mospy code. However, if you want to test the ureka package yourself, you will need to run ur\_setup manually. The latest version of Ureka that is confirmed to work with the pipeline is 1.5.1

2. Download the pipeline.

1. Start an xterm session on your local machine
2. Run “cd” to navigate to the home directory
3. Download either the .zip file, or the .tar.gz file from the website <https://keck-datareductionpipelines.github.io/MosfireDRP/>

Note that this is the stable and supported version of the pipeline.

1. Alternatively, if you are a github user, you can just clone the repository using:

<https://github.com/keck-DataReductionPipelines/MosfireDRP.git>

This is the development version, and it is NOT supported.

1. Expand the zip or tar file and rename the resulting directory. For example:

mkdir ~/MOSFIRE

mv MosfireDRP-1.1 ~/MOSFIRE/DRP\_CODE

1. cd ~/MOSFIRE/DRP\_CODE to navigate to that directory.

3. Create sub directories for raw data, and reduced data. These sub directories are not specific. You can set up sub directories any way you would like. For the purposes of this manual, we have choosen generic directory names. You may choose to store the raw and reduced data using andy directory structure you would prefer. For our example, we created:

1. a raw data directory in the code repository:

mkdir ~/MOSFIRE/DRP\_CODE/DATA

1. a reduction directory in the code repository that will store reduced data:

mkdir ~/MOSFIRE/DRP\_CODE/REDUX

4. Copy the mospy file into your bin dir

1. Navigate to the newly creted bin dir:

cd ~/MOSFIRE/DRP\_CODE/bin

1. Copy the mospy executeable to the bin dir

cp ../apps/mospy .

5. edit mospy in your bin dir and update a few lines of code. Using yoru favorite editor (emacs ../bin/mospy)

1. Update the path for the ur\_setup. Replace /home/npk/.ureka/ur\_setup with your / *your\_full\_path\_name*/.ureka/ur\_setup full path.
2. Update the path for the ur\_forget. Replace /home/npk/.ureka/ur\_setup with /*your\_full\_path\_name*/.ureka/ur\_forget
3. Update the MOSPATH with the full path to the source code directory. Replace /src2/mosfire/DRP/mosfire with /*your\_full\_path\_name*/MOSFIRE/DRP\_CODE

As an example: the original file might look like the following (items to change in blue text):

#Update the full path to the ureka install for the

# two aliases below.

alias ur\_setup 'eval `/home/npk/.ureka/ur\_setup -csh \!\*`'

alias ur\_forget 'eval `/home/npk/.ureka/ur\_forget -csh \!\*`'

# If pythonpath is not previously defined, define it so that

# the setenv works below..

if (! $?PYTHONPATH ) setenv PYTHONPATH

#Update the full path to the mosfire DRP code repository

# example: /src2/mosfire/DRP/mosfire change to /Users/myname/MOSFIRE/DRP\_CODE

# in which the sub dirs drivers, apps, badpixel, etc. live

setenv MOSPATH /scr2/mosfire/DRP/mosfire

setenv PYTHONPATH ${PYTHONPATH}:${MOSPATH}

And the modified version for an observers particular setup may look something like this (items changed in green text):

#Update the full path to the ureka install for the

# two aliases below.

alias ur\_setup 'eval `/Users/mkassis/.ureka/ur\_setup -csh \!\*`'

alias ur\_forget 'eval `/Users/mkassis/.ureka/ur\_forget -csh \!\*`'

# If pythonpath is not previously defined, define it so that

# the setenv works below..

if (! $?PYTHONPATH ) setenv PYTHONPATH

# Update the full path to the mosfire DRP code repository

# example: /src2/mosfire/DRP/mosfire

# in which the sub dirs drivers, apps, badpixel, etc. live

setenv MOSPATH /Users/mkassis/Documents/KeckInstrs/MOSFIRE/DRP\_CODE\_March2014/

setenv PYTHONPATH ${PYTHONPATH}:${MOSPATH}

6. Ensure that it is executable: chmod +x mospy

7. Update your .cshrc file with the code bin dir in the path. Add the following line to your .cshrc file:

set path = ( #mosfire\_drp\_bin\_dir# $path )

e.g. set path = ( ~/MOSFIRE/DRP\_CODE/bin $path )

If you do not normally run csh or tcsh, you may not have a .cshrc file. You will need to create one or download an example file like this one: <http://www2.keck.hawaii.edu/inst/mosfire/.cshrc>. The .cshrc file must be in your home directoyr. By default, MacOSX does not show files that start with a “.” But you can access them via the terminal.

For a bash shell:

# Adding MOSFIRE pipeline

PATH="/pathtomosfiredrp/MOSFIRE/DRP\_CODE/bin:${PATH}"

export PATH

8. Now source your .cshrc file: source ~/.cshrc which will pyt your bin dir into your executable path.

The installation is now complete. Take a moment to inventory your directory structure.

DRP\_CODE – Main Code Directory containing all sub-dirs for executeable code and in our example the raw and reduced sub-directories.

* MOSFIRE – directory containing the reduction code
* apps – directory containing a few additional applications:
  + - what – useful pretty printer for files
    - handle – the entry point for creating driver files (more later)
* badpixels – directory containing badpixel maps.
* Drivers – directory containing example driver files. These files are used to initiate the redution process and you will modify them for your specific data sets. This will be discussed in more detail later.
  + Driver.py – used for YJH reductions
  + K\_driver.py – Contains code specific to K band observations
  + Longslit\_driver.py – Longslit reductions
  + Long2pos\_driver.py – long2pos and long2pos\_specphot reductions
* Platescale – contains a file that describes the detector plate scale

Directories created by you

* DATA – sub directory in which you can store your data. This is not a necessary sub-directory but may help you manage files. Raw data may be stored in other areas on your disk.
* REDUX – sub directory where reductions will be stored. Also not critical, but helpful.
* bin – has the modified mospy executable command

From now on, if you want to run any pipeline commands, you will always execute “mospy” as seen in our first step in section 5 below.

# Retrieve your data

Before running the drp, you will need a set of spectroscopic data to reduce that includes flats, science observations, and if the observations are K-band, arcs and thermal flats. NOTE: You **must** preserve Keck’s file naming convention as the DRP uses the file name to parse data sets. The standard naming convention is mYYMMDD\_####.fits.

If you need to retrieve your data, you may either use a secure copy (scp) assumine your data is still accessible from the Keck/MOSFIRE data directory (contact your SA if you need assistance) or use [**KOA –the Keck Observatory Archive**](https://koa.ipac.caltech.edu/cgi-bin/KOA/nph-KOAlogin) to navigate to the KOA log in page. From there, KOA has forms where you specify the data to retrieve and will create a tar ball for you to download.

A useful tool is the file translator script that will convert your KOA file names to the standard filenames as they were written to disk during your observing session ([koa\_translator](ftp://ftp.keck.hawaii.edu/pub/ObservingTools/common)). Again, your filenames must preserve the standard naming convention and the koa\_translator script does this for you.

If you do not have data of your own and wish to download the example:

1. Grab the data from: <http://mosfire.googlecode.com/files/DRP_Test_Case_Hband.zip>.
2. Move the test case data to the newly created data dir

**mv ~/Downloads/DRP\_Test\_Case\_Hband.zip**

**~/MOSFIRE/DRP\_CODE/DATA/.**

1. unzip the DRP test case file: **unzip DRP\_Test\_Case\_Hband.zip**

Again, although any data directory is valid, for the purposes of this document, we will assume you have put it in the DATA sub-directory.

# DRP First Reduction Step: Handle

Now that you have data to reduce, we need to set up the pipeline with the appropriate files so that the drp knows what files to use in the reduction. The handle step will parses the FITS header information and determine what files are associated with each of your masks. Because the DRP no longer has a designated output directory, you will need to run handle in your designated reduction sub-directory (REDUX in our example).

Steps to perform

1. cd ~/MOSFIRE/DRP\_CODE/REDUX # Go to your output directory
2. mospy handle /home/yourhomedir/MOSFIRE/DRP\_CODE/2014may08\*fits  
   … A lot of data summarizing the observations is outputed. This includes a table of the observations:

Example output:

…..

m130514\_0132 Flat:mos Y mosmaskA 16.0 s mosmaskA Y YJ

m130114\_0133 Flat:mos Y mosmaskA 16.0 s mosmaskA Y YJ

m130114\_0134 Flat:mos Y mosmaskA 16.0 s mosmaskA Y YJ

m130114\_0135 Flat:mos Y mosmaskA 16.0 s mosmaskA Y YJ

m130114\_0136 Flat:mos Y mosmaskA 16.0 s mosmaskA Y YJ

m140114\_0137 Flat:mos Y mosmaskA 16.0 s mosmaskA Y YJ

…..

and file lists that organize the observation types:

….

mosmaskA /2013jan14/Y/Unknown.txt

mosmaskA /2013jan14/Y/Align.txt

mosmaskA /2013jan14/Y/MIRA.txt

mosmaskA /2013jan14/Y/Ne.txt

mosmaskA /2013jan14/Y/Offset\_2.txt

mosmaskA /2013jan14/Y/Offset\_-2.txt

mosmaskA /2013jan14/Y/Flat.txt

mosmaskA /2013jan14/Y/Image.txt

mosmaskA /2013jan14/Y/FlatThermal.txt

mosmaskA /2013jan14/Y/Dark.txt

mosmaskA /2013jan14/Y/Ar.txt

mosmaskA 2013jan14/Y/Aborted.txt

…..

The handle step creates a set of directories organized as

[maskname]/[date]/[band]/

Aborted.txt: Aborted files

Align.txt: Alignment frames   
Ar.txt: Argon spectra

Dark.txt: Darks

Flat.txt: Flat fields

FlatThermal.txt: Thermal Flats (lamps off)

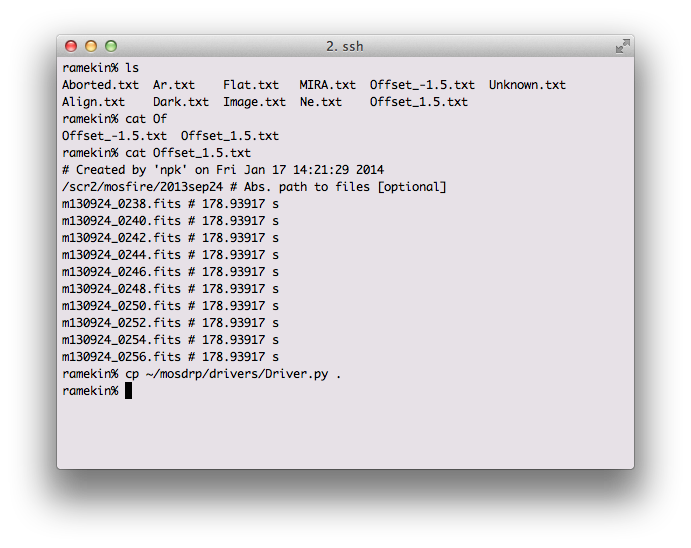
Image.txt: Imaging mode

MIRA.txt: MIRA focus images

Ne.txt: Neon lamp spectra

Unknown.txt: Unknown files   
Offset\_[p].txt: Science frames

The output directory structure is designed to make finding reduced data easy, and to seaprate reductions of the same mask across multiple dates. Below is a screen shot showing the example output from an Offset\*.txt file.



For longslit observations, separate offsets files are created for each object, but the same offset files are used if the same object is observed many times during the night. You might want to separate the different observations of the same object.

For long2pos observations, again different offset files are created for each object. Besides, the suffixes \_PosA and \_PosC are added to the offset files to identify the two left and right positions used in the observations.

The following section describes in details how to use the driver file to control the pipeline processes, and points at a number of standard driver files that we provide for reference.

# Automatic generation of the driver file

The pipeline is able to produce a driver file automatically for most cases, thus removing the need to copy one of the standard files and manually edit it.

To generate the driver file, go to the directory where your Offset files live, and where the reduction is going to happen and type:

mospy AutoDriver

This will generate a file called Driver.py, which you should inspect before running it. Highly specialized cases such as particular combinations of sky lines and arcs might not be dealt with correctly. Note that the automatic generation of the driver file works for long2pos and longslit as well.

To handle special cases, the automatic generation of the driver file makes a number of assumptions, that might not be correct for your science case.

* + 1. If either Ar.txt or Ne.txt or both are available, they are being used.
    2. If the band is K, and FlatThermal.txtis available, it is used
    3. For long2pos: if no arcs are available, only specphot in non spectrophotometric mode can be reduced and the pipeline will attempt to use the sky lines. Note that is likely to fail, as sky lines are too faint in short exposures for an accurate wavelength determination. The spectrophotometric mode contains wide slits that cannot be reduced using sky lines only.
    4. In longslit, the pipeline will try to determine the size of the slit using the mask name. For example, if the maskname is LONGSLIT-3x0.7, the pipeline assumes that you have used 3 slits to generate the longslit and that they are centered around the middle line of the detector.
    5. If any of the mandatory observations are missing (such as the flat fields), the pipeline will still generate a Driver.py file, but it will contain warnings about the missing files, and it will NOT run correctly.
    6. If multiple observations of different stars are done in long2pos or in longslit mode, the pipeline will generate multiple driver files, one for each object. If the same object is observed multiple times during the same night, all the observations will end up in the same driver file. If you are observing a telluric standard at different times and you need to have separate spectra, you need to manually create Offset files and Driver files.

# The driver.py file

The driver file controls all the pipeline steps, and in the drivers sub-directory, you will find a number of driver files: Driver.py, K\_Driver.py, Long2pos\_driver.py, and Longslit\_Driver.py. The Driver and K\_Driver will reduce your science data for bands Y,J, and H (this includes the sample data set). The K band requires a special approach because there are too few bright night-sky emission lines at the red end and so the K\_Driver synthesizes arclamps and night sky lines. The Long2pos\_driver.py handles long2pos and long2pos\_specphot observations, while the Longslit\_driver.py deals with observations of single objects using a longslit configuration.

The driver.py files included with the code download contains execution lines that are commented out. For this example, we will run the driver file one line at a time, but as you become familiar with the DRP process, you will develop your own driver file execution sequencing. Although in the future we hope to further automate the driver file, currently some steps require you to update the inputs with filenames created from previous steps.

Below is a driver.py file:

import os, time

import MOSFIRE

from MOSFIRE import Background, Combine, Detector, Flats, IO, Options, \

Rectify

from MOSFIRE import Wavelength

import numpy as np, pylab as pl, pyfits as pf

np.seterr(all="ignore")

#Update the **insertmaskname** with the name of the mask

#Update **S** with the filter band Y,J,H,or K

maskname = **'insertmaskname'**

band = **'S'**

flatops = Options.flat

waveops = Options.wavelength

obsfiles = [**'Offset\_1.5.txt'**, **'Offset\_-1.5.txt'**]

#Flats.handle\_flats('Flat.txt', maskname, band, flatops)

#Wavelength.imcombine(obsfiles, maskname, band, waveops)

#Wavelength.fit\_lambda\_interactively(maskname, band, obsfiles,

#waveops)

#Wavelength.fit\_lambda(maskname, band, obsfiles, obsfiles,

#waveops)

#Wavelength.apply\_lambda\_simple(maskname, band, obsfiles, waveops)

#Background.handle\_background(obsfiles,

#**'lambda\_solution\_wave\_stack\_H\_m130429\_0224-0249.fits',**

#maskname, band, waveops)

redfiles = ["eps\_" + file + ".fits" for file in obsfiles]

#Rectify.handle\_rectification(maskname, redfiles,

#  **"lambda\_solution\_wave\_stack\_H\_m130429\_0224-0249.fits",**

# band,

# **"/scr2/npk/mosfire/2013apr29/m130429\_0224.fits",**

# waveops)

#

To set up your driver file do the following

1. Navigate to the desired output directory created by handle:

cd ~/MOSFIRE/DRP\_CODE/REDUX/testmask/2013sep24/H

1. Copy the appropriate driver file:
   1. cp ~/ MOSFIRE/DRP\_CODE/drivers/Driver.py .
   2. *NOTE:* If you are observing a K band mask you’ll want to copy the K\_driver.py file over.
2. Edit driver.py (see bold text in driver file example)
   1. Update maskname
   2. Update band to be Y,J,H
   3. Update the Offset\_#.txt name. Handle creates offset files with names that are specific to the nod throw. The default driver file uses 1.5 arcsec offsets in the file name.

In the sections that follow, we will describe the function and outputs of the commented lines found in the driver file starting with the creation of flats.

If you prefer to override the standard naming convention of the output files, you can specify

target = “targetname”

at the beginning of the driver file. If you do so, remember to also add target=target to both the Background and Rectify steps. Example:

Background.handle\_background(obsfiles,

'lambda\_solution\_wave\_stack\_H\_m150428\_0091-0091.fits',

maskname, band, waveops, target=target)

# Flats

The first action the driver file will take is to generate a pixel flat and slit edge tracing. To initiate the flat generation, uncomment the line below in the

* Driver.py file:

#Flats.handle\_flats('Flat.txt', maskname, band, flatops)

* and in your xterm run the DRP

> mospy Driver.py

Example output from the xterm session

> mospy Driver.py  
… Truncated output …  
Flat written to combflat\_2d\_H.fits

00] Finding Slit Edges for BX113 ending at 1901. Slit composed of 3 CSU slits

01] Finding Slit Edges for BX129 ending at 1812. Slit composed of 2 CSU slits

02] Finding Slit Edges for xS15 ending at 1768. Slit composed of 1 CSU slits

Skipping (wavelength pixel): 10

03] Finding Slit Edges for BX131 ending at 1680. Slit composed of 2 CSU slits

The slit names output to the screen should look familiar as they originated from the mask design process. The output files from this process are the following:

|  |  |
| --- | --- |
| Filename | Contains |

combflat\_2d\_J.fits FITS image of the flats

flatcombine.lst The list of files used in the creation of the flat. Contains the full path name to the files.

pixelflat\_2d\_J.fits FITS image of the normalized flat. This is the flat used in other redution steps.

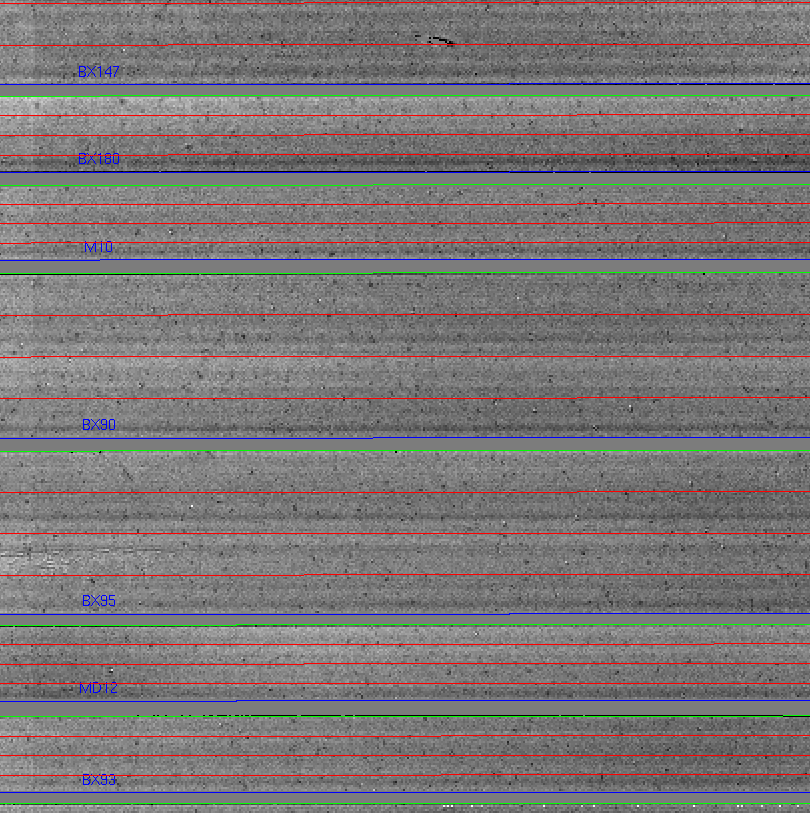
slit-edges\_J.npy File containing the slit edge information

slit-edges\_J.reg DS9 regions file that may be overlayed to show the locations of the slits.

At the end, check the output in ds9. For example:

> ds9 pixelflat\_2d\_H.fits -region slit-edges\_H.reg

The regions file overlayed on the pixelflat image should look something like:



The green lines must trace the edge of the slit. If they don’t, then the flat step failed. All values should be around 1.0. There are some big features in the detector that you will become familiar with over time.

## K-band flats

At K-band, the dome is hot enough that light is detected at the longest wavelengths at a level of a few hundred counts. Little to no light is seen at the shortest wavelengths. The light from the dome is not entering MOSFIRE at the same angles that the light from the spot illuminated on the dome by the dome lights. Some observers may wish to correct for this difference by subtracting the thermal flat emission from the dome flat emission before normalizing the flats. To complete this flat subtraction, you use the optional keyword lampsofflist in the flat process as seen in the command below:

Flats.handle\_flats('Flat.txt', maskname, band, flatops,lampOffList='FlatThermal.txt')

If thermal flats were included in your calibration sequence (default behavior for K-band), then the FlatThermal.txt file should be populated with a list of thermal flats. Use FlatThermal.txt as the list or modify it as you see necessary.

The outputs from the flat process will include two additional files.

combflat\_lamps\_off\_2d\_K.fits

combflat\_lamps\_on\_2d\_K.fits

and now the combflat\_2d\_K.fits being the difference between the two files.

# Wavelength Calibration – Y, J, and H bands:

In the shorter wavebands, when using the recommended exposure times, the wavelength calibration is performed on night sky lines. The mospy Wavelength module is responsbile for these operations. See the example driver file in section 7.

## Combine files

First step is to produce a file with which you will train your wavelength solution. Since we’re using night sky lines for training, the approach is to combine individual science exposures. This is performed by the python Wavelength.imcombine routine. For a lot of users, this will look something like in the Driver.py file:

Wavelength.imcombine(obsfiles, maskname, band, waveops)

The first parameter is obsfiles which is a python string array indicating the list of files in the offset positions. Note that obsfiles has defaults of “Offset\_1.5.txt” and “Offset\_-1.5.txt” and may need to be updated as described in section 6.

Suppose you want to exclude a file for reasons such as weather or telescope fault, simply remove the offending file from the appropriate Offset\_\*.txt. Likewise, you are welcome to add files in as you like, such as observations from the previous night.

Outputs of this step are:

|  |  |
| --- | --- |
| Filename | Contains |
| wave\_stack\_[band]\_[range].fits | A median-combined image of the files to be used for the wavelength solution. |

## Interactive wavelength fitting

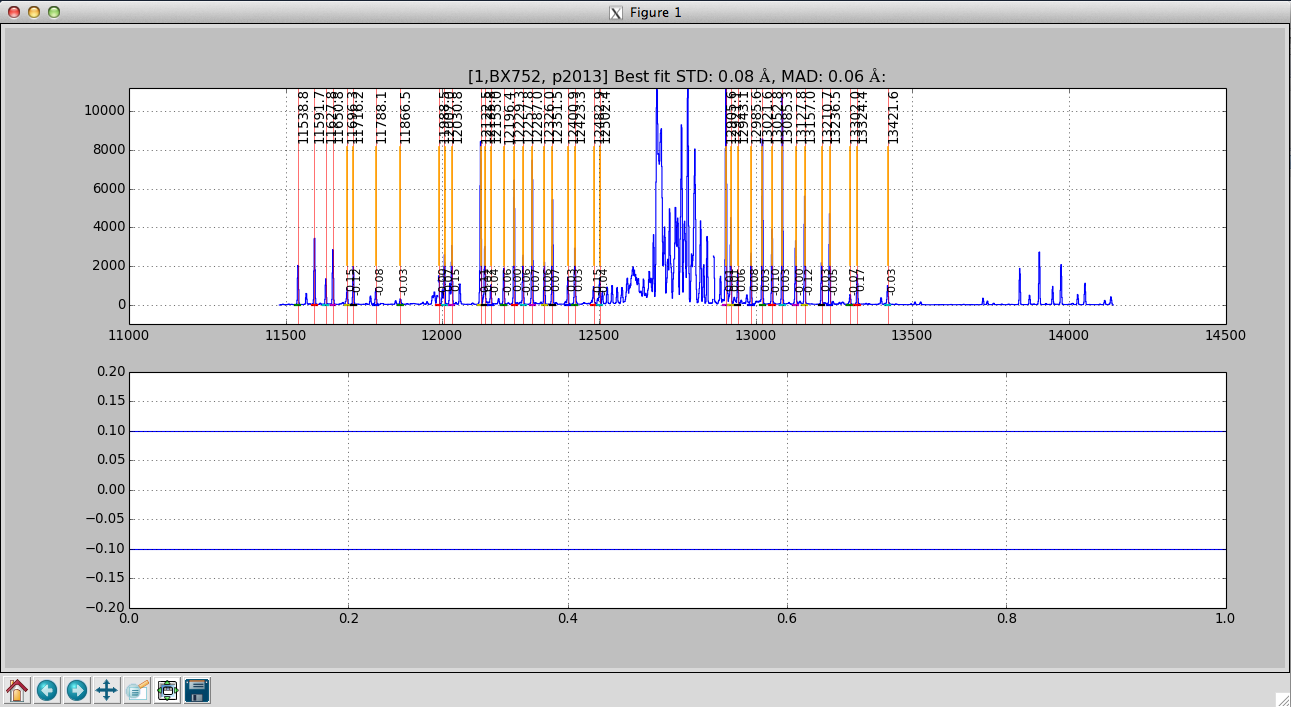
The next step is to use the wave\_stack\_\*.fits file and determine an initial wavelength solution for each slit. During this process, we interactively fit the lines using a gui that displays. To initiate this process, uncomment the line in the Driver.py file:

#Wavelength.fit\_lambda(maskname, band, obsfiles, obsfiles, waveops)

And then re-execute the driver file:

mospy Driver.py

when you run this step, a GUI window appears. The

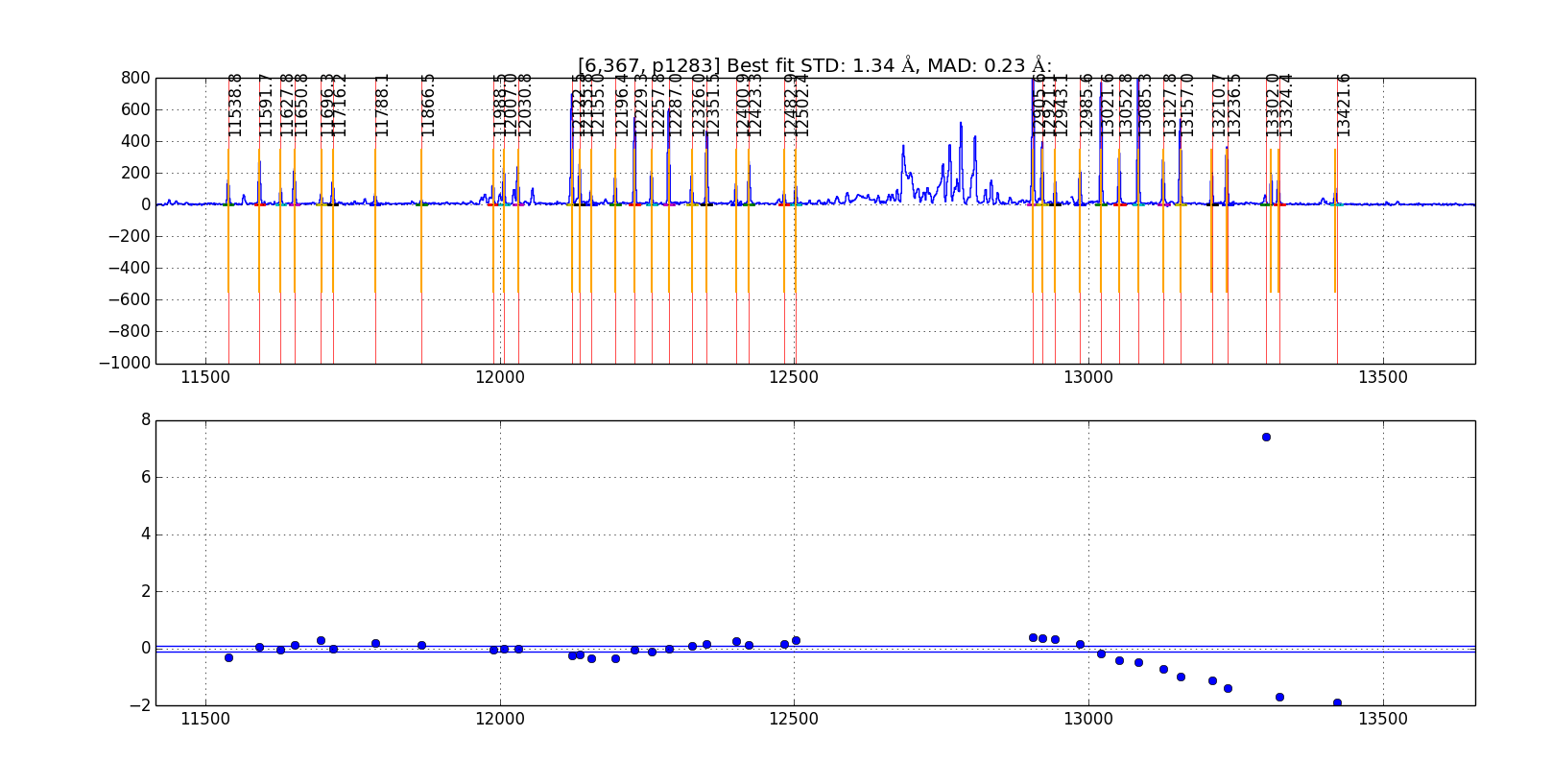


Ignore these buttons

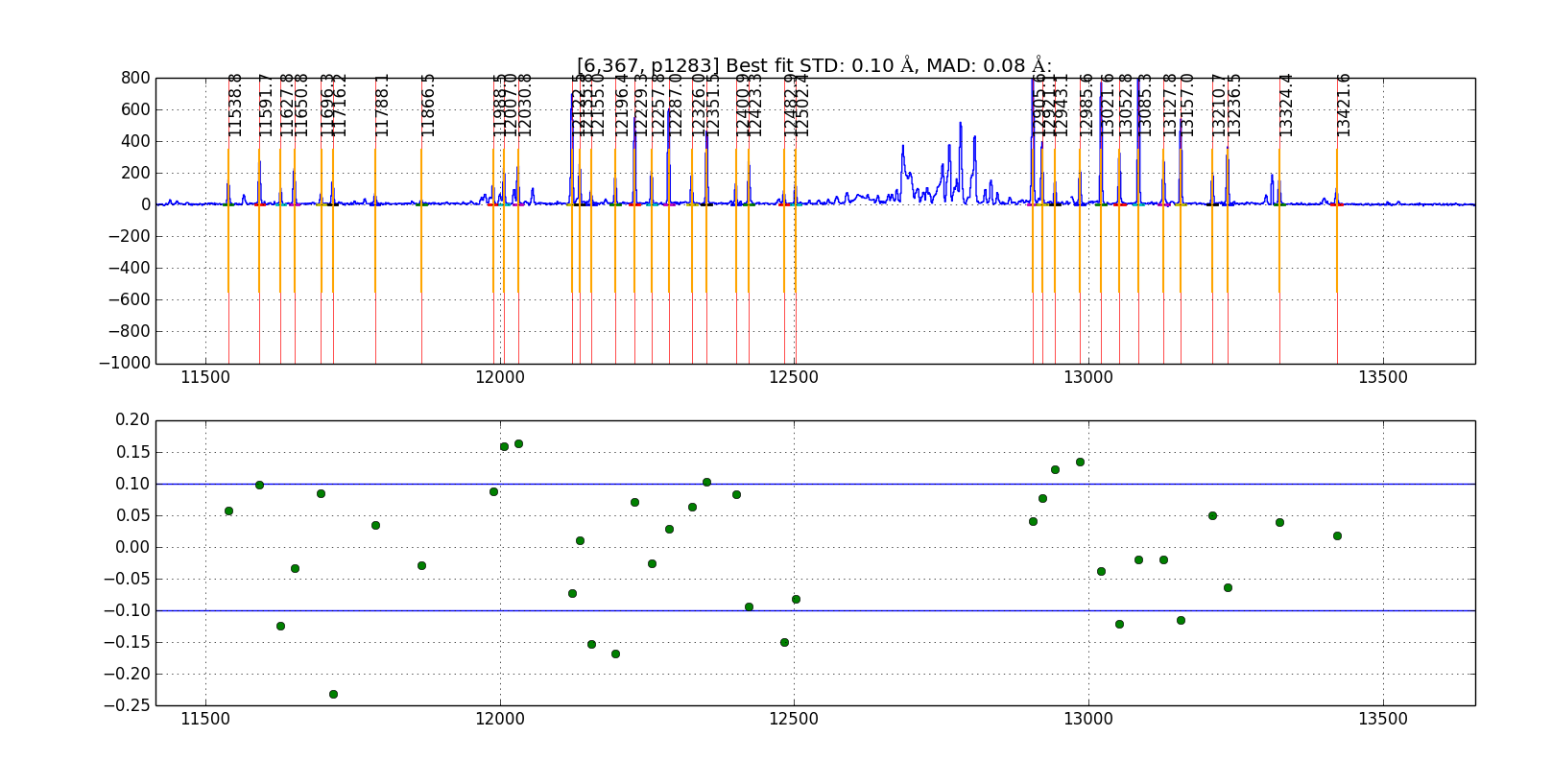
Spectra

Residual wavelength

*The interactive wavelength solving window. This is a J-band night sky spectrum.*



*The interactive wavelength solving window showing an initial fit. This is a J-band night sky spectrum and one of the night sky lines on the right hand side is clearly a poor fit compared to the rest of the identified lines.*



*The interactive wavelength solving window showing a good fit with the initial poor line removed from the calculation.*

Plotted in the gui will be a sky line spectrum and vertical lines denoting positions and wavelengths of the sky lines. Your goal is to help the pipeline by identifying the night sky lines in the center of each slit. Once you come up with a good solution in the center, the pipeline will propagate it spatially along the slit. In the gui, Press ? to see a list of commands in the console window. The list of commands available on the GUI are:

* **c** - to center on the nearest peak (first thing to do to shift the initial wavelength guess)
* **c** - Center the nearest line at the cursor position
* **\** - Fit fit the data
* **f** – Alternate way to fit the data, equivalent to \ but may cause the spectrum to become full screen.
* **k** – Toggles k-sigma clipping on and off. Also performs a new fit.
* **b** - Turns on bypass mode. From now on, the interactive window is not displayed and the fit continues automatically.
* **d** - Delete a point (remove the wackadoos)
* **n** - proceed to the Next object
* **p** - return to back to the Previous object
* **r** - Reset the current slit (try this if the plot looks strange)
* **z** - Zoom at cursor position
* **x** - Unzoom: full screen
* **s** - Save figure to disk
* **h** - Help
* **q** - Quit and save results

Below is a rough procedure for completing the interactive fitting process. The steps you need to take are as follows.

* First, check to see if the orange lines match up with obvious night sky lines. If not the expected position does not match the actually position of the line do the following:
  + Place your cursor over a line
  + Press the “c’ button that will shift the predicted position to the observed line.
* Press “f” to fit. An initial fit is done automatically (starting with version 2015A) A Chebyshev polynomial, *f*, such that *f*(pixel #) returns the wavelength in Angstroms.
* You can chose to use k-sigma clipping to avoid the manual operation of removing bad lines with the “k” key.
* Press “x” to unzoom to the full size region
* Assess the fit:
  + If a line is poorly fit and should be removed
    - Move the cursor to the line
    - Press “d” to delete the line from the fit
  + For good fits, the residual points turn green.
* When the satisfied with the fit, press “n” to move to the next object.
* If you want to disable the interactive fit and switch to the automatic fit, press “b” (bypass)
* Repeat the process above until you see the red Done! text in the center of your screen.
* Press “q” to quit the interactive gui and move to the next step.

The prompt should return following the fitting process. The outputs from this process are:

|  |  |
| --- | --- |
| Filename | Contains |

barset.npy bar positions for each slit are specified

lambda\_center\_coeffs\_wave\_stack\_band\_filenames.npy

The coefficients of the fit and positions of the measured lines.

## Wavelength fitting for the entire slit

The next step in the wavelength fitting process is to propogate the solution spatially along each slit. To complete this process we uncomment the line in the Driver.py file:

#Wavelength.fit\_lambda(maskname, band, obsfiles, obsfiles, waveops)

This is one of the longer running processes and the output should look something like:

....

resid ang S09 @ p 978: 0.10 rms 0.07 mad [shift-22]

resid ang S09 @ p 979: 0.09 rms 0.06 mad [shift-22]

resid ang S09 @ p 980: 0.10 rms 0.06 mad [shift-22]

resid ang S09 @ p 981: 0.09 rms 0.06 mad [shift-22]

resid ang S09 @ p 982: 0.08 rms 0.05 mad [shift-22]

resid ang S09 @ p 983: 0.08 rms 0.04 mad [shift-22]

….

The prompt should return following the fitting process. The outputs from this process are:

|  |  |
| --- | --- |
| Filename | Contains |

lambda\_coeffs\_wave\_stack\_J\_m130114\_0443-0445.npy

coefficients of the fit for each row within the slit.

## Apply the wavelength solution.

The last step in the wavelength fitting process is to apply the solution and create maps of the wavelength for the data set. To complete this process we uncomment the line in the Driver.py file:

#Wavelength.apply\_lambda\_simple(maskname, band, obsfiles, waveops)

The prompt should return following the fitting process. The outputs from this process are:

|  |  |
| --- | --- |
| Filename | Contains |

lambda\_solution\_wave\_stack\_J\_m130114\_0443-0445.fits – contains a map of the wavelength for each pixel in the spectra

sigs\_solution\_wave\_stack\_J\_m130114\_0443-0445.fits –contains the uncertainty in the measured wavelength position for each pixel in the spectra

rectified\_wave\_stack\_J\_m130114\_0443-0445.fits – contains the spatially and wavelength rectified resampled sky emission. A column in the image contains all pixels at the same wavelength.

# Wavelength Calibration – K band: merging arcs and skys:

The night sky lines at the red end of the K-band are too faint to achieve small-fraction of a pixel RMS wavelength calibration. You will have to observe a Neon and Argon arc lamps during your afternoon calibrations. By default, the calibration script at the observatory is setup to acquire both the Ne and Argon arcs.

Because the beams emminating from the arclamp do not follow the same path as the beams coming from the sky, there will be a slight difference between the two solutions. For the afformentioned beam matching reason, the most accurate solution is the night sky lines. Thus, the code has to be clever about merging the two solutions.

The following subsections describe the additional steps that are necessary to process the arcline data and combine the arcs and night sky line wavelength solutions.

## Combine the arc line spectra

Just like the step in section 8.1 where you combined the science frames to create nightsky line spectra, we first need to combine the arcline data. The arcs are typically three files and you should see them listed in the Ne.txt and Ar.txt file lists in your K band sub directory. To combine the images simply uncomment and run:

Wavelength.imcombine('Ne.txt', maskname, band, waveops)

Wavelength.imcombine('Ar.txt', maskname, band, waveops)

## Identify arc lines using night sky solution

Instead of having to interactively determine the wavelenth solution for the arcs like we did in section 8.2 for the night sky lines, we are going to use the solutions for the night sky lines as a first approximation for the arcs. This may usually be done because the arcs differ from the night sky lines by a fractions of pixels. You are welcome to interactively solve the neon lamp solution with the Wavelength.fit\_lambda\_interactively routine; however, the need to run the interactive solution method should be rare.

To apply the solution from the night sky lines to the arcs center slit position, uncomment and run the following lines.

Wavelength.apply\_interactive(maskname, band, waveops, apply=obsfiles, to='Ne.txt', neon=True)

Wavelength.apply\_interactive(maskname, band, waveops, apply=obsfiles, to='Ar.txt', argon=True)

This step, when run will produce output like:

slitno 1 STD: 0.16 MAD: 0.06

slitno 2 STD: 0.03 MAD: 0.02

slitno 3 STD: 0.04 MAD: 0.04

slitno 4 STD: 0.05 MAD: 0.01

For each slit, a new solution is generated for the neon line. The output mimics that described previously where STD is the standard deviation and MAD is the median absolute deviation in angstroms.

## Wavelength fitting for the entire slit using arcs

## Th next step in the wavelength fitting process is to propogate the arc solution spatially along each slit. Again this is the same process essentially as the fit for the night sky lines. This moves along each row for the slit to determine a wavelenth solution. The output files are comperable to those in step 8.3

Wavelength.fit\_lambda(maskname, band, 'Ne.txt', 'Ne.txt', waveops, wavenames2='Ar.txt')

You will note that the Ar.txt file is listed as an optional argument. If you choose not to use the Argon lamps, then you may simply remove the optional wavenames2 and execute this using only the Ne arcs.

Again, this process takes some time to complete.

## Merge the arc and sky lists

In this portion of the procedure, we merge the two lists. These commands may not be run individually. Instead any command containing the variable LROI needs to be run in one mospy driver file session in order to pass the LROI variable. In this section we determin the offsets between the region of overlap between the nightskylines and the arclines. A plot of that region is displayed. To move on you will have to close the plot.

To execute this step you will need to uncomment the following lines in the driver file.

LROI = [[21000, 22800]] \* 1

LROIs = Wavelength.check\_wavelength\_roi(maskname, band, obsfiles, 'Ne.txt', LROI, waveops)

Wavelength.apply\_lambda\_simple(maskname, band, 'Ne.txt', waveops)

Wavelength.apply\_lambda\_sky\_and\_arc(maskname, band, obsfiles, 'Ne.txt', LROIs, waveops, neon=True)

The merged output solution will have a filename that looks like:

merged\_lambda\_coeffs\_wave\_stack\_K\_m130114\_0451-0453\_and\_wave\_stack\_K\_m140508\_0197-0199.npy

merged\_lambda\_solution\_wave\_stack\_K\_m130114\_0451-0453\_and\_wave\_stack\_K\_m140508\_0197-0199.fits

merged\_rectified\_wave\_stack\_K\_m130114\_0451-0453\_and\_wave\_stack\_K\_m140508\_0197-0199.fits.gz

The ouput files have the same format as those in section 8.4 and will need to be used as inputs to the Background and Rectify section below.

# Background Subtraction

This DRP assumes that targets are nodded along the slit with integration times as described on the instrument web page. The integration times described were selected such that the shot-noise in the region between night sky lines is over 5x larger than the read noise of a 16-fowler sample. For MOSFIRE, we define this as background limited.

Despite MOSFIRE’s (unprescedented) f/2.0 camera, the desired integration time for background-limited operation is longer than the time for the atmosphere to vary by several percent. As a result, a further background subtraction step is required to remove the residual features. The step is performed by a function called background\_subtract\_helper() and follows the notation and procedure outlined in Kasen (2003; PASP 115). For most users, you’ll want to use the standard Driver file and not worry about the details.

In the Driver.py file you want to uncomment the following:

Background.handle\_background(obsfiles,

'lambda\_solution\_wave\_stack\_J\_m130114\_0443-0445.fits',

maskname, band, waveops)

The lambda\_solution\_wave\_stack file needs to be updated in your driver file. If reducing Kband, be sure to use the merged wave\_stack solution. It is one of the outputs from the last wavelength step (see section 8).

In this step:

* Apply the flat field corrections
* A position files are combined (Offset\_\*.txt)
* B postion files are combined (Offset\_-\*.txt)
* Subtract A-B
* Correct for small differences in the background sky emission

## Output Files

The background subtraction step produces the following files.

As usual elements in [brackets] are replaced with the value for that mask.

Table 1 - Background subtraction step unrectified outputs. These files are used later in the DRP.

|  |  |
| --- | --- |
| Filename | Content (units) |
| eps\_Offset\_[###].txt.fits | Average signal in the ### stack () |
| var\_Offset\_[###].txt.fits | Total variance in each pixel of above file () |
| itimes\_Offset\_[###].txt.fits | Total exposure time in each pixel of above files () |
| sub\_[maskname] \_[bandname]\_[plan].fits | Difference (but non background subtracted) file () |
| bsub\_[maskname]\_{ bandname]\_[plan].fits | Background subtracted signal () |
| bmod\_[maskname]\_{ bandname]\_[plan].fits | Background model signal () |
| var\_[maskname]\_{ bandname]\_[plan].fits | Total variance |
| itime\_[maskname]\_{ bandname]\_[plan].fits | Average integration time |

There is redundant information in the above set of files. For instance:

sub\_Mask\_K\_A-B.fits = eps\_Offset\_1.5.txt.fits – eps\_Offset\_-1.5.txt.fits

var\_Mask\_K\_A-B.fits = var\_Offset\_1.5.txt.fits + var\_Offset\_1.5.txt.fits

itime\_Mask\_K\_A-B.fits = mean(itime\_Offset\_1.5.txt.fits, itime\_Offset\_1.5.t.xt.fits)

If you want to drill further into how these are constructed, examine the Background.py imcombine and handle\_background functions.

Recitified outputs are also computed as tabulated in Table 2.

Table 2 - Background subtracted and rectified outputs. The following files are not used in the later steps of the DRP, they are provided for convenience.

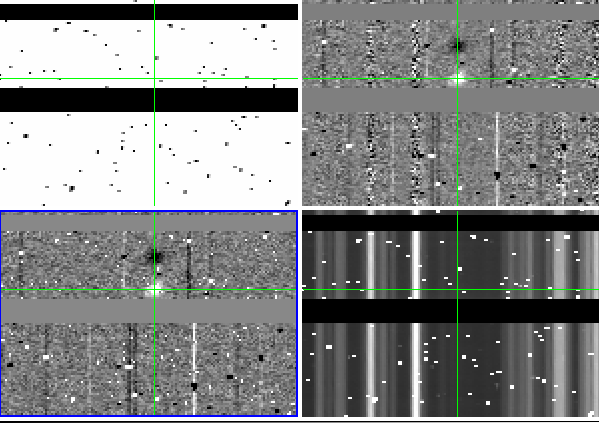
|  |  |
| --- | --- |
| Filename | Content (units) |
| [maskname]\_rectified\_[bandname]\_[plan].fits | Signal () |
| [maskname]\_rectified\_itime\_[bandname]\_[plan].fits | Integration time |
| [maskname]\_rectified\_var\_[bandname]\_[plan].fits | Variance |
| [maskname]\_rectified\_sn\_[bandname]\_[plan].fits | Signal to noise () |

Note that signal to noise is computed as follows:

yes, we violate the first normal form for convenience. Also note that the STD is computed assuming the detector has a read noise of Detector.RN (documented in the MOSFIRE Pre Ship Review as 21 electron) per fowler sample. Thus, the final STD is

assuming the gain in Detector.gain. Note that there is no shot noise from dark current, which was measured to be negligible at pre-ship review.

An example of what the output looks like is here:



*Image showing the itime, bsub, and rectified wavelength images. The green crosses are marking the location of the same pixel in each image.*

# Rectify

The last step in the reduction process is to combine the wavelength solution with the backgroun subtracted images and then shift and combine the nod positions. If reducing Kband, be sure to use the merged wave\_stack solution. To do this we uncomment the following lines in the Driver.py file:

redfiles = ["eps\_" + file + ".fits" for file in obsfiles]

Rectify.handle\_rectification(maskname, redfiles,

'lambda\_solution\_wave\_stack\_J\_m130114\_0443-0445.fits',

band,

waveops)

The output from this procedure produces four files for every slit.

|  |  |
| --- | --- |
| Filename | Content (units) |
| [maskname]\_[band]\_[object]\_eps.fits | Signal () |
| [maskname]\_[band]\_[object]\_itime.fits | Integration time |
| [maskname]\_[band]\_[object]\_sig.fits | Variance ? |
| [maskname]\_[band]\_{object}\_snrs.fits | Signal to noise () |

There is also four images without the “object” in the name. These four files contain the composit spectra with all spectra aligned spectrally and both beams combined. In the \*eps.fits files, you will see two negative traces and one positive trace. For a two position nod, the eps files is (A-B) +((B-A)shifted).

When extracting the emission from an object or measuring the position of an emission line, you should be accessing the \*eps.fits files with the wavelength solution written into the WCS information.

# Long2pos Reductions

A special driver is provided for long2pos reductions. The driver can also be generated automatically.

As s reminder, these observations are taken using a script which is run either from the command line (acq\_long2pos) or via the background menu. The script produces different results depending on whether the long2pos mask was setup in science mode (only narrow slits) or in alignment mode (narrow and wide slits).

In the general case of a combination of narrow and wide slits, each run of the script generates 6 images, 3 for each of the two slits. We will refer to the two positions as position A and position C (position B is the intial position used only for alignment).

Depending on when your data was generated, you might find different Offset files in your directory. Files generated before June 10, 2015 use a different set of YOFFSET keywords than files generated after that date. Unfortunately, the set of keywords generated before June 10, 2015 is not compatible with the pipeline and must be updated: for this we provide a special set of instructions as part of as the driver file to automatically update the keywords.

For files generated before June 10, 2015, you will find 6 Offset files, named Offset\_XXX\_object\_name\_PosY.txt, where XXX can be -21, -14,-7, 7, 14 and 21, the object name is taken from the object keyword, and Y can be either A or C. Similar names are produced if the observations has the correct keywords, but in that case XXX will be one of -7, 0, or 7.

It is important to notice that the reduction described here is based on the assumption that proper arc lamps are obtained in the afternoon. Specifically, either a Ne or Ar calibration must be obtained with the long2pos mask executed in science mode, and not in alignment mode. In science mode the wide part of the slits is not present. If the slit was executed in alignment mode, the wide part of the slits would prevent a wavelength calibration.

Note that this also means that if you took your science data at night in long2pos\_specphot mode, the mask name of your science file might be long2pos\_specphot, rather than long2pos, and the arcs and flats might end up in the wrong subdirectory when the files are processed via mospy handle. In this case it will be necessary to copy Ar.txt, Ne.txt and Flat\*.txt from the directory long2pos to your long2pos\_specphot directory.

Let’s now look at the driver file.

The declaration “longslit =” is used to define the pixel boundaries of the long2pos observations. In general, it is correct and should not be changed. It might need to be updated in the future is a new long2pos mask is used. Note that it is important to specify ‘mode’=’long2pos’

The following section describes the rather long list of Offset files that we will use for the reduction.

For observations obtained before June 10, 2015, this section might look like this:

obsfiles\_posC\_narrow = ['Offset\_-21\_HIP85871\_PosC.txt', 'Offset\_-7\_HIP85871\_PosC.txt']

targetCnarrow = "HIP85871\_posC\_narrow"

obsfiles\_posA\_narrow = ['Offset\_7\_HIP85871\_PosA.txt', 'Offset\_21\_HIP85871\_PosA.txt']

targetAnarrow = "HIP85871\_posA\_narrow"

obsfiles\_posC\_wide = ['Offset\_-14\_HIP85871\_PosC.txt','Offset\_-7\_HIP85871\_PosC.txt']

targetCwide = "HIP85871\_posC\_wide"

obsfiles\_posA\_wide = ['Offset\_14\_HIP85871\_PosA.txt','Offset\_21\_HIP85871\_PosA.txt']

targetAwide = "HIP85871\_posA\_wide"

Files -21\_PosC and -7\_PosC are the A and B positions for the C pointing, files 7 and 21 are the A and B positions for the A pointing. For the wise slits, file 7\_PosC is used as a sky (B) for the -14\_PosC position, and file 21\_PosA is used as a sky for the 14\_PosA position. The target keywords must also be specified to avoid accidental overwrite of intermediate files.

For files obtained after June 10, 2015, the same section would look like this:

obsfiles\_posC\_narrow = ['Offset\_7\_FS134\_posC.txt','Offset\_-7\_FS134\_PosC.txt']

targetCnarrow = "FS134\_posC\_narrow"

obsfiles\_posA\_narrow = ['Offset\_7\_FS134\_posA.txt','Offset\_-7\_FS134\_PosA.txt']

targetAnarrow = "FS134\_posA\_narrow"

obsfiles\_posC\_wide = ['Offset\_0\_FS134\_posC.txt','Offset\_-7\_FS134\_PosC.txt']

targetCwide = "FS134\_posC\_wide"

obsfiles\_posA\_wide = ['Offset\_0\_FS134\_posA.txt','Offset\_-7\_FS134\_PosA.txt']

targetAwide = "FS134\_posA\_wide"

The first step is to produce a flat field.

Flats.handle\_flats('Flat.txt', maskname, band, flatops, longslit = longslit)

or

Flats.handle\_flats('Flat.txt', maskname, band, flatops,lampOffList='FlatThermal.txt', longslit=longslit)

Using argon (or neon) lines, we can now produce a wavelength calibration.

Wavelength.imcombine(argon, maskname, band, waveops)

Wavelength.fit\_lambda\_interactively(maskname, band, argon, waveops, longslit=longslit, argon=True)

Wavelength.fit\_lambda(maskname, band, argon, argon, waveops, longslit=longslit)

Wavelength.apply\_lambda\_simple(maskname, band, argon, waveops, longslit=longslit, smooth=True)

While using the interactive fitting, note that there are two slits to fit.

The next section of the driver reduces the narrow slits. The optional line

IO.fix\_long2pos\_headers(obsfiles)

is ONLY necessary if your observations were taken before June 10, 2015. It is safe to leave this line on for a second run: the script will not modify the same files twice.

Rememeber to update the lambda\_solution\_wave\_stack file: you can update this in the variable wavelength\_file, which will be used by the following instructions.

The driver contains instructions on how to perform background subtraction and finally rectification, in a similar way as for a normal mask.

The resulting files are the same as in the standard reduction, but the main results are contained in:

{object}\_posA\_narrow\_{filter}\_eps.fits

and

{object}\_posC\_narrow\_{filter}\_eps.fits

For the wide slits, since there is no AB pattern, we use the sky provided by one of the observations in the narrow slits, and we do not perform the final rectification.

In this case the final science results are contained in:

bsub\_{object}\_posC\_wide\_{filter}\_A-B.fits

and

bsub\_{object}\_posA\_wide\_{filter}\_A-B.fits

# Longslit Reductions

The longslit reductions require transferring the Longslit\_Driver.py file into the reduction directory. A few key parameters have to be adjusted in Longslit\_Driver.py to help the pipeline figure out where to extract the longslit from.

1. cd /path/to/LONGSLIT/
2. cp ~/mosdrp/drivers/Longslit\_driver.py .
3. Check all the .txt files to make sure your observations are included. You may have to merge files from various LONGSLIT\* directories. This happens when your observations use a shorter longslit than the calibrations.
4. Note that mospy handle generates separate offset files for each of your targets, using the target name, but does NOT separate repeated observations of the same targets at different times of the night.
5. edit Driver\_Longslit.py
   1. Examine a longslit image (see figure below) and adjust ‘yrange’: [709, 1350] to the vertical range covered by the slit
   2. From the same examined longslit, select ‘row\_position’ so that it is uncontaminated by the spectrum. See Figure 1.
   3. make sure that ‘mode’:’longslit’ is specified in the longslit variable
   4. The result should look like Figure 2.
6. For each step in a section, uncomment the necessary line and run mosdrp on the Driver file. Once the apply\_lambda\_simple step is complete, fill in the ‘lambda\_solution\_wave\_stack\_...’ line with the correct wave stack file.

You now have two options based on the results. If the night sky lines are not bright enough to identify in the interactive step you should use arclamps. In the following instructions, replace wavefiles with ‘Ne.txt’ or ‘Ar.txt’ and specify neon=True or argon=True.

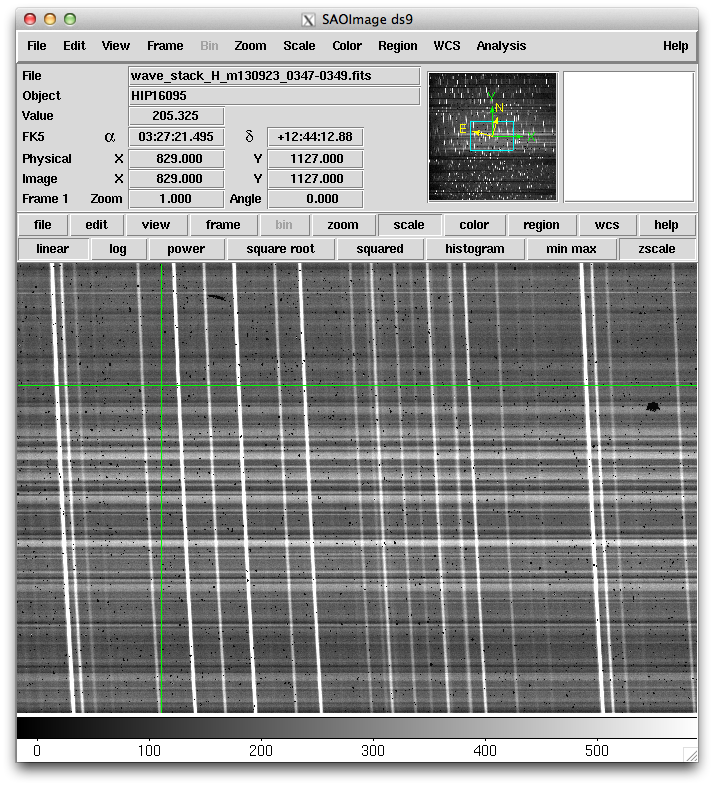


Figure 1- An example of an uncontaminated row (#1127) in the longslit.

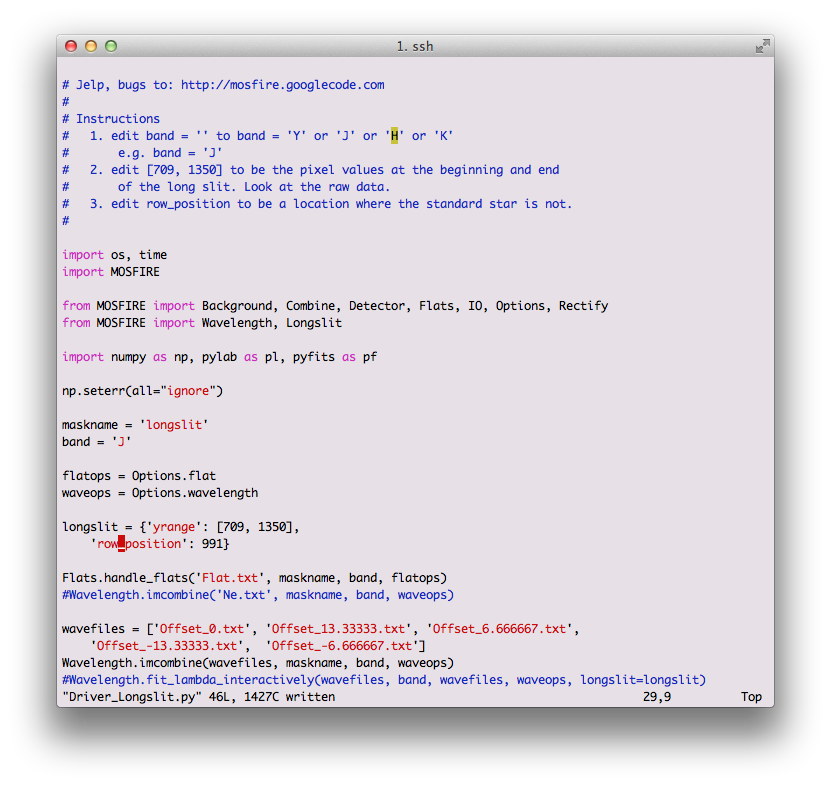


Figure 2- Example of a modified Driver\_Longslit.py. Notice that pixel 991 is selected as the row to perform the initial wavelength solution on. In Figure 2, this is the equivalent of 1127.

# A word about header comments

Files produced by the DRP have a series of information in the FITS header that helps users determine the pedigree of files involved in the reduction. Since many files go into reductions, FITS headers are enormous and some documentation about them is useful.

The derived product FITS headers are organized as follows. The header of the first science file in the ‘A’ frame goes directly into the header. As the rest of the ‘A’ frames go into the header, the new keyword is checked against the current header. If the value of the keyword is different, a new keyword is added with the key postpended by \_img### where ### is the file number. A special keyword called imfno### is created showing the full path to the file in the data reduction set. An example is shown below:

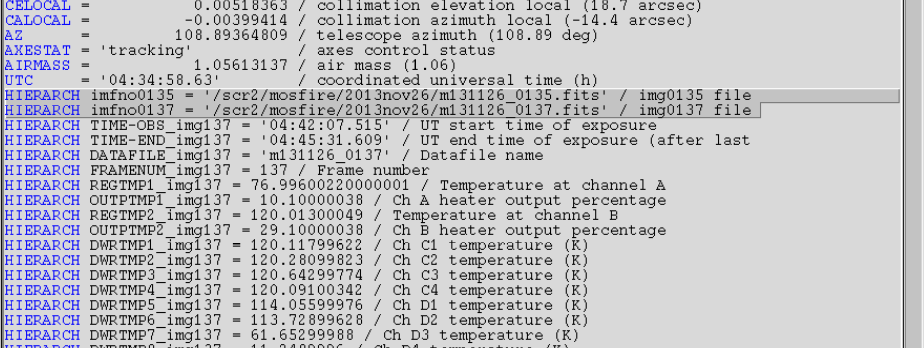
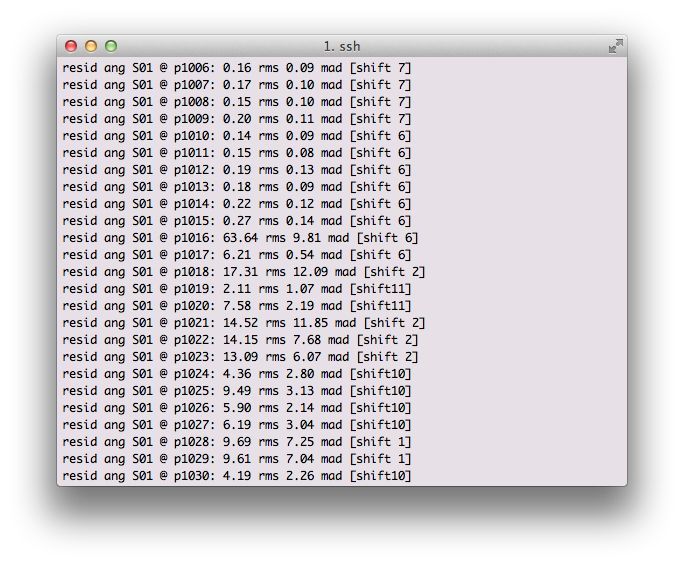


Figure 3- ds9 output of the FITS header. Note that the first "A" frame file is located in /scr2/mosfire/2013nov26/m131126\_0135.fits. The second file (#137) has a similar path. The keywords which follow from file #137 have different values than those in file #135 and are thus named KEY\_img###.

# Some Hints

## Pay attention to the wavelength fitting output:

The output above shows that up to pixel 1015 the RMS was 0.27 Angstrom level, and then dramatically jumped to 60 angstrom. Look at the image and examine pixel 1016, figure out what happened. You may have to adjust your input files or remove a file from the set.

## Look at rectified\_wave\_stack files

Look at rectified\_wave\_stack\* files and make sure the night sky lines are vertical on the detector.