Table of Contents

The manual is also available in pdf form: MOSFIRE_DRP_Manual.pdf

Note: pdf version of the manual is generated from the individual markdown files for each section by running the `generate_pdf` script in the `docs/` subdirectory of this repository.

Change Log

Changes in Version 2018

New features

- Python3 compatibility!
- Improved installation procedure uses anaconda cloud
- The 1D spectral extraction code now provides the user the ability to set the default aperture width.
- Revamped documentation using mkdocs including the addition of an examples section.

Note: This release of the DRP is only tested and supported on python 3.6+. If you must use python 2, use the 2016 release of the DRP. We encourage all users to use python 3 and we make the install of python 3 easy using anaconda python and the anaconda cloud (see the installation instructions).

Improvements and bug fixes

- Fixed bug where 1D extraction apertures could not be interactively positioned.
- Fixed astropy deprecation warning related to clobber option
- Fixed bug where slit nods with an odd number of positions would fail
- Fixed numpy deprecation related to indexing arrays with non-integers
- Fixed header info bug which would cause DS9 coordinates to display incorrectly

Changes in Version 2016

Important Note: The Ureka package has been deprecated as of April 26, 2016. As a result, the MOSFIRE pipeline has migrated to a version which is not dependent on IRAF/PyRAF, but only on python packages. It should work with any python install which provides the required packages and versions.

New features

- DRP is no longer dependent on IRAF/PyRAF
  - The use of IRAF's `geoxytran`, `imcombine`, and `imarith` tasks have been replaced with python equivalents.
  - The DRP should now work with any python install which has the required python packages
- Improved slit tracing using a better thresholding algorithm
- An updated (and now web based) instruction manual
- The DRP now performs optimal spectral extraction Horne 1986 and outputs a 1D spectrum. Please note that this is intended as a quick look tool, not for final science use.
- The `handle` step now writes `filelist.txt` which contains a list of all the files processed by `handle` instead of printing that output to the screen. The file also contains messages for files not categorized for processing explaining why. In addition, `handle` now no longer writes list files with no content. This is intended to make it easier to quickly see what files are available for reduction.

Improvements and bug fixes

- Changed dependence on `pylab` to `matplotlib.pyplot`
- Uses `astropy.io.fits` instead of `pyfits` when available
- Adjust log messages to send more to DEBUG instead of INFO. Leads to less clutter in messages visible to user.

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 https://github.com/Keck-

DataReductionPipelines/MosfireDRP. Please use this website to track issues and and submit requests.

Installation

Requirements

The pipeline requires the following python modules:

- numpy
- astropy
- ccdproc
- scipy

Installing Python

Using Anaconda Cloud and Conda Environments

Install Anaconda as per the instructions on the Anaconda web site.

Now we will create a conda environment specifically for the MOSFIRE DRP. Rather than specify everything on the command line, we will get the specification for the environment from the Anaconda Cloud service. There are two specifications, one for linux (tested on a CentOS 7 system) and one for macOS (tested on macOS 10.12.6). Get the one appropriate for your system using one of the commands below:

```
conda env create KeckObservatory/mospy_2018_linux
```

or

```
conda env create KeckObservatory/mospy_2018_macos
```

Now we will invoke that environment:

```
source activate mospy_2018_linux
```

or

```
source activate mospy_2018_macos
```

Now we will install the DRP itself. From now on, if you want to run the DRP, first invoke the appropriate environment using `source activate mospy_2018_linux` or `source activate mospy_2018_macos`.

Download and Install the DRP

Download the zip file of the released version from GitHub.

Move the zip file to a location on your computer where you want the source code
to reside, then unzip the file:

`unzip MosfireDRP-2018release.zip`

Change in to the resulting `MosfireDRP-2018release/` directory:

`cd MosfireDRP-2018release`

Run the install program:

`python setup.py install`

The executable `mospy` should now be in your path. If you used the
Anaconda based install, it will be in the Anaconda bin directory (e.g.
`~/anaconda/envs/mospy_2018_macos/bin/mospy`).

Alternate Methods of Installing Python

Note, these are no longer the recommended methods of installing the DRP as
they do not gauranttee that the various package versions are compatible with
the DRP.

Using the Anaconda Distribution

Install Anaconda as per the instructions on the Anaconda web site. The pipeline
currently (2016 release) only runs on python 2.7, so download and install that
version, not the python 3.x version.

To generate an environment similar to the one in the recommended anaconda
cloud based install, you can use the following command:

`conda create --no-default-packages -c astropy -n mospy_2018_macos python=3.6.3 astropy=2.0.3`

You should now have all the requirements to run the MOSFIRE DRP. This
should work on any anaconda install, even if the pre-packaged linux and macOS
environments are incompatible with your machine.

Using Other Python Install Methods

The DRP support group recommends the anaconda python install and has tested
the DRP using that installer, but if an appropriate version of python is installed
via some other package manager (e.g. apt-get, brew, yum, etc.), then you should
be able to install the python package dependencies using either that package

manager (if they are available via that package manager) or using `pip`. For example:

```
pip install numpy
pip install astropy
pip install ccdproc
```

Retrieve

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 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). 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: Grab the data from: https://www2.keck.hawaii.edu/realpublic/inst/mosfire/DRP_Test_Case_Hband.zip.

Move the test case zip file to a directory of your choice (we will assume `~/Data` for the examples in this manual) and unzip the DRP test case file:

```
unzip DRP_Test_Case_Hband.zip
```

This will create a `DRP_Test_Case_Hband` subdirectory under your current directory which will contain the raw data which you can use to follow along in subsequent steps of the DRP manual.

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 (`reduced` in our example).

```
mkdir reduced
cd reduced
mospy handle /home/[yourhomedir]/Data/DRP_Test_Case_Hband/2012sep10/*fits
```

Please use the full path to the raw data when invoking `mospy handle`.

A lot of data summarizing the observations is output. This includes a table of the observations:

```
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]/`

Containing

- 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 separate reductions of the same mask across multiple dates. Below is a screen shot showing the example output from an Offset*.txt file.

Figure 1: Screenshot

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.

AutoDriver

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 ~/Data/reducedMOSFIRE_DRP_MASK/2012sep10/H`
2. Copy the appropriate driver file: `cp ~/MosfireDRP-master/drivers/Driver.py .` NOTE: If you are observing a K band mask you'll want to copy the `K_driver.py` file over.
3. Edit driver.py (see bold text in driver file example)
   - Update maskname
   - Update band to be Y,J,H
   - 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:
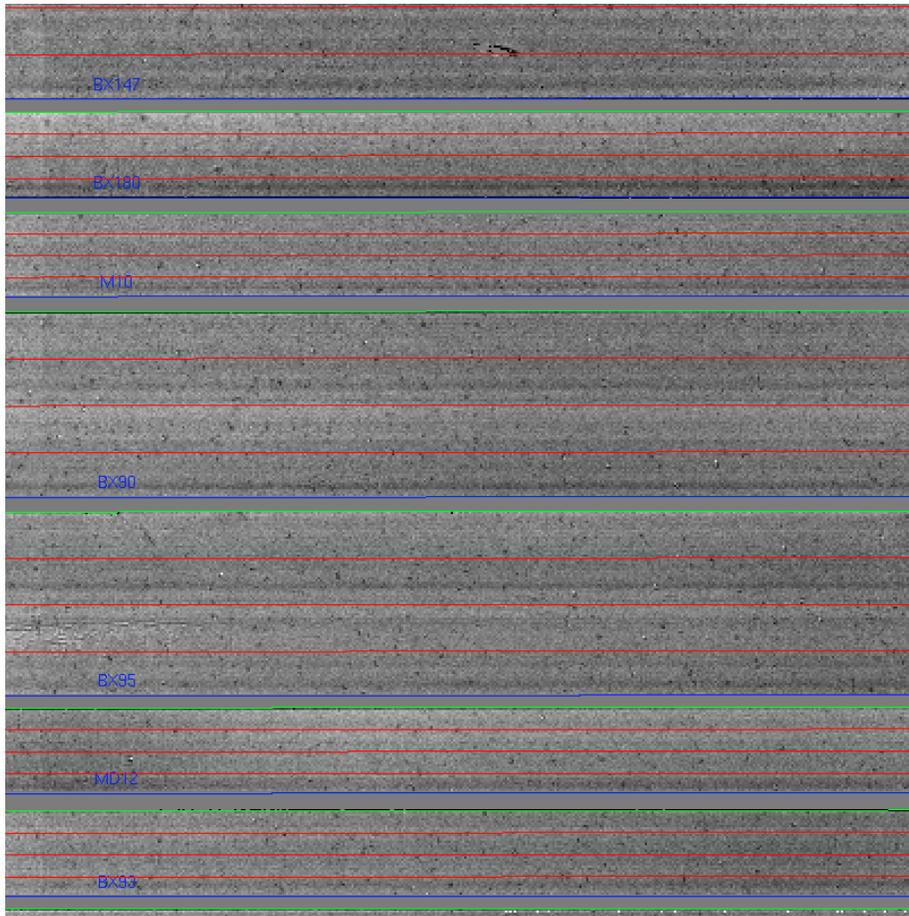
Figure 2: Screenshot

```
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, H)

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.

14

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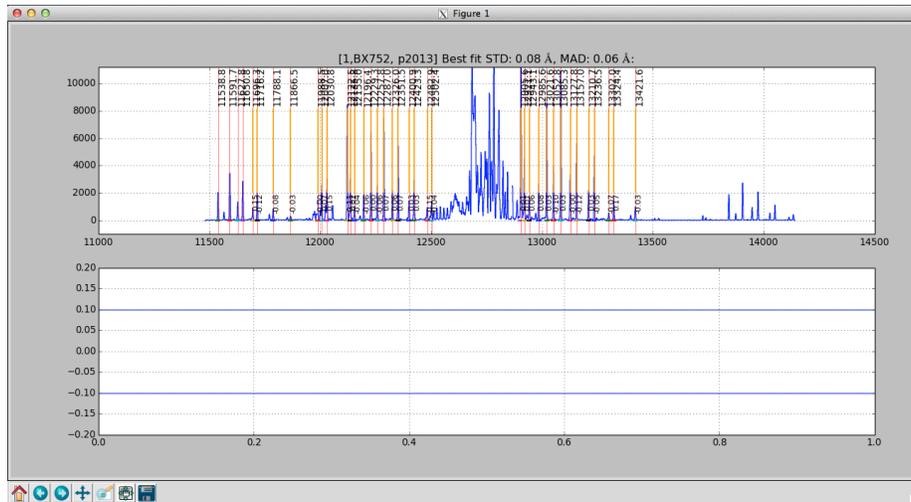
when you run this step, a GUI window appears.

Figure 3: Screenshot

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. 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.

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
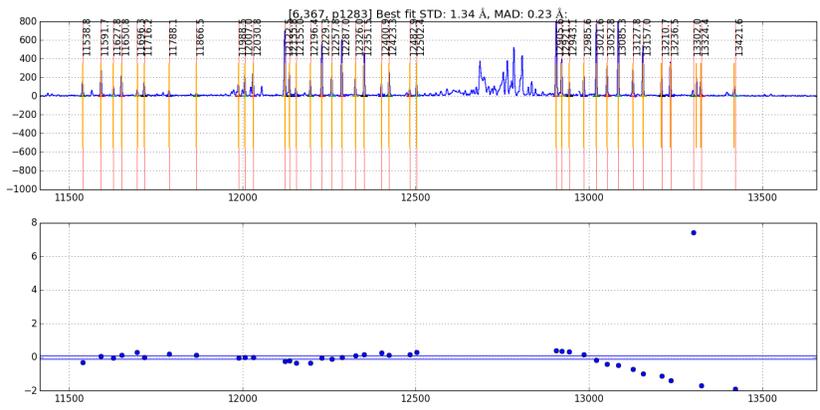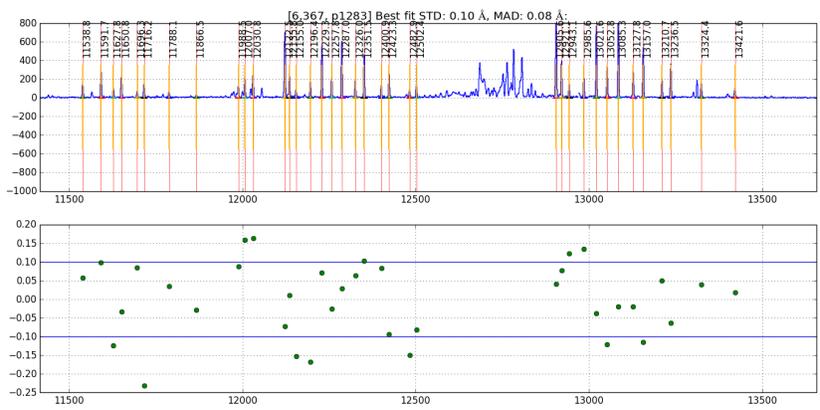
15

Figure 4: Screenshot

Figure 5: Screenshot

16

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_wavestack_band.npy	The coefficients of the band and pixel 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_wavestack_mh180114_0443-0445.npy	coefficients of the fit for each 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_mt30014_v0448-0445.fits	contains a map of the wavelength for each pixel in the spectra
sigs_solution_wave_stack_J_mt30014_v0443-0445.fits	contains the uncertainty in the fitted wavelength position for each pixel in the spectra
rectified_wave_stack_J_mt30014_v0443-0445.fits	contains the wavelength rectified resampled sky emission. A column in the image contains all pixels at the same wavelength.

Wavelength Calibration (K)

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=Tru
```

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

The 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, ne
```

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.fit
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'
```

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.

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 the table below.

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

$$sn = \frac{signal \cdot integration\ time}{\sqrt{variance}}$$

Figure 6: Equation

in the MOSFIRE Pre Ship Review as 21 electron) per fowler sample. Thus, the final STD is

$$STD = \frac{21\ electron}{\sqrt{N_{reads}}} + \sqrt{\#\ detected\ electron}$$

Figure 7: Equation

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 next 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 ?

Filename	Content (units)
`[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.

Spectral Extraction

Interactive Spectral Extraction Instructions

The final step is to extract a 1D spectrum for each object in each slit. The final line of the `Driver.py` (or equivalent) file will looks something like this:

`Extract.extract_spectra(maskname, band, interactive=(not bypassflag))`

This will iterate through all the slits for this mask-band combination and if the interactive flag is set, then it will show a plot of the spatial profile of each slit (collapsed in the spectral direction). By default, the software will make a guess at one aperture for each slit. The apertures are indicated by a yellow shaded region and their center position and half width in pixels is annotated near the top of each shaded region.

The apertures define the pixels which will be used as input to the optimal spectral extraction (Horne 1986) algorithm. Having wide a wide aperture should not add additional noise as that will be optimized during the spectral extraction step. The apertures are shown here in order for the user to verify 1) that there is no overlap between adjacent objects, 2) that the apertures are wide enough to reasonably encompass all flux from the object, and 3) that all objects have properly defined apertures.

The user can add, modify, or delete apertures interactively using this plot window.

To delete an existing aperture: place the mouse near the center of the aperture and press the "d" key.

To add an aperture by fitting a gaussian to the profile: place the mouse near the peak of the profile and press the "g" key. The half width of the aperture will be set at 5 times the sigma of the fitted gaussian.
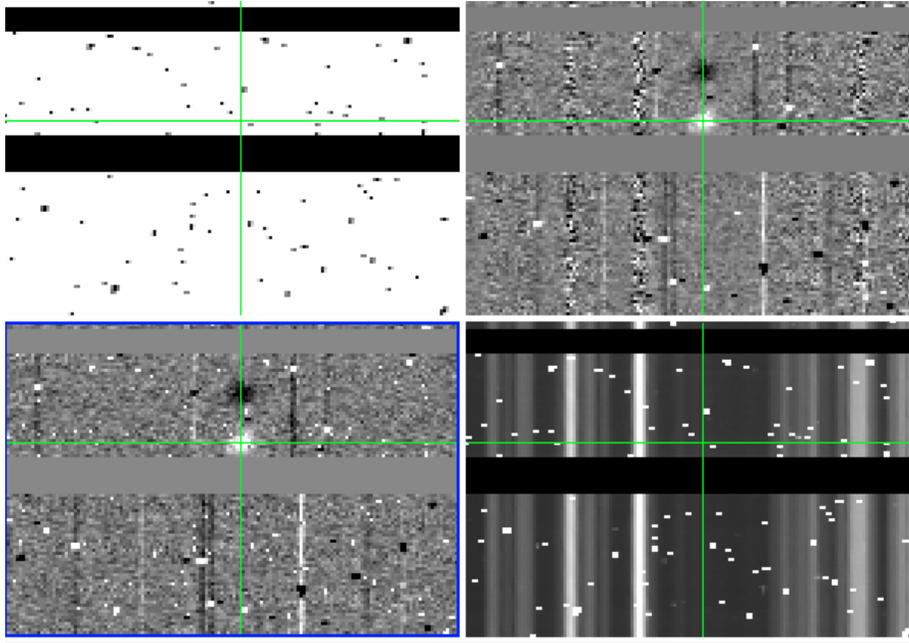
Figure 8: Screenshot

To add an aperture manually: place the mouse in the X position where the new aperture should be centered and press the "a" key. Then type the half width (in pixels) for that aperture in response to the query in the terminal.

To modify the half width of an existing aperture: place the mouse near the center of the aperture and press the "w" key. Then type the half width (in pixels) for that aperture in response to the query in the terminal.

To modify the center position of an existing aperture: place the mouse near the center of the aperture and press the "p" key. Then type the position (in pixels) for that aperture in response to the query in the terminal.

When you are done adding or removing apertures, close the interactive plot window by clicking the close button in the upper right corner (or by whatever method is typical for your OS or windowing system) or press the "q" or "n" keys (for "quit" or "next" respectively).

Spectral Extraction Results

Whether you used the interactive tool for spectral extraction or allowed the software to automatically guess at the apertures to extract, the software will output both a FITS version of the resulting 1D spectrum and an PNG plot.
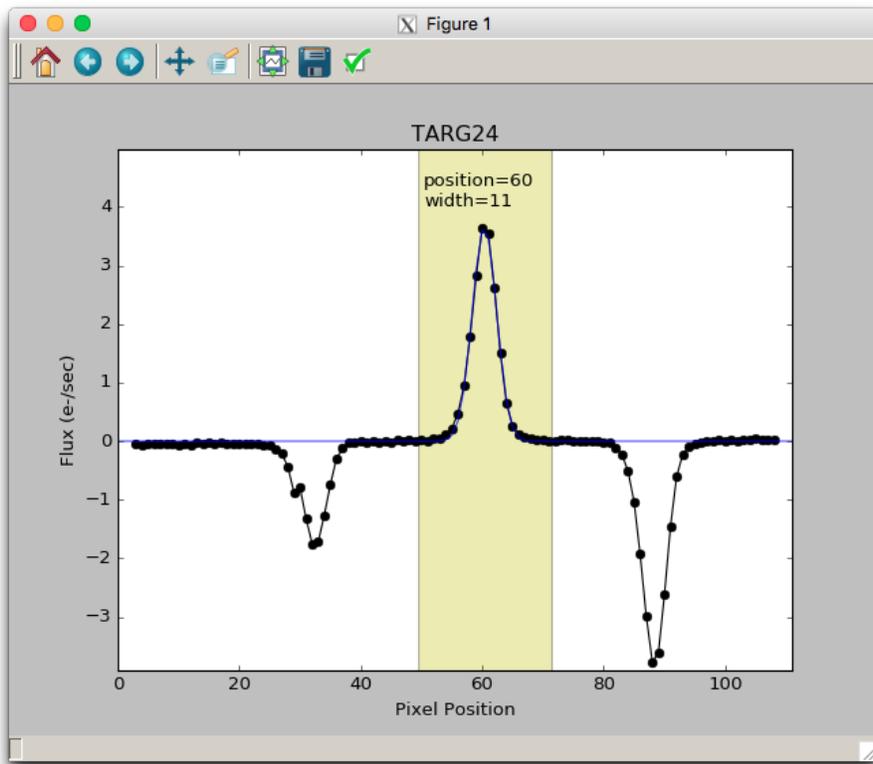
Figure 9: Screenshot
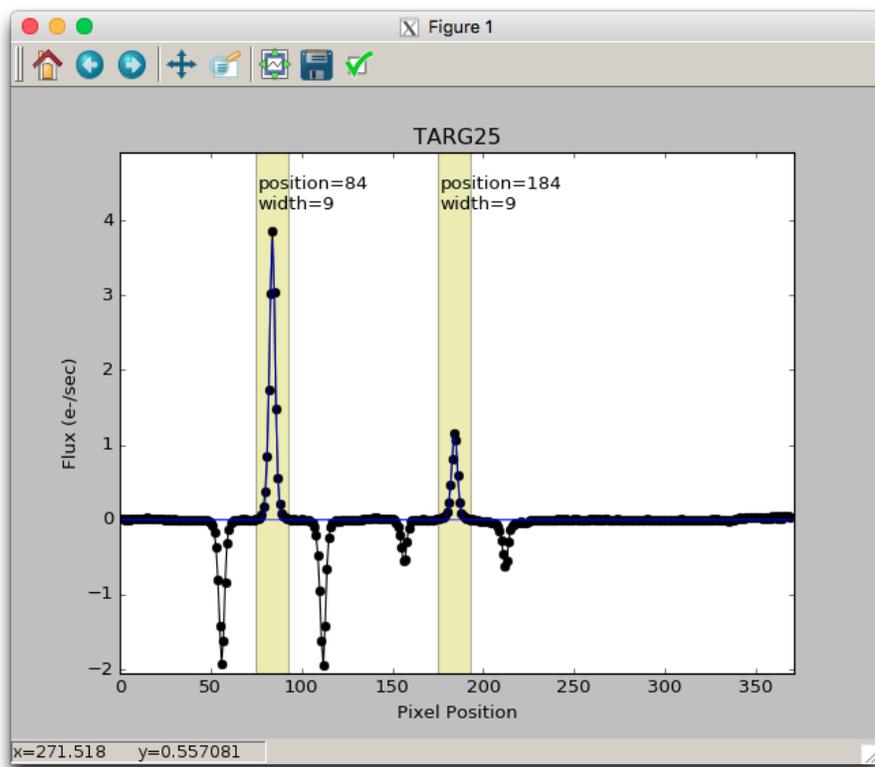
Figure 10: Screenshot

These filenames will have the form:

```
[maskname]_[band]_[targetname]_[aperture].png
[maskname]_[band]_[targetname]_1D_[aperture].fits
```

where `[aperture]` is a two digit integer indication which aperture for that slit this corresponds to (zero based). If the apertures were determined automatically by the software, then only one aperture will have been generated for each slit, so all files will end in `_00.png` or `_1D_00.fits`.

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', longsl
```

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
Wavelength.fit_lambda(maskname, band, argon, argon, waveops, longslit=longslit)
Wavelength.apply_lambda_simple(maskname, band, argon, waveops, longslit=longslit, smooth=Tru
```

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.

- `cd /path/to/LONGSLIT/`
- `cp ~/MosfireDRP-master/drivers/Longslit_driver.py .`

- 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.
- 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.
- edit `Driver_Longslit.py`
  - Examine a longslit image (see figure below) and adjust 'yrange': [709, 1350] to the vertical range covered by the slit
  - From the same examined longslit, select 'row_position' so that it is uncontaminated by the spectrum. See Figure 1.
  - make sure that 'mode':'longslit' is specified in the longslit variable
  - The result should look like Figure 2.
- 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.

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

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.
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:

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###.

Figure 11: Screenshot

Figure 12: Screenshot

Figure 13: Screenshot

Hints

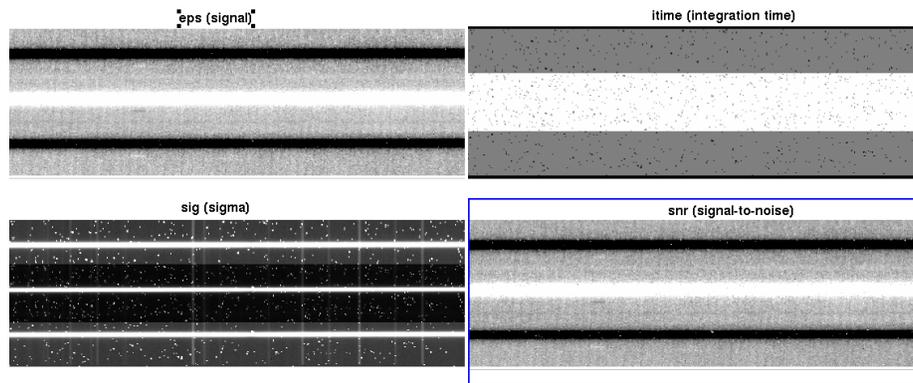## Pay attention to the wavelength fitting output

Figure 14: Screenshot

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.

Examples of Running the Pipeline

Here we demonstrate three walkthroughs of how to run the MOSFIRE pipeline. We include a longslit reduction, a slitmask reduction, and a long2pos reduction in the H and K bands. Example datasets can be downloaded from this link, and includes all three data types. Details of each step in the reduction process can be seen in the manual, so please read the previous sections in the manual to understand the reductions being performed on these data.

Getting Started

After downloading and unzipping all of the test data, make a directory in your preferred location to perform your reduction and run handle.

```
mkdir reduced
cd reduced
mospy handle /path/to/data/test_dataset/*.fits
```

You should see five new directories after handle is done.

```
LONGSLIT-3x0.7 <-- Longslit observations
LONGSLIT-46x0.7 <-- Longslit calibrations
MOSFIRE_DRP_MASK <-- Slitmask calibrations and observations
long2pos <-- long2pos calibrations
long2pos_specphot <-- long2pos_specphot observations
```

Longslit Reduction

For more information on the reduction of longslits, go to the [longslit][longslit] section.

Move to the Longslit observation directory and copy the calibrations to the observation directory:

```
cd LONGSLIT-3x0.7/2012nov28/K
cp ../../../LONGSLIT-46x0.7/2013oct15/K/*.txt ./
```

Run the autodriver to create the driver file Longslit_HIP17971.py

Edit the driver file so the y-range so it covers the range of the slit, and 'row_position' is not contaminated with the spectrum (you only want sky lines). Look at the files in Offset_-5_HIP17971.txt and Offset_5_HIP17971.txt to determine where these values should be.

Run the pipeline using:

```
mospy Longslit_HIP17971.py
```

You can run step by step by commenting out reduction steps in the driver file, or you can run all at once. First the pipeline will create a flat field, as described in the Flats section. Check out your combined and pixel flats to make sure they look reasonable, no odd edges to slits.

We do not recommend non-interactive wavelength reductions unless you have a previous reliable wavelength solution in the same directory ("lambda_coeffs_wave_stack..."). Refer to the Wavelength Calibration sections (for H,J, and Y, or K band) for information on how to perform the interactive fit. In this case, we will be performing a K band solution, so check that section for more details.

After background subtraction and rectification, the output (notably, HIP17971_K_eps.fits, HIP17971_K_itime.fits, HIP17971_K_sig.fits, HIP17971_K_snrs.fits) you get should look like this:
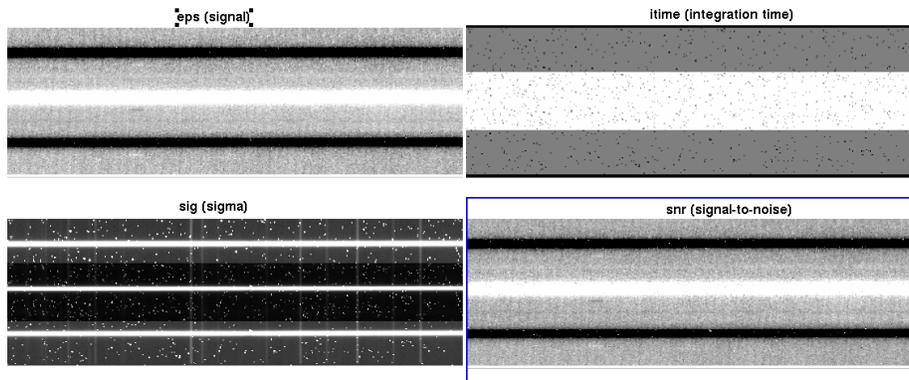
Figure 15: Longslit DRP output

For longslits with bright continuua (such as in this example), a spectral extraction if you should wish to perform one will be very simple. In the case of a bright continuum like this, you may or may not choose to perform the aperture selection manually, but we encourage manual control of the apertures whenever reasonable, even if just to check the aperture is correct.

Slitmask Reduction

Move to the MOSFIRE_DRP_MASK directory and run the autodriver:

```
cd MOSFIRE_DRP_MASK/2012sep10/H/
mospy AutoDriver
```

The driver file will be called Driver.py. Open the file to make sure there are no abnormalities, and proceed to call Driver.py with either the full reduction, or step by step.

```
mospy Driver.py
```

Again, we recommend a manual wavelength solution. Once the wavelength solution is propogated, we recommend checking the wavelength propagation by opening lambda_solution_wave_stack_H_m120910_0163-0176.fits to check that no slits are missing and as you check the intensity levels (which will be the wavelength values) for reasonable numbers for your filter (H band in this case).

After rectification, the output whole masks will look like this:

And the entire mask (zoomed out) will look like this:

Even with bright continuua like this slitmask has, we recommend manually controlling the apertures in case secondary objects are in the slits. In the case
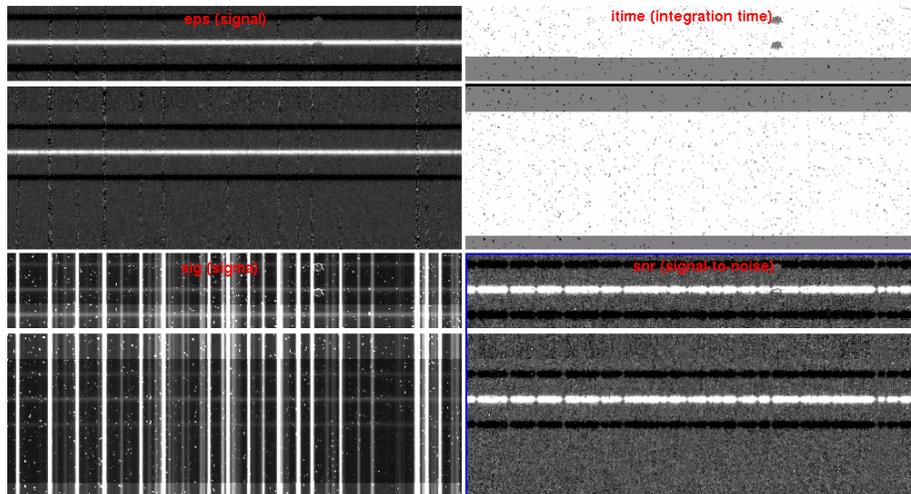
Figure 16: Slitmask DRP output

Figure 17: Slitmask eps output

of faint or emission line only objects, it might be necessary to manually enter where an aperture should be located and the width of the extraction.

Long2pos_specphot Reduction

Move to the long2pos_specphot directory. Since calibrations for long2pos can be used for long2pos_specphot, copy those calibration files to your long2pos_specphot directory:

```
cd long2pos_specphot/2017sep10/K
cp ../../../long2pos/2017sep29/K/*.txt ./
```

Run the autodriver and open its output, Long2pos_ZW_TEL.py

Either comment out or copy the extract command for each desired output target (positions A or C, wide or narrow). Then run the driver file step by step or as a full reduction.

After rectification, your output eps files for PosA and PosC narrow and wide will look like this:
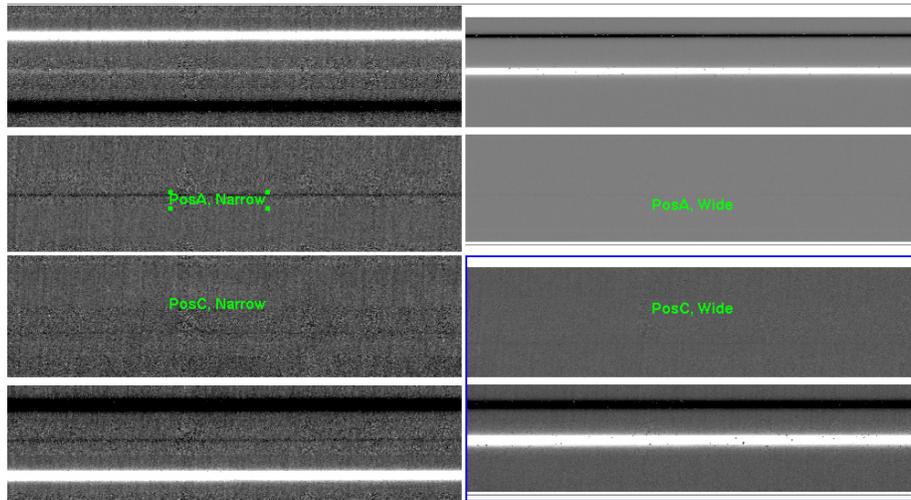

Figure 18: Output long2pos_specphot images