Changes

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- Added procedure to fix the header of old images acquired with an outdated version of long2pos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) Install Ureka

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

   mkdir ~/MOSFIRE mv MosfireDRP-1.1 ~/MOSFIRE/DRP_CODE cd ~/MOSFIRE/DRP_CODE to navigate to that directory.

3) Create Data Directories

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:

- a raw data directory in the code repository: `mkdir ~/MOSFIRE/DRP_CODE/DATA`
- 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

Navigate to the newly creted bin dir:

```
cd ~/MOSFIRE/DRP_CODE/bin
```

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 your favorite editor (emacs ../bin/mospy), update the path for the `ur_setup`. Replace `/home/npk/.ureka/ur_setup` with your `/your_full_path_name/.ureka/ur_setup` full path.

Update the path for the `ur_forget`. Replace `/home/npk/.ureka/ur_setup` with `/your_full_path_name/.ureka/ur_forget`

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:

```
#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:

```
#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 mospy is executable

```
chmod +x mospy
```

7) Update your .cshrc file

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 )
```

for example:

```
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 directory. 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
```

This will put 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.

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: http://mosfire.googlecode.com/files/DRP_Test_Case_Hband.zip.

Move the test case data to the newly created data dir

```
mv ~/Downloads/DRP_Test_Case_Hband.zip ~/MOSFIRE/DRP_CODE/DATA/.
```

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.

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:

```
cd ~/MOSFIRE/DRP_CODE/REDUX # Go to your output directory
mospy handle /home/yourhomedir/MOSFIRE/DRP_CODE/2014may08*fits
```

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

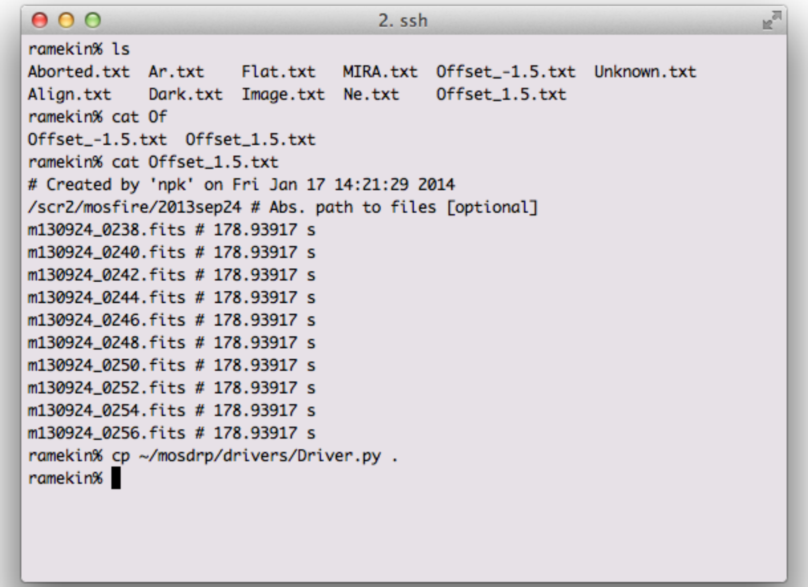
```
[maskname]/[date]/[band]/
```

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.

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



```
ramekin% ls
Aborted.txt  Ar.txt    Flat.txt   MIRA.txt  Offset_-1.5.txt  Unknown.txt
Align.txt    Dark.txt  Image.txt  Ne.txt    Offset_1.5.txt
ramekin% cat Of
Offset_-1.5.txt  Offset_1.5.txt
ramekin% cat Offset_1.5.txt
# Created by 'npk' on Fri Jan 17 14:21:29 2014
/scr2/mosfire/2013sep24 # Abs. path to files [optional]
m130924_0238.fits # 178.93917 s
m130924_0240.fits # 178.93917 s
m130924_0242.fits # 178.93917 s
m130924_0244.fits # 178.93917 s
m130924_0246.fits # 178.93917 s
m130924_0248.fits # 178.93917 s
m130924_0250.fits # 178.93917 s
m130924_0252.fits # 178.93917 s
m130924_0254.fits # 178.93917 s
m130924_0256.fits # 178.93917 s
ramekin% cp ~/mosdrp/drivers/Driver.py .
ramekin% 
```

Figure 1: Screenshot

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.

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 file controls all the pipeline steps, and in the drivers subdirectory, 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`
2. Copy the appropriate driver file: `cp ~/ MOSFIRE/DRP_CODE/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:

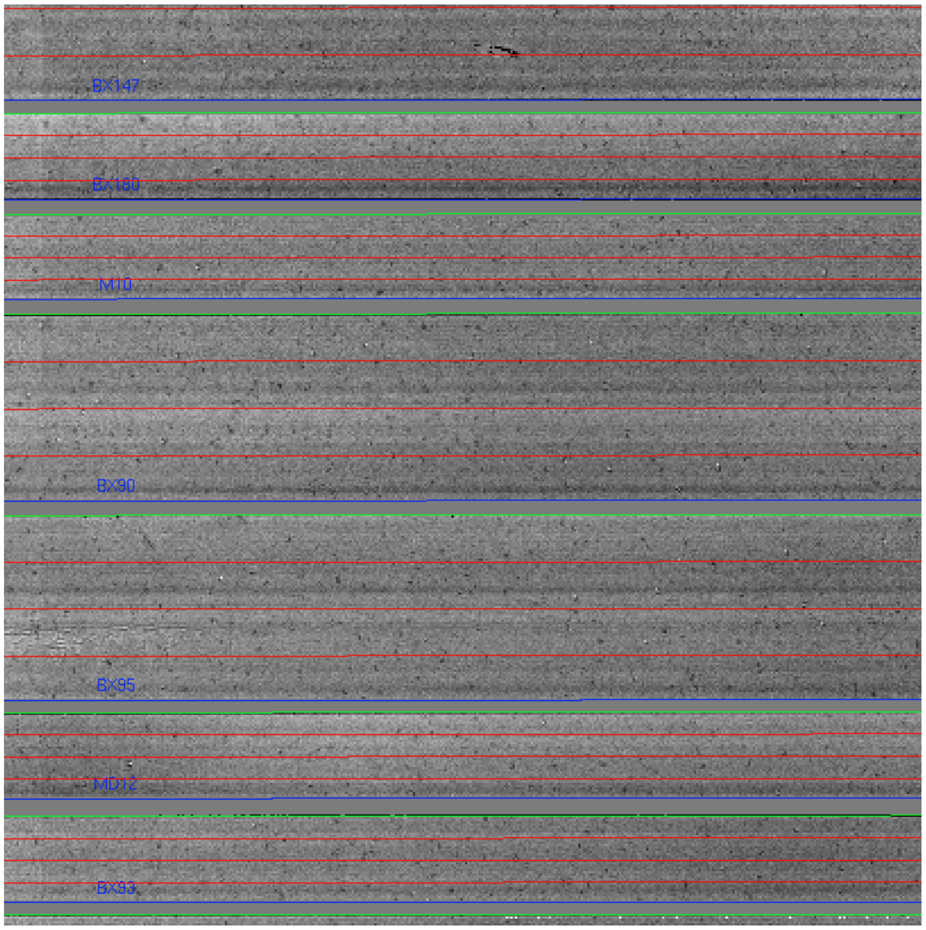


Figure 2: Screenshot

```
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, 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.