Draft version June 20, 2019
Typeset using LaTeX **manuscript** style in AASTeX62

Jupyter Notebook Tutorial

1. INTRODUCTION

This tutorial explains the basic data reduction process for the SOAR Adaptive-Module Optical Spectrograph (SAMOS). The accompanying jupyter notebook is aptly named ‘`jupyter_SAMOS`’.

2. STARTING UP

As of now, the pipeline uses PyRAF for spectral line identification, which requires Python2. The environment file `EnvSAMOS.yml` is located in the main working directory. The user may copy the environment to their own computer by executing ‘`conda env create --file EnvSAMOS.yml`’ in their terminal.

The user's working directory should contain the following:

- `SAMOS-working-dir/`:
  - `cleaner.sh`
  - Readme.md
  - `Docs/`
  - `jupyterNB_tutorial/` **you are here**
  - `LDSS3/`
  - `helper_files/`
  - `linelists/`
  - `py_mods/`
- `SAMOS-working-dir/py_mods/`:

  - `CreateFuelStructure.py`
  - `CreateInput.py`
  - `CreateSlitStructure.py`
  - `FlatNorm.py`
  - `InitializeSAMOS.py`
  - `NormDivFlats.py`
  - `OutlineSlits.py`
  - `Overscan.py`
  - `OverscanAndTrim.py`
  - `SAMOSHelpers.py`
  - `SlitCutout.py`
  - `SlitID.py`
  - `PyrafIdentify.py`
  - `PyrafTransform.py`
  - `iraf_steps.py`
  - `iraf_params.py`
  - `viewFITS.py`
  - `StartFrom.py`
  - `__init__.py`

- `SAMOS-working-dir/helper_files`:
  - LMask1.SMF
  - LMask2.SMF
  - LMask1_ycoords_c1.txt
  - LMask2_ycoords_c1.txt

– LMask2_ycoords_red_c1.txt

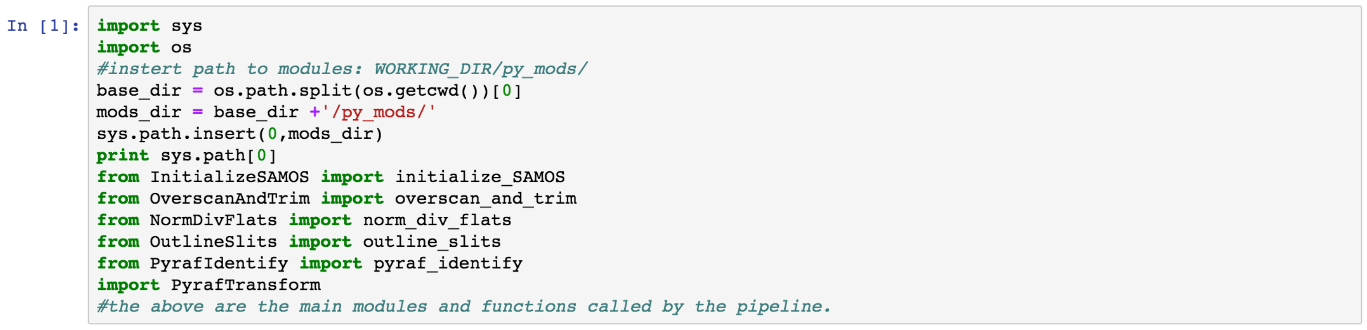
To run this pipeline on their own, I recommend the user create a separate working directory with symbolic links from the cloned repository. In your terminal, start a new session from scratch by running '`./cleaner.sh`' in the working directory to clean up files and directories left over from previous runs. Then, open the JupyterNB tutorial by typing '`jupyter-notebook SAMOS-working-dir/jupyterNB_tutorial/jupyter_SAMOS.ipynb`' in your terminal. Note that the helper files are specifically for reducing the LDSS3 test data. These files will be obsoleted when we have sufficient data from the SAMOS instrument. When that happens, I will be adjusting the pipeline according to the instrument data. It should therefore be noted that parts of the pipeline that call these helper files employ less robust data reduction methods.

Eventually, the pipeline will be used to reduce multi-object spectroscopy data taken with the SOAR Adaptive-Module Optical Spectrograph (SAMOS). For information on the instrument, see (Robberto et al. 2016). In order to avoid reinventing the wheel, this pipeline is heavily based on the Flame DRP written by Belli, Contursi, and Davies (2017). Flame is written in IDL, while the SAMOS pipeline will ultimately be written in Python2.

3. NOTEBOOK WALKTHROUGH

3.1. `fuel_init = initialize_SAMOS(data_dir,mask)`

The first cell of notebook sets up the paths to the various modules needed to run the pipeline so the notebook can find and use them.

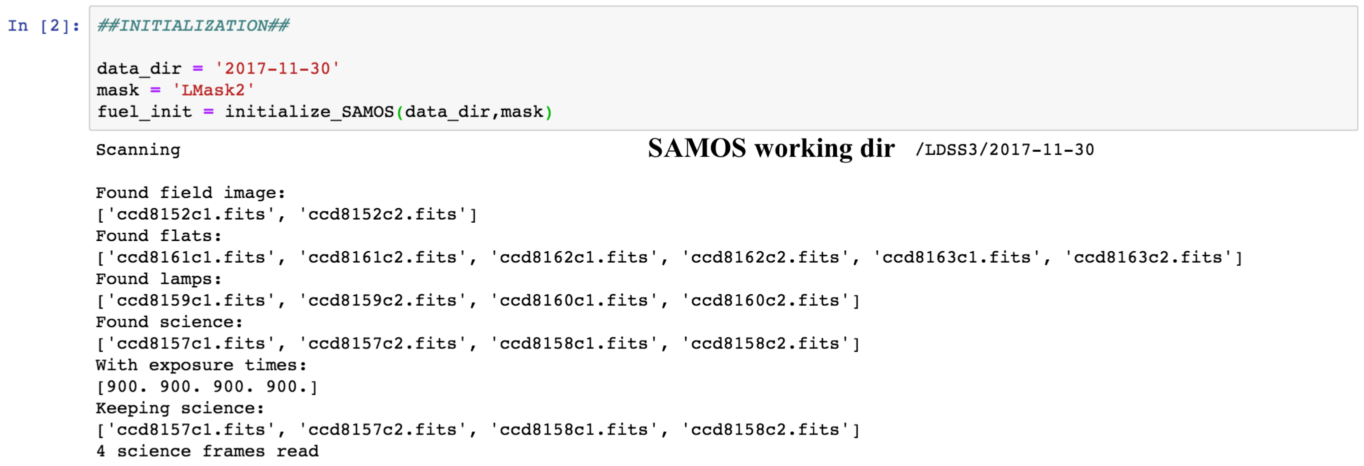


```
In [1]: import sys
        import os
        #instert path to modules: WORKING_DIR/py_mods/
        base_dir = os.path.split(os.getcwd())[0]
        mods_dir = base_dir +'/py_mods/'
        sys.path.insert(0,mods_dir)
        print sys.path[0]
        from InitializeSAMOS import initialize_SAMOS
        from OverscanAndTrim import overscan_and_trim
        from NormDivFlats import norm_div_flats
        from OutlineSlits import outline_slits
        from PyrafIdentify import pyraf_identify
        import PyrafTransform
        #the above are the main modules and functions called by the pipeline.
```

Figure 1. Cell initialization for a new data reduction session.

First, the pipeline runs the function `initialize_SAMOS`, which the FITS headers creates the basic data structure, separating the calibration files from the science files.

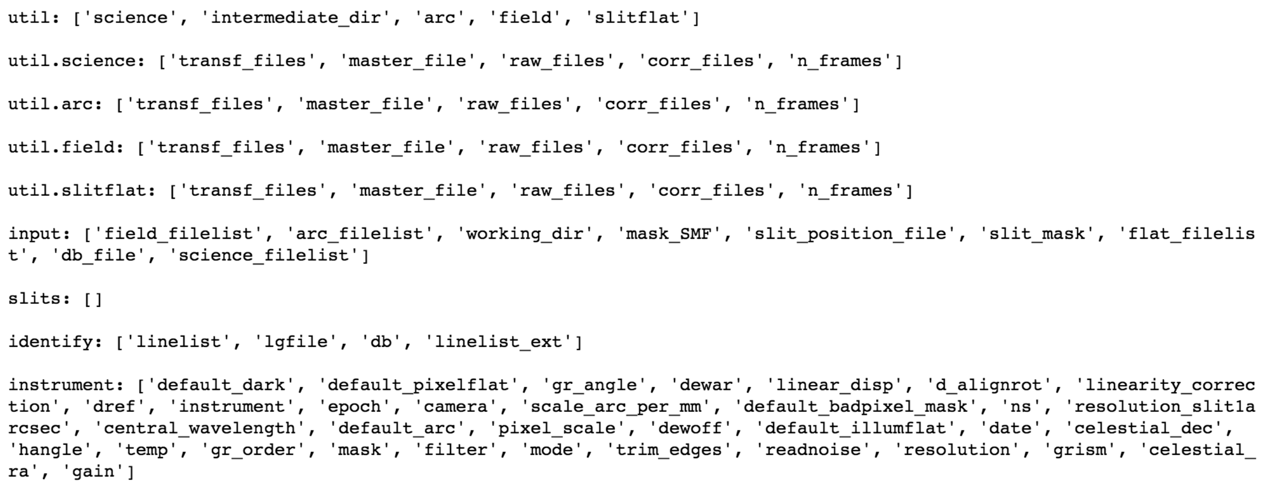
The output while the script is running will give the names of files blah.



```
In [2]: ##INITIALIZATION##

        data_dir = '2017-11-30'
        mask = 'LMask2'
        fuel_init = initialize_SAMOS(data_dir,mask)

        Scanning                                    SAMOS working dir  /LDSS3/2017-11-30

        Found field image:
        ['ccd8152c1.fits', 'ccd8152c2.fits']
        Found flats:
        ['ccd8161c1.fits', 'ccd8161c2.fits', 'ccd8162c1.fits', 'ccd8162c2.fits', 'ccd8163c1.fits', 'ccd8163c2.fits']
        Found lamps:
        ['ccd8159c1.fits', 'ccd8159c2.fits', 'ccd8160c1.fits', 'ccd8160c2.fits']
        Found science:
        ['ccd8157c1.fits', 'ccd8157c2.fits', 'ccd8158c1.fits', 'ccd8158c2.fits']
        With exposure times:
        [900. 900. 900. 900.]
        Keeping science:
        ['ccd8157c1.fits', 'ccd8157c2.fits', 'ccd8158c1.fits', 'ccd8158c2.fits']
        4 science frames read
```

Figure 2. output from running the first step of the code.

Each instance of the fuel structure returned by `initialize_SAMOS` is organized as follows:



```
util: ['science', 'intermediate_dir', 'arc', 'field', 'slitflat']

util.science: ['transf_files', 'master_file', 'raw_files', 'corr_files', 'n_frames']

util.arc: ['transf_files', 'master_file', 'raw_files', 'corr_files', 'n_frames']

util.field: ['transf_files', 'master_file', 'raw_files', 'corr_files', 'n_frames']

util.slitflat: ['transf_files', 'master_file', 'raw_files', 'corr_files', 'n_frames']

input: ['field_filelist', 'arc_filelist', 'working_dir', 'mask_SMF', 'slit_position_file', 'slit_mask', 'flat_filelis
t', 'db_file', 'science_filelist']

slits: []

identify: ['linelist', 'lgfile', 'db', 'linelist_ext']

instrument: ['default_dark', 'default_pixelflat', 'gr_angle', 'dewar', 'linear_disp', 'd_alignrot', 'linearity_correc
tion', 'dref', 'instrument', 'epoch', 'camera', 'scale_arc_per_mm', 'default_badpixel_mask', 'ns', 'resolution_slit1a
rcsec', 'central_wavelength', 'default_arc', 'pixel_scale', 'dewoff', 'default_illumflat', 'date', 'celestial_dec',
'hangle', 'temp', 'gr_order', 'mask', 'filter', 'mode', 'trim_edges', 'readnoise', 'resolution', 'grism', 'celestial_
ra', 'gain']
```

Figure 3. Base data structure created upon initialization.

The pipeline gets most of the initial information from header files and creates empty files and directories for future use, but slit locations for the current test data were created manually and stored in one of the helper files. The SAMOS instrument is a Digital Micromirror Device (DMD)

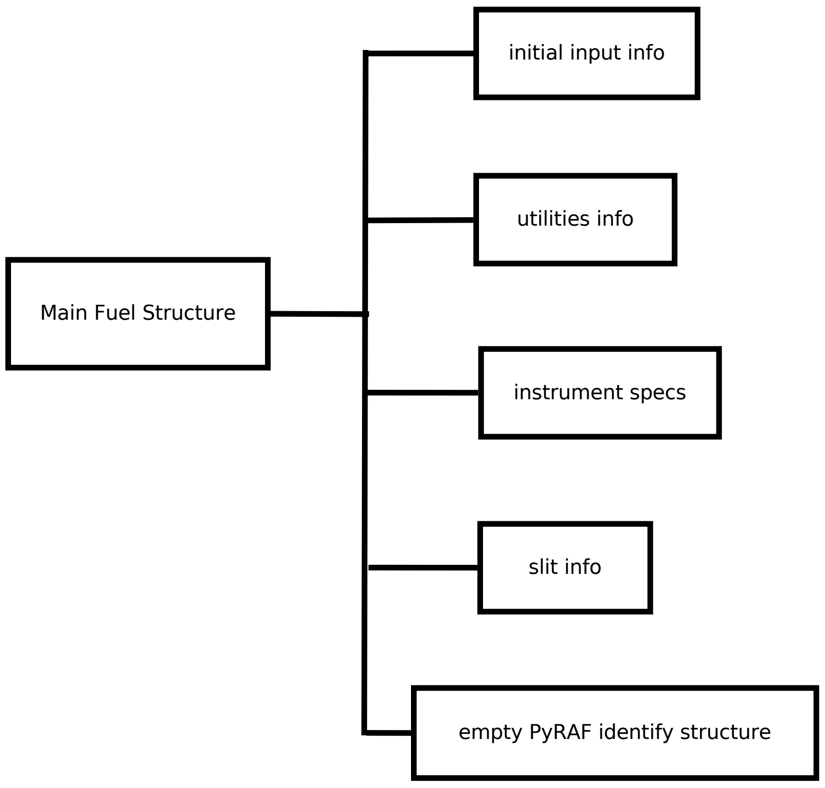


Figure 4. Base data structure flowchart.(Until I make more nice flow charts that better demonstrate the data structures, I will include screenshots of printed instance attributes when relevant.)

(Smee et al. 2018), allowing for multiple slit configurations in a single observing run. Each exposure will have the slit configurations stored in the FITS headers so the pipeline will be able to access them easily and have them stored upon instantiation. For now, the slit position file is stored as `fuel_init.input.slit_position_file`.

3.2. `fuel_ost = overscan_and_trim(fuel_init)`

Next, we have to determine the overscan and trim the edges of the fits data. This code reads the database file and creates arrays which separate the data into flats, comparison lamps, and science data.

The main part of `overscan_and_trim` is the calling of the module `Overscan.py`. This module parses the fits data array into regions of bias and target data by reading the fits headers BIASSEC and DATASEC respectively. There is a function in `Overscan.py` called `FieldTrim(...)` that trims the field mask to the correct size.

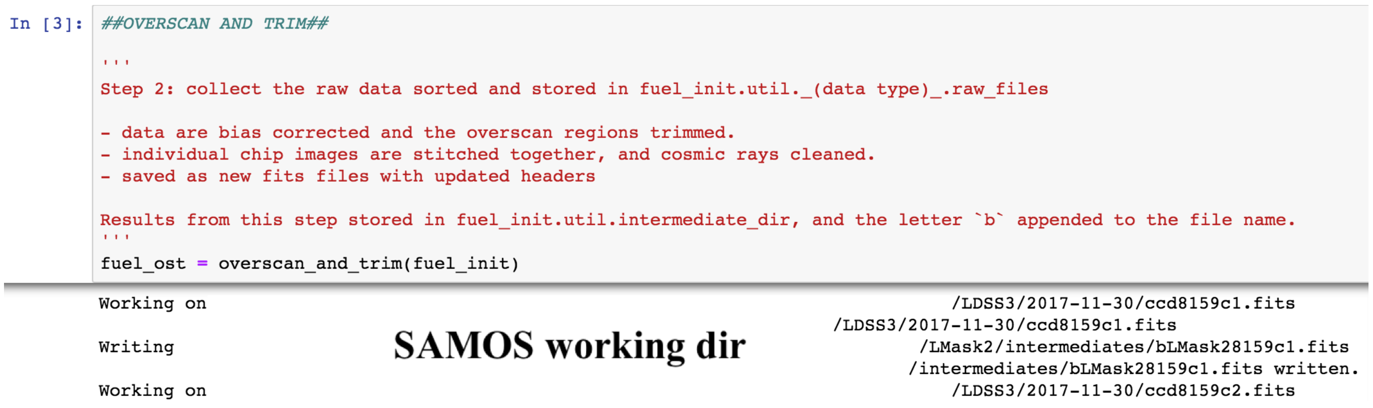
The bias is due to a combination of the camera noise from the readout process and electric "pre-charge" on a CCD chip by the electronics. The BIASSEC header gives the region of overscan, which contains information for this bias. An array of this overscan region is made by grabbing the rows and columns from the data array (`d = f[0].data.astype(‘‘f’’)` $\rightarrow$ `overscan = d[biassec rows, biassec columns]`). The function then takes the median (or mean) of the overscan regions along the rows (axis=1), and subtracts these values from each data value in their respective data rows. The main procedure is shown in the figure below.

$$\begin{pmatrix} cropped & and \\ bias & subtraced \\ data & matrix \end{pmatrix} = \begin{pmatrix} d11 & d12 & d13 & ... & d1m \\ d21 & d22 & ... & ... & d2m \\ ... & ... & ... & ... & ... \\ dn1 & ... & ... & ... & dnm \end{pmatrix} - \begin{pmatrix} [b1] \\ [b2] \\ [b3] \\ [...] \\ [bn] \end{pmatrix} \quad (1)$$

During this task, the pipeline prints the input (`fuel_ost.util.DTYPE.raw_files`) and output (`fuel_ost.util.intermediate_dir`) file locations as it progresses.

3.3. `fuel_ndf = norm_div_flats(fuel_ost)`

Flat field images used to charactarize the variation in pixel sensitivity across the detector. During this step, each flat is scaled by its median. The data are then median-combined and normalized to



```
In [3]: ##OVERSCAN AND TRIM##

        '''
        Step 2: collect the raw data sorted and stored in fuel_init.util._(data type)_.raw_files

        - data are bias corrected and the overscan regions trimmed.
        - individual chip images are stitched together, and cosmic rays cleaned.
        - saved as new fits files with updated headers

        Results from this step stored in fuel_init.util.intermediate_dir, and the letter `b` appended to the file name.
        '''
        fuel_ost = overscan_and_trim(fuel_init)

        Working on                                                             /LDSS3/2017-11-30/ccd8159c1.fits
                                                                      /LDSS3/2017-11-30/ccd8159c1.fits
        Writing           SAMOS working dir                                /LMask2/intermediates/bLMask28159c1.fits
                                                                       /intermediates/bLMask28159c1.fits written.
        Working on                                                             /LDSS3/2017-11-30/ccd8159c2.fits
```

Figure 5. Step 2 for bias subtraction and overscan trimming.

create the master flat. Finally, the pipeline divides the master flat from the science and arc lamp images.

The user can view FITS files used/created by the pipeline by calling the `view_FITS(filename)` function from the `viewFITS` module, shown in figure 6.

```python
from viewFITS import viewFITS
```

```python
master_flat = viewFITS().view_FITS(fuel_ost.util.slitflat.master_file)
master_flat.fig #display figure
```

```
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU     108   (2048, 1500)   float32
```

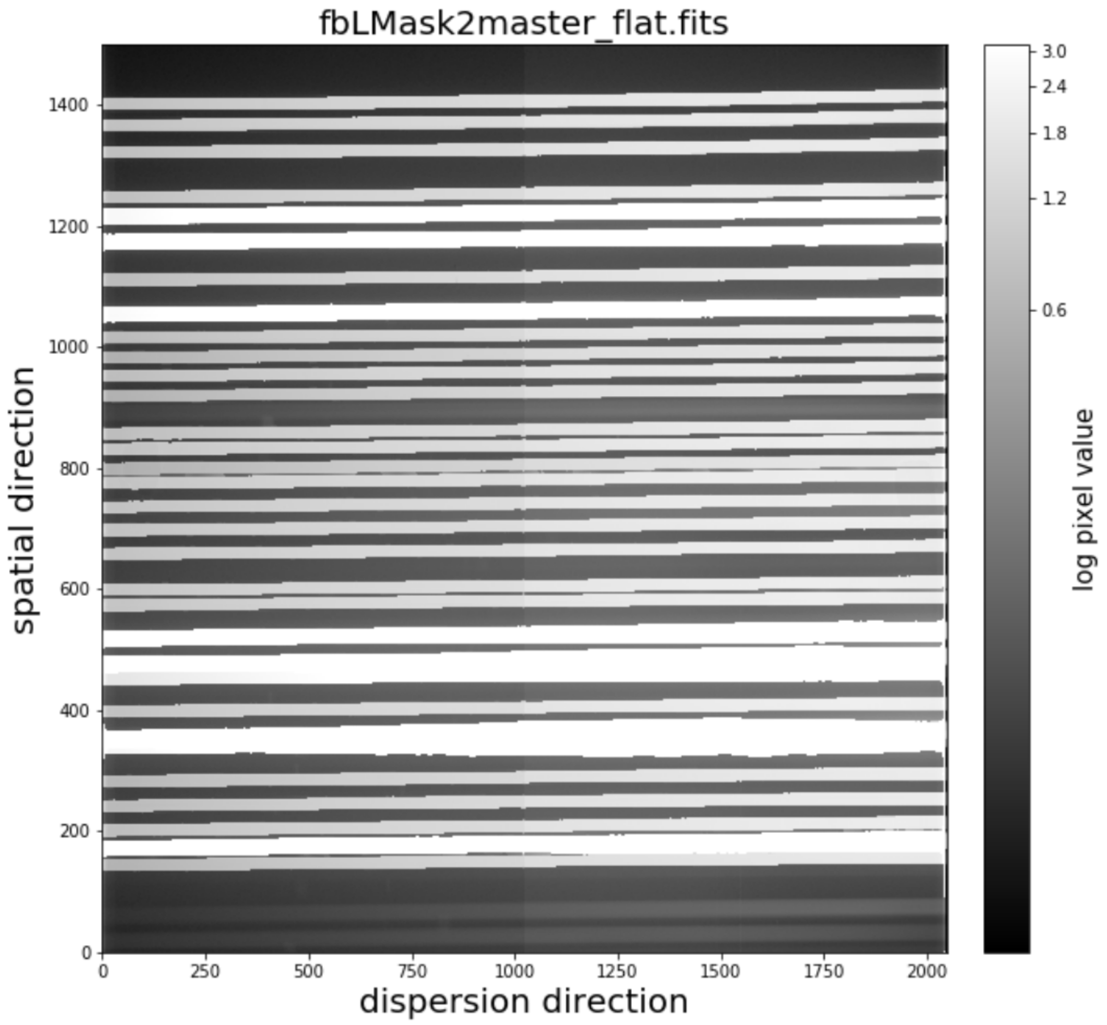


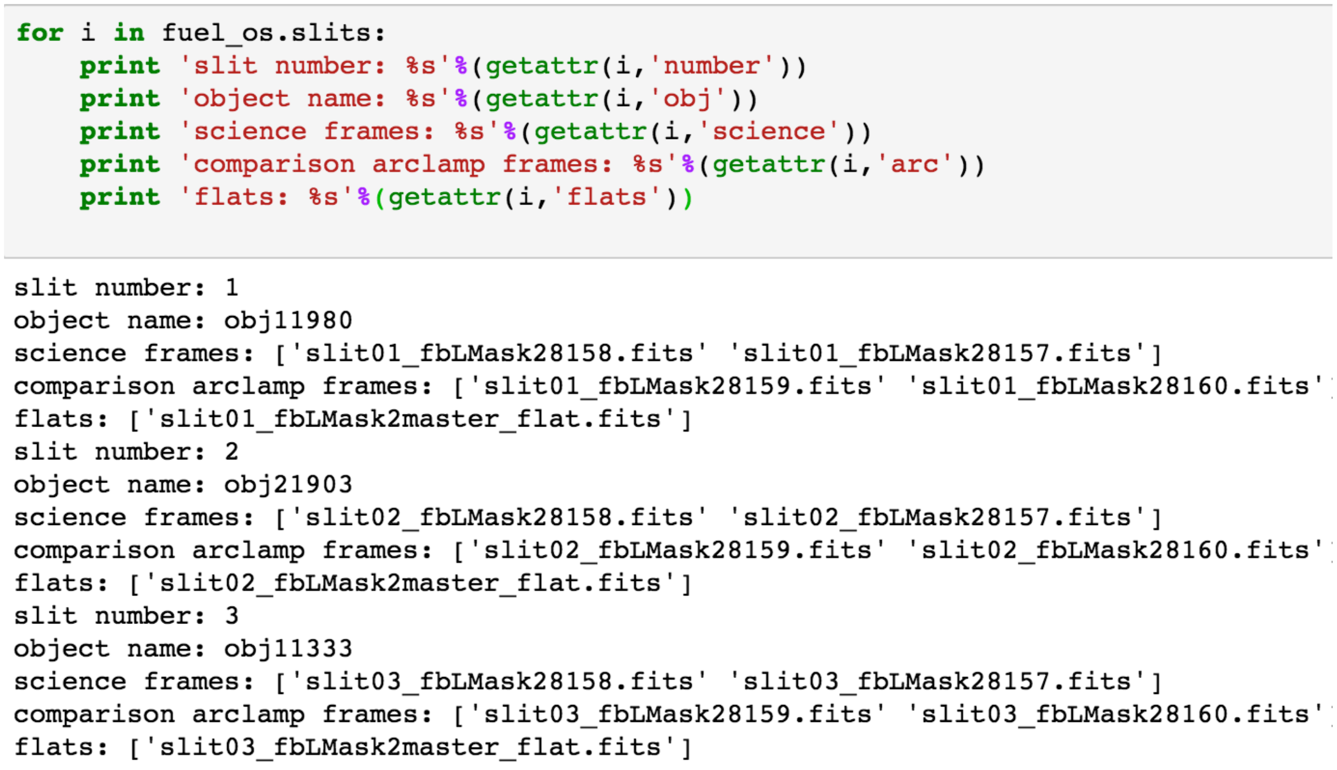
Figure 6. Master slit flat created by the function `norm_div_flats`.

3.4. `fuel_os = outline_slits(fuel_ndf)`

The current version of this pipeline uses data from LDSS3 on Magellan, which uses slit masks for multi-object spectroscopy. Since this will be replaced in the future, the method for identifying and excising individual slits from the images uses a manually made text file of pixel locations. SAMOS is desigend with a large Digital Micromirror Device (DMD). The mirrors can be programmed to direct light to a spectrograph arm or an imaging arm, with the slit configurations stored in the FITS headers for each exposure.

Eventually, this pipeline will be modified to analyze real instrument data.

As the pipeline cycles through each slit, it creates an instance of the `CreateSlitStructure` class, and stores it in `fuel_os.slits` as a list element. Some attributes for a few of the slits are shown in figure 7.



```
for i in fuel_os.slits:
    print 'slit number: %s'%(getattr(i,'number'))
    print 'object name: %s'%(getattr(i,'obj'))
    print 'science frames: %s'%(getattr(i,'science'))
    print 'comparison arclamp frames: %s'%(getattr(i,'arc'))
    print 'flats: %s'%(getattr(i,'flats'))
```

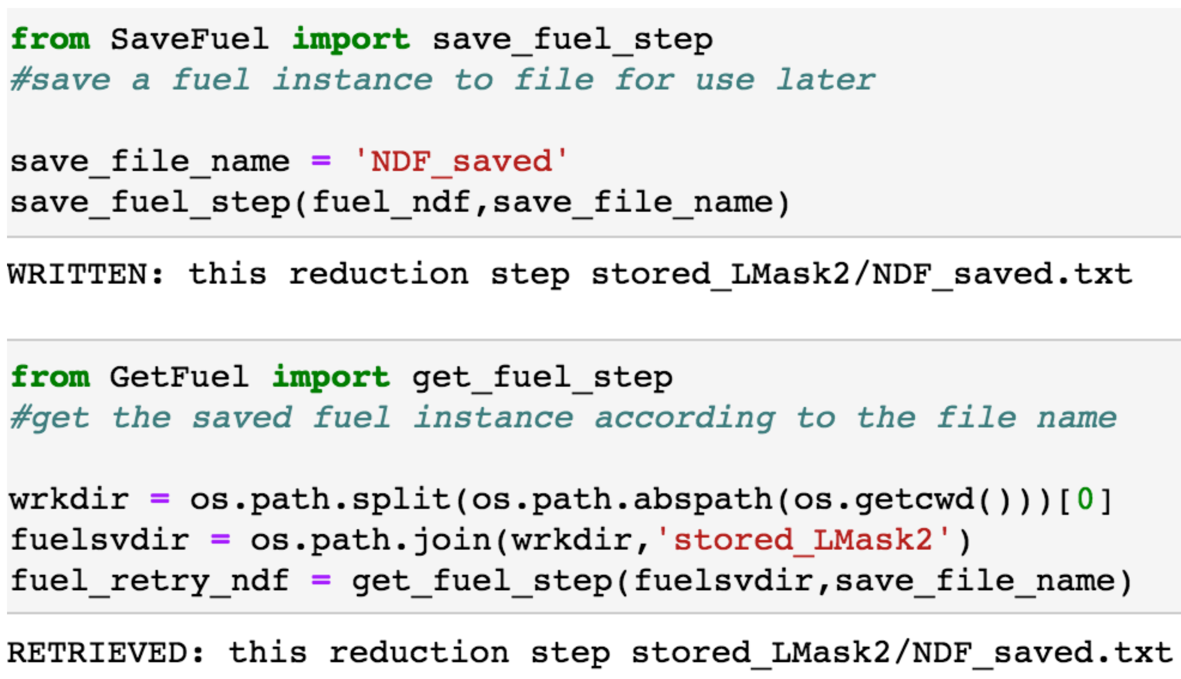
```
slit number: 1
object name: obj11980
science frames: ['slit01_fbLMask28158.fits' 'slit01_fbLMask28157.fits']
comparison arclamp frames: ['slit01_fbLMask28159.fits' 'slit01_fbLMask28160.fits'
flats: ['slit01_fbLMask2master_flat.fits']
slit number: 2
object name: obj21903
science frames: ['slit02_fbLMask28158.fits' 'slit02_fbLMask28157.fits']
comparison arclamp frames: ['slit02_fbLMask28159.fits' 'slit02_fbLMask28160.fits'
flats: ['slit02_fbLMask2master_flat.fits']
slit number: 3
object name: obj11333
science frames: ['slit03_fbLMask28158.fits' 'slit03_fbLMask28157.fits']
comparison arclamp frames: ['slit03_fbLMask28159.fits' 'slit03_fbLMask28160.fits'
flats: ['slit03_fbLMask2master_flat.fits']
```

Figure 7. Some attributes for each slit extracted during `fuel_os`.

4. SAVING PIPELINE PROGRESS

The two modules responsible for object serialization are `SaveFuel.py` and `GetFuel.py` under `py_mods`.

The pipeline automatically saves the current class structure after each step and outputs the location of the saved data. The user can also call the `save_fuel_step` on their own using the fuel structure and a filename as input. To reconstruct a fuel structure, call the `get_fuel_step` with the storage directory and file name (without the '.txt' extnesion) as input. Running this function constructs a copy of the class data structure from the saved file and can be used to pick the pipeline up wherever the user left off. An example of this can be seen in figure 8.



```
from SaveFuel import save_fuel_step
#save a fuel instance to file for use later

save_file_name = 'NDF_saved'
save_fuel_step(fuel_ndf,save_file_name)
```

```
WRITTEN: this reduction step stored_LMask2/NDF_saved.txt
```

```
from GetFuel import get_fuel_step
#get the saved fuel instance according to the file name

wrkdir = os.path.split(os.path.abspath(os.getcwd()))[0]
fuelsvdir = os.path.join(wrkdir,'stored_LMask2')
fuel_retry_ndf = get_fuel_step(fuelsvdir,save_file_name)
```

```
RETRIEVED: this reduction step stored_LMask2/NDF_saved.txt
```

Figure 8. Use of modules `SaveFuel` and `GetFuel` for object serialization.

REFERENCES

Smee, S. A., Barkhouser, R., Harding, A., et al. 2018, Ground-based and Airborne Instrumentation for Astronomy VII, 107021O