# NIRSPEC Data Reduction Pipeline

## *Software Design Document*

*Last update: May 18, 2017*

*R. Cohen, A. Colson*

**Change History**

| | | |
|---|---|---|
| Dec. 15, 2015 | R. Cohen | Initial draft |
| Feb. 28, 2017 | R. Cohen | Added discussion of multi-frame wavelength calibration to section 6.8 and added section 6.9 Background subtraction. |
| May 18, 2017 | A. Colson | Added additional command line options and discussion. |
| | | |

# 1. Introduction

This is the software design document for the NIRSPEC Data Reduction Pipeline (NSDRP) developed at Keck for the Keck Observatory Archive (KOA). It details the overall architecture and functional decomposition and includes descriptions of the main data structures and algorithms.

## 1.1. Purpose

This document provides a description of as-built software. Indented audiences for this document are 1. KOA team members interested in the design, capabilities and limitations of the DRP and 2. KOA software developers who may extend or maintain this software. Enough information is provided to allow a programmer to efficiently navigate the code base.

## 1.2. Scope

Key aspects of the design and implementation are discussed, ranging from top-level program structure to details of the most important low-level algorithms.

A description of the data products produced by the NSDRP are contained in a separate document. It is recommended that it be viewed in conjunction with this document.

## 1.3. Overview

NSDRP is an automatic level-1 data reduction pipeline for NIRSPEC high-resolution object spectra. A grating equation with empirically determined coefficients is used to approximate the position of spectral orders on the detector and their approximate wavelength scale. Flat field images are used to precisely determine actual location of orders on the detector. OH sky lines are used to refine the theoretical wavelength scale.

Data products include flux and spatial profile tables for each order and a table of wavelength calibration data for each object frame. Quick-look plots the spatial profile and flux, sky and noise spectra are produced for each order.

The incarnation of the DRP described herein is the result of refactoring code originally written by Jen Holt. Refactoring was done to improve code readability, reduce unnecessary complexity and improve maintainability. In addition, many improvements and refinements were made to the DRP as part of the refactoring effort.

## 2. System Overview

The central tasks of NSDRP are to locate and extract individual spectral orders from object frames and flats, rectify orders in the spatial and spectral dimensions and combine OH emission sky lines identified in each order to find a global (to the frame) wavelength calibration solution.

Flat field frames taken with the same filter, slit and Echelle and cross disperser angles as the object frame are required for data reduction. Flats are used not only to account for intensity variations caused by variations in pixel-to-pixel sensitivity but also to locate the spectral orders on the detector.

Figure 1 shows a raw object frame of a point source taken with the NIRSPEC-3 (J) filter and a corresponding flat. These images are histogram equalized. Note the many sky lines that uniformly illuminate the slit in the object frame. If dark frames with the same exposure time as the object frame are available they are used in dark frame subtraction to minimize fixed-pattern noise. Figure 2 shows a dark frame.

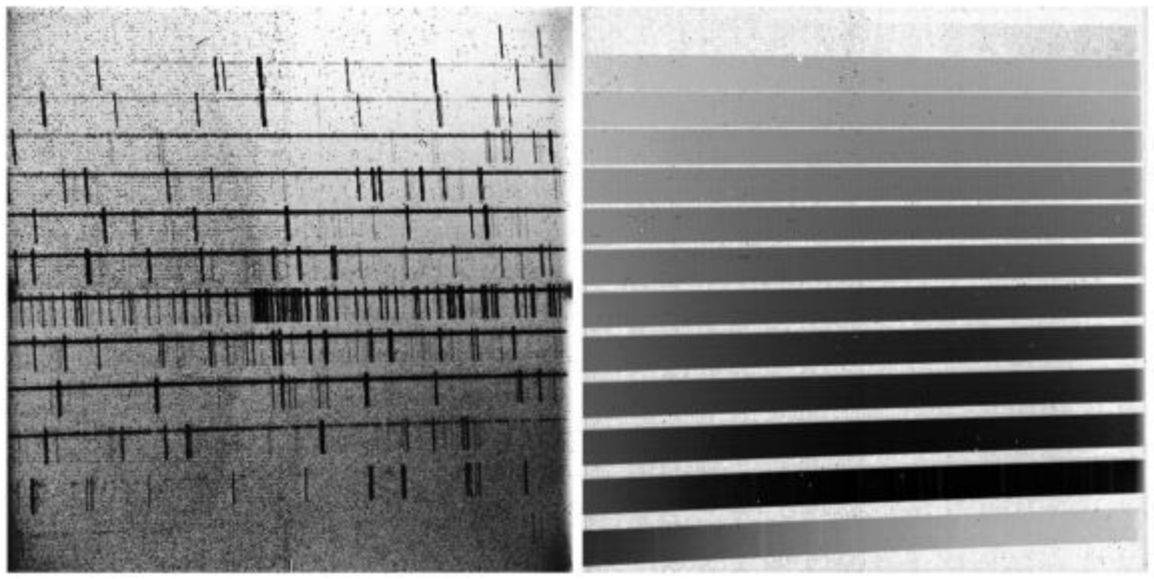

**Figure 1 Object frame and flat field in J.**

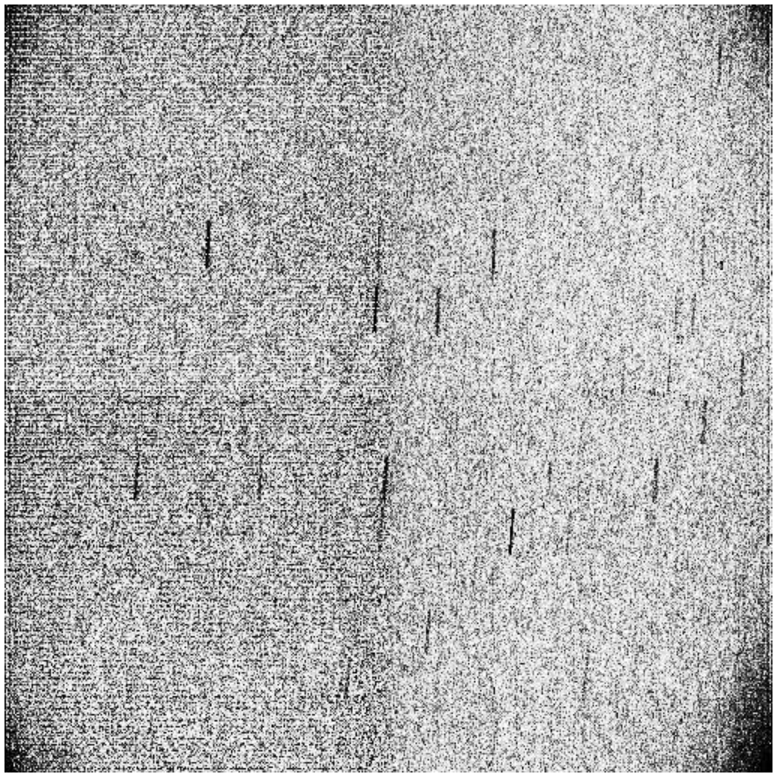

**Figure 2 Dark frame**

Extended objects are not well accommodated by the DRP though the program will produce output if there is a discernable peak in the spatial intensity profile. For successful wavelength calibration, exposures long enough to yield discernable OH sky lines are required. In the absence of sky lines sufficient in number and intensity, wavelength calibration is not possible and wavelength scales are approximated based on the grating equation. Arc lamp frames for wavelength calibration are not supported by the DRP now but this has been identified as an area for future development.

NSDRP can reduce data collected with NIRSPEC filters 1 through 7 and a wide range of Echelle and cross disperser angles and all the high-resolution slits.

The KOA DAQ invokes NSDRP automatically. DAQ supplies the DRP with input and output directory paths as command line arguments. In addition to level 1 data products, two types of log files are produced, one to record per-night summary data and another to record details of data reduction on a per object frame basis.

Several areas for future improvement have been identified. NSDRP does not currently support co-adding, telluric correction or any form of photometric calibration. Since there is not enough information in the FITS headers to make the necessary file associations, some mechanism for associating related files will be needed to implement these features.

NSDRP uses conventional extraction but the noise calculations already performed could be used to implement optimal extraction to maximize signal-to-noise ratio while preserving spectrophotometric accuracy.

Other areas for improvement are dynamic adjustment of extraction window widths and separations, improvements to order edge detection algorithm to handle step inversions on overlapping orders, optimization of empirically determined grating equation coefficients.

## 3. System Architecture

NSDRP is a procedural program that runs in a single thread of execution. The simple, serial nature of a data reduction pipeline is naturally represented in the procedural paradigm.

The NSDRP is written entirely in the Python programming language and makes extensive use of common libraries including numpy, astropy, scipy and matplotlib. The software is written mainly in the procedural style though the main application-specific data structures are implemented as Python classes. There are approximately 4,000 lines of code in 21 modules (files). The functions in each module are related primarily by their level of abstraction. For example, the module nirpsec_drp.py, which contains the function nsdrp(), represents the NSDRP as a whole, and is at the highest level of abstraction. Functions that perform low-level image processing not specific to NIRSPEC, such as image normalization and array shifting for rectification, are at the lowest level of abstraction and are collected in the module image_lib.py.

Although the overall design of the NSDRP does not use the object-oriented paradigm, four central application-specific data structures used in the program are implemented as Python classes. This design choice simplifies construction and management of multiple simultaneous instances of these classes. The four central classes are:

1. RawDataSet – Represents an associated set of raw image files—object frames, darks and flats.
2. ReducedDataSet – Represents the results of an object frame reduction. Has fields representing reduction results pertaining to the frame as well as a list of Order objects that represent reduction results for individual orders.
3. Order – Represents reduction results for an individual order. Has fields representing reduction parameters pertaining to the order and a list of Line objects representing identified sky lines.
4. Line – Represents an observed sky line that has been associated with an OH emission line of known wavelength.

With a few exceptions made for convenience, the four classes do not have class or instance methods. However, functions closely associated with a given class are collected in a single module. For example, reduce_order.py contains functions that operate on Order objects. As a general rule, low-level functions do not access higher-level data structures.

Low-level functions are largely implemented using the functional style of programming in the sense that they tend to behave like mathematical functions. These functions are highly decoupled from the environment in which they are used, do not contain any state information and produce results as function return values rather than by modifying mutable data structures.

When reading the code, it is helpful to be familiar with the detector coordinate system conventions used. The detector used in the spectrograph is 1024x1024 pixels. Wavelength increases with column number. Images are usually displayed with column number and wavelength increasing in the positive x direction. The terms 'top' and 'bottom' are frequently used in the code. Row number zero is at the 'bottom' of the frame and row 1023 is at the 'top'. Images are usually displayed with row number increasing in the negative y direction.

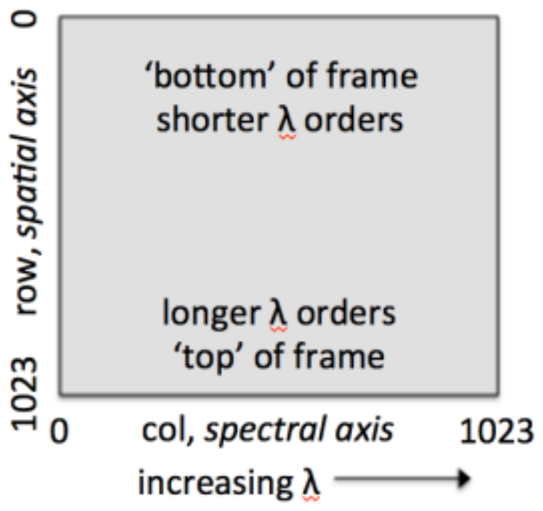

**Figure 3 Row and column numbering convention**

## 3.1. Architectural design

**Context**

NSDRP is designed to be called by KOA DQA and to perform its processing autonomously with no operator intervention. Raw FITS image files are read from a source directory, a path to which is passed to the DRP on the command line. NSDRP automatically determines which files in the source directory can be reduced by the NSDRP. Darks, if any, and flats are expected to be found in the same source directory. The NSDRP associates calibration files with object files based on values of certain keywords in the FITS header.

In addition to object files, flats and optionally dark frames, the only other input data required by the DRP is a file containing an atlas of OH emission lines in the near infrared (Rousselot et al).

Data products are generated from the reduction results which include ASCII tables, binary FITS tables, FITS images and various preview plots. A root output directory path is passed to the NSDRP on the command line and data products are stored in a directory structure subordinate to the root output directory and defined in the data products document. NSDRP also writes log files to the output directory.

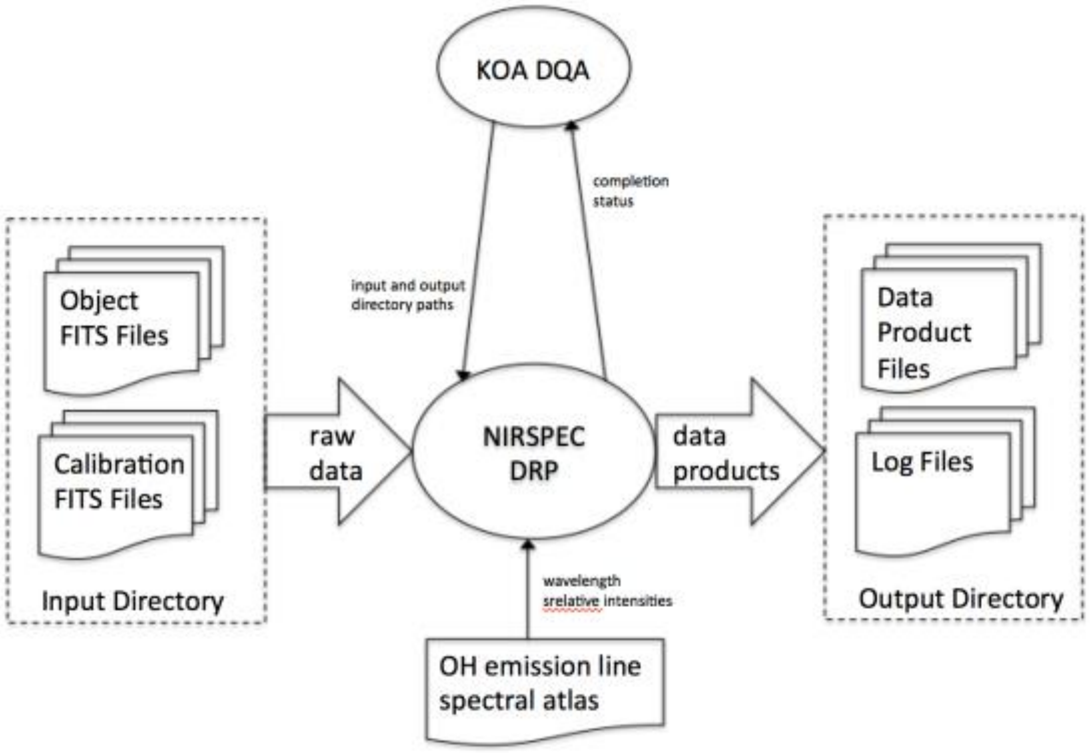

**Figure 4 System Context Diagram**

**Top-level program structure**

The top-level program structure is summarized in the following pseudo code:

```
nsdrp_drp:
    construct raw data sets
    for each raw data set:
        for each spectral order in the frame:
            locate and extract order
            reduce order
            identify sky lines
        find and apply wavelength solution
        generate data products
```

The top-level function of the NSDRP is nsdrp.nsdrp(). The NSDRP first scans the source directory and identifies object frames that can be reduced. For each object frame that can be reduced, create_raw_data_set.create() associates object frames with corresponding calibration frames and constructs a series of RawDataSet objects. Each raw data set consists of an object frame, one or more flat field frames and zero or more darks.

An outer loop iterates through each raw data set and calls reduce_frame.reduce_frame() to reduce each object frame. A ReducedDataSet object is instantiated by reduce_frame() and all reduction results for the frame are stored in this object. If there are multiple darks or flats they are median combined. Combined darks are subtracted from object and combined flat. Cosmic ray artifacts are removed using a Laplacian edge detection algorithm.

The grating equation is used to determine which orders are expected to be on the detector. For each order expected to be on the detector, reduce_order.reduce_order() is called to instantiate an Order object. Order processing involves cutting out a rectangular region containing the order from the full frame. Rectification in the spatial dimension is accomplished by tracing order edges in flat frames, and in the spectral dimension by tracing sky lines along the slit. The rectified order is collapsed along the spectral axis to compute the mean spatial profile along the slit. Order reduction continues with spectral extraction and sky line identification. Spectra are extracted using fixed size extraction windows. Background subtracted object, sky and estimated noise spectra are produced. All reduction products for the order are stored in the Order object.

When all orders have been reduced and sky lines identified, reduce_frame.find_global_wavelength_soln() is called to find coefficients to the wavelength equation. Calibration lines from all the orders are used in a 2-d fit of wavelength as a function of order, column location of order to known wavelengths.

Finally, products.gen()is called to generate the required data products from the reduced data.

Processing continues with the next object frame until all reducible object frames in the input directory have been processed. Figure 5 shows the top-level data flow for the DRP.

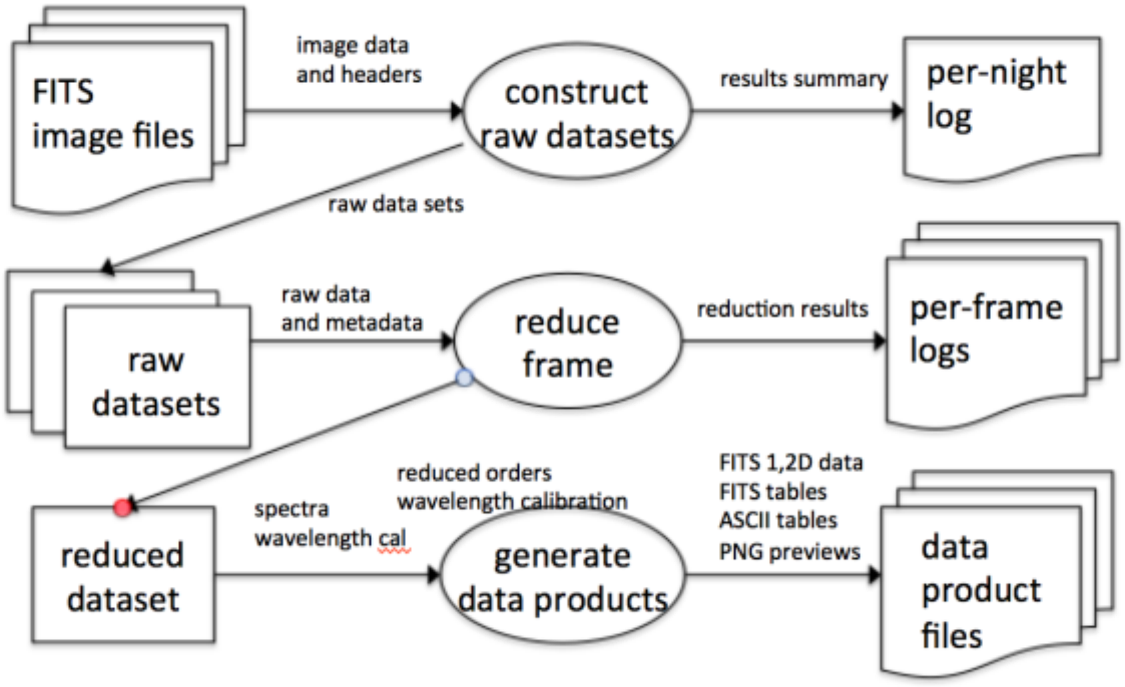

**Figure 5 Top-level data flow**

## 3.2. Error handling

Since the NSDRP is intended to run automatically, without direct operator intervention, it is important that the program is capable of detecting, reporting and recovering from most error conditions.

Low level code does not write to log files and in most cases, does not throw exceptions. When a return value cannot be computed by a low-level function, it returns the Python type None which represents the absence of a value.

Middle layer processing attempts to handle errors originating in the low-level libraries. For example, if one order edge can be traced by the low level library functions but the other edge cannot be traced, mid-level processing will approximate the location of the edge that could not be traced from the edge that was traced and height of the order predicted from the grating equation. Middle layer code is aware of loggers and all error conditions and corrective actions are logged.

If a low-level error cannot be handled by mid-level routines then an exception is thrown to signal the error to procedures higher in the call hierarchy.

Higher-level functions catch exceptions thrown by middle layer processing. These functions are aware of the loggers so it is at this level that most errors are logged. If a high-level routine determines that an error is critical and processing cannot continue, it is reported as critical in the main log file and the DRP terminates with exit code 1. Examples of critical errors are when the input or output directory cannot be accessed, or the file containing the OH emission line atlas cannot be opened. A critical error means that an operator must take action.

Most errors, such as when an order cannot be located or rectified or when insufficient sky lines have been identified to find a wavelength solution, are reported and the program continues with the next order or the next object frame. Normally, the DRP terminates with exit code 0, the standard Unix return value on success.

Successful DRP completion status does not necessarily mean that reduction was successful. To determine the success and quality of the reduction, the log files must be examined manually.

## 4. Data Design

Four application-specific data structures are defined. These data structures are implemented as Python classes even though the NSDRP does not use a true object-oriented design. The classes defined have instance variables but they do not have class variables or and with very few exceptions the classes do not have method members.

All data members are public. Getters and setters are generally not defined; in most cases member fields are accessed directly.

Each class is defined in its own Python module of the same names as the class, e.g. RawDataSet.py.

### 4.1. RawDataSet

Class representing a raw data set consisting of an object file name, the object file header and file name lists for flats and darks. Note that instances of this class contain references to data file

**Table 1 RawDataSet class field summary**

| **Name** | **Type** | **Description** |
|---|---|---|

| | | |
|---|---|---|
| objFileName | string | File name of object file. |
| objHeader | astropy.io.fits.header | Object file FITS header. |
| flatFileNames | list of strings | File names of flat field files. |
| darkFileNames | list of strings | File names of dark frame files. |

## 4.2. ReducedDataSet

Represents a reduced data set. Among its members is a list of Order objects representing the results of per-order data reduction.

**Table 2 ReducedDataSet class field summary**

| **Name** | **Type** | **Description** |
|---|---|---|
| fileName | string | File name of object file. |
| baseName | string | Base file name used as base of data product file names. |
| header | astropy.io.fits.header | FITS header from object frame including keywords added by the DRP. |
| flatKOAIds | string list | List of flat file names. |
| hasDark | boolean | True if one or more dark frames are associated with this data set. |
| darkKOAIds | string list | List of dark file names |
| darkSubtracted | boolean | True after (master) dark subtracted from object frame and (master) flat. |
| cosmicCleaned | boolean | True if cosmic ray rejection processing has occurred. |
| obj | 2-d ndarray float64 | Object frame data. |
| flat | 2-d ndarray float64 | Flat or master flat data. |
| dark | 2-d ndarray float64 | Dark or master dark data. |
| orders | list of order objects | Per-order reduction results. |
| coeffs | 1-d ndarray float64 | File names of dark frame files. |
| rmsFitRes | float | RMS wavelength fit residual. |

## 4.3. Order.py

Represents a spectral order and contains data results from all stages of order reduction. Among its members are extracted spectra and wavelength calibration line data.

**Table 3 Order class field summary**

| Name | Type | Description |
|---|---|---|
| orderNum | int | Spectral order number. |
| topCalc | float64 | Calculated pixel row of top of zeroth column of order. |
| topMeas | float64 | Measured pixel row of top of zeroth column of order. |
| botCalc | float64 | Calculated pixel row number of bottom of zeroth column of order. |
| botMeas | float64 | Measured pixel row number of bottom of zeroth column of order. |
| wavelengthScaleCalc | 1-d ndarray of float64 | Calculated wavelength scale of order. |
| wavelengthScaleMeas | 1-d ndarray of float64 | Wavelength scale of order based on wavelength calibration. |
| wavelength_shift | float | Shift between calculated wavelength scale and synthesized sky. |
| lines | list of Order objects | Array of Line object representing sky lines identified for wavelength calibration. |
| padding | int | Number of padding rows above and below order in cutout. |
| perOrderCal | Boolean | True if per-order wavelength calibration used. |
| perOrderSlope | float | Slope of linear per-order wavelength fit. |
| perOrderIntercept | float | y-axis (measured wavelength) of linear per-order wavelength fit. |
| perOrderCorrCoeff | float | Correlation coefficient of per-order linear |

| | | |
|---|---|---|
| | | wavelength fit. |
| objCutout | 2-d ndarry of float64 | Raw object frame cutout. |
| flatCutout | 2-d ndarry of float64 | Raw flat frame cutout. |
| onOrderMask | 2-d ndarry of booleans | Elements corresponding to pixels inside the order are True. |
| offOrderMask | 2-d ndarry of booleans | Elements corresponding to pixels outside the order (in the padding) are True. |
| topTrace | 1-d ndarray of float64 | Trace of top edge of order. |
| botTrace | 1-d ndarray of float64 | Trace of bottom edge of order. |
| avgTrace | 1-d ndarray of float64 | Average of top and bottom traces. |
| smoothedTrace | 1-d ndarray of float64 | Smoothed average trace. |
| traceMask | 1-d ndarray of booleans | Elements corresponding to points ignored in trace smoothing are False. |
| spectralTrace | 1-d ndarray of float64 | Smoothed trace of sky lines. |
| spatialRectified | boolean | True if 2-d order arrays are rectified in spatial dimension. |
| spectralRectified | boolean | True if 2-d order arrays are rectified in spectral dimension. |
| flatMean | float64 | Normalization scale factor. |
| flatImg | 2-d ndarray of float64 | Flat order image data. |
| normalizedFlatImg | 2-d ndarray of float64 | Normalized flat order image data. |
| objImg | 2-d ndarray of float64 | Object order image data. |
| flattenedObjImg | 2-d ndarray of float64 | Flattened object order image data. |
| noiseImg | 2-d ndarray of float64 | Calculated noise image data. |
| spatialProfile | 1-d ndarray of float64 | Spatial profile of order collapsed in spectral dimension. |
| peakLocation | int | Row location of peak in spatial profile. |
| centroid | float | Fractional row location of spatial profile peak. |

| | | |
|---|---|---|
| objWindow | list of ints | List of pixel row numbers used for object spectrum extraction. |
| gaussianParams | list of floats | Gaussian fit parameters of profile peak. |
| topSkyWindow | list of ints | List of pixel row numbers used for top background extraction. |
| botSkyWindow | list of ints | List of pixel row numbers used for bottom background extraction. |
| objSpec | 1-d ndarray of float64 | Extracted object spectrum. |
| noiseSpec | 1-d ndarray of float64 | Extracted noise spectrum. |
| skySpec | 1-d ndarray of float64 | Extracted sky spectrum. |
| synthesizedSkySpec | 1-d ndarray of float64 | Synthesized sky spectrum. |
| topBgMean | float | Mean intensity in top background window |
| botBgMean | float | Mean intensity in bottom background window |
| snr | float | Signal-to-noise ratio. |

## 4.4. Line.py

Represents an identified emission line used for wavelength calibration. Among its member fields are the order in which the line was found, fractional column number of line centroid, accepted wavelength of line, fit wavelength.

**Table 4 Line object field summary**

| **Name** | **Type** | **Description** |
|---|---|---|
| col | int | Integer column number of observed line |
| centroid | float | Fractional pixel location of line centroid |
| order | int | Spectral order in which the line was located |
| acceptedWavelength | float64 | Accepted wavelength of the line |
| fitWavelength | float64 | Wavelength of the line based on global wavelength solution |
| globalFitResidual | float64 | Difference between |

|  |  |  |
|---|---|---|
|  |  | global fit wavelength and accepted wavelength. |
| peak | float64 | intensity in the peak column of the line averaged along the slit |
| globalFItSlope | float64 | Differential of the global wavelength solution at this wavelength |
| usedInGlobalFit | boolean | True if this line was used in the global wavelength fit |
| localFitWavelength | float | Wavelength of the line based on per-order linear fit |
| localFitResidual | float | Difference between local fit wavelength and accepted wavelength. |
| localFitSlope | float | Slope of the local wavelength solution |

## 5. Components

In this section, the decomposition of the program into Python modules (source files) is presented.

For more detailed information about each function, see the function docstring in the code that summarizes function behavior, arguments, return values and exceptions raised.

The modules are divided into four categories: high-level processing, libraries, core data reduction processing, utilities and libraries. Not included in this section are the class definition modules that are described in the data design section.

### 5.1. High-level processing modules

**nsdrp.py**

The module contains the main entry point of the NSDRP. The main routine sets up and runs the command line parser, calls an initialization routine to confirm access to the input and output directories and to initialize the loggers. The function nsdrp() is then called which implements the NSDRP top level processing.

**create_raw_data_sets**

This module is concerned with associating calibration files with object files and constructing raw data sets in the form of RawDataSet objects. The central function of this module is create() which takes an input directory path and returns a list of RawDataSet objects.

## 5.2. Core data reduction modules

**extract_order**

The functions in this module are concerned with extracting rectangular regions containing individual orders from the full image frame.  This module contains NIRSPEC-specific extraction tuning parameters and it makes use of the Order class. Note that code for spectral extraction is not contained in this module but rather in image_lib.

**reduce_frame**

This module contains the main object frame reduction function reduce_frame.reduce_frame and is closely associated with the ReducedDataSet class. An initialization routine reduce_frame.init() sets up the per-object logger. reduce_frame.reduce_orders() determines which orders should be on the detector and calls reduce_order.reduce_order to reduce them. The results of order reduction are stored in a list in the ReducedDataSet object. Utility functions are called to find and apply a global wavelength solution.

**reduce_order**

This module contains the main order object reduction function:reduce_order.reduce_order() and is closely associated with the Order class. Utility and library functions are called to normalize the flat, flatten the object image, rectify the order and find and identify sky lines.

## 5.3. Utility modules

**config.py**

Configuration parameters are stored in a dictionary data structure named params defined in this module.

**cosmics.py**

This module was written by Molte Tewes and implements the Laplacian cosmic ray detection algorithm

**grating_eq.py**

This module contains the function evaluate() which finds the order location on the detector and the approximate wavelength scale for an order, given the filter, slit, Echelle angle, cross disperser angle and date. Coefficients to the grating were determined empirically. An empirically determined set of “corrections” for filter, slit, order number and date are applied to the values returned by the grating equation. In the future, grating equation coefficients should be recomputed and/or terms should be added to the grating equation to reduce or eliminate the need for making these corrections.

#### products.py

All data products are generated in this module. At the completion of frame processing, the ReducedDataSet object is passed to the function products.gen(). All the data required to produce the data products is contained in the ReducedDataSet object.

#### tracer.py

This module contains the general-purpose line tracing algorithm tracer.trace_edge(). Library function in nirspec_lib() configure this function to operate on order edges or sky lines.

#### wavelength_utilities.py

Utility function for calculating the global wavelength solution are included in this module.

### 5.4. Libraries

#### image_lib.py

This module contains image processing routines that do not depend on NIRSPEC-specific parameters and use only built-in data structures and arrays. Routines for order rectification and spectral extraction are included in this module.

#### nirspec_lib.py

This library contains image processing that depend on NIRSPEC-specific parameters but use only built-in data types and arrays.

## 6. Algorithms

### 6.1. File association

Each object frame must be associated with at least one flat since flats are required to locate order edges. Darks are used if they are available but are not required. No other calibration images (e.g. arc lamp images, telluric calibration stars, etc.) are used by the NSDRP.

The file association algorithm operates on files in the input. Subdirectory traversal is not attempted so only files in the input directory itself, not its subdirectories, are used. The same calibration files may be associated with multiple object files and this is a common case.

Each object file found in the input directory is treated independently. Only object frames taken with NIRSPEC in high-resolution mode as indicate by the DISPERS keyword (='high') can be reduced by the NSDRP. Object frames taken with the instrument in low-resolution mode are logged in with NIRSPEC configured in the instrument in low-resolution mode are logged in the summary log file but otherwise ignored.

Association between object files and flat and dark frames is based on the values of FITS keywords. First, IMAGETYP is used to determine file type: 'object' for object frames, 'flatlamp' for flats and 'dark' for dark frames. For each object file, all the matching flat frames are collected. Matching flats are those for which the following keywords have the same value as the corresponding keywords in the object frame:

- 'disppos' for cross disperser angle,
- 'echlpos' for Echelle angle,
- 'filname' for filter in the beam and
- 'slitname' for slit in use.

If all keywords match then the flat is associated with the object. At least one flat must be found for reduction to succeed. If no flats can be associated with the object frame then the condition is logged in the summary log file and no further processing of the object frame is attempted.

If multiple flats are found, as is typically the case, they are combined at a later stage in the pipeline by taking the median value across frames for each pixel to form a master flat.

Dark frames are matched with object frames based on exposure time as indicated by the 'elaptime' keyword. If multiple darks are associated with the object frame they are median combined later in the pipeline to form a master dark frame. Dark frames are not required by the DRP to complete processing. In the absence of dark frames, fixed-pattern noise caused by dark current, defective pixels (hot or dark) and gain variations in detector quadrant amplifiers will contribute more to overall noise.

The results of file association are represented by instances of the RawDataSet class which has fields for object, flat and dark file names and headers. File association is performed by create_raw_data_sets.create().

## 6.2. Cosmic ray cleaning

Images are analyzed to identify artifacts caused by cosmic ray hits on the detector during exposure. Comic ray rejection is performed on object frames after flat-fielding but before any other image processing.

Cosmic ray artifacts (and some types of electronic noise and sensor defects) are characterized by intensity discontinuities or sharp-edges. NSDRP uses an algorithm

for cosmic ray rejection called L.A. Cosmic based on Laplacian edge detection developed by Dokkum at Caltech (Cosmic-Ray Rejection by Laplacian Edge Detection) and implemented in Python by Malte Tewes (2010).
Cosmic ray cleaning is processing time-intensive; about one third of NSDRP processing time is spent on cosmic ray reduction. The remaining two-thirds is split about equally between image processing and data product generation.

Cosmic ray cleaning is done by image_lib.cosmic_clean() which instantiates and initializes a cosmic.cosmicsImage() object, calls its run() method and then returns the cleanarray field. Figure 6 shows an order image before cosmic ray cleaning and figure 7 shows the same image after cleaning.

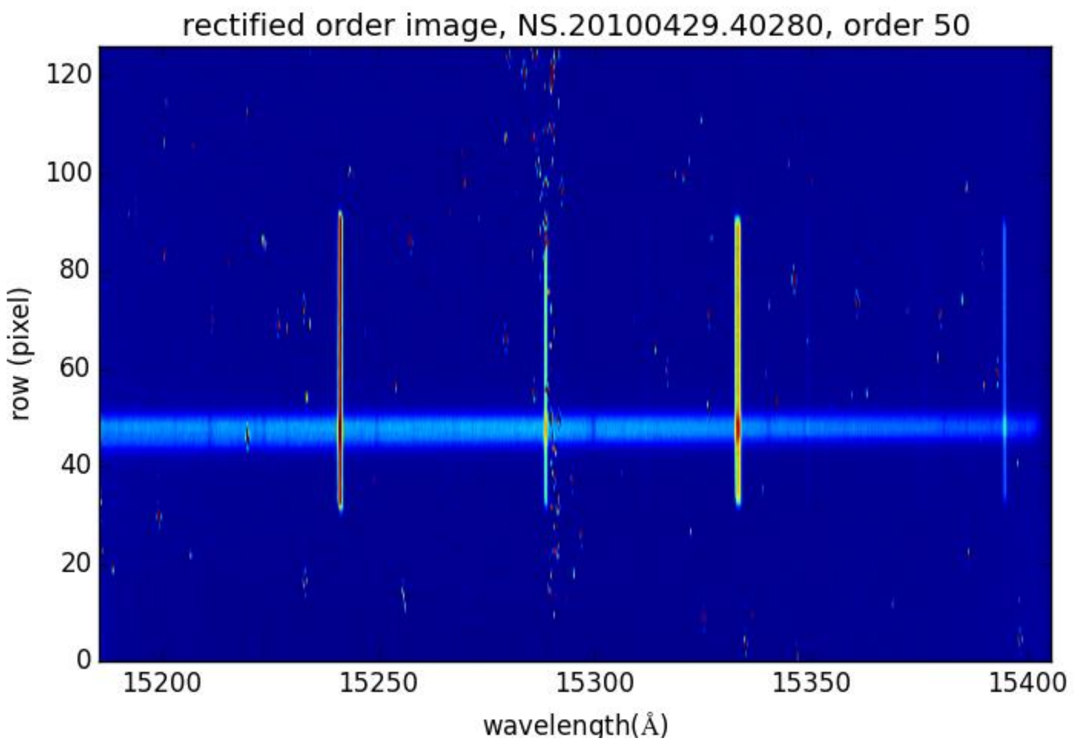

**Figure 6 Flat-fielded, rectified order with no cosmic ray cleaning.**

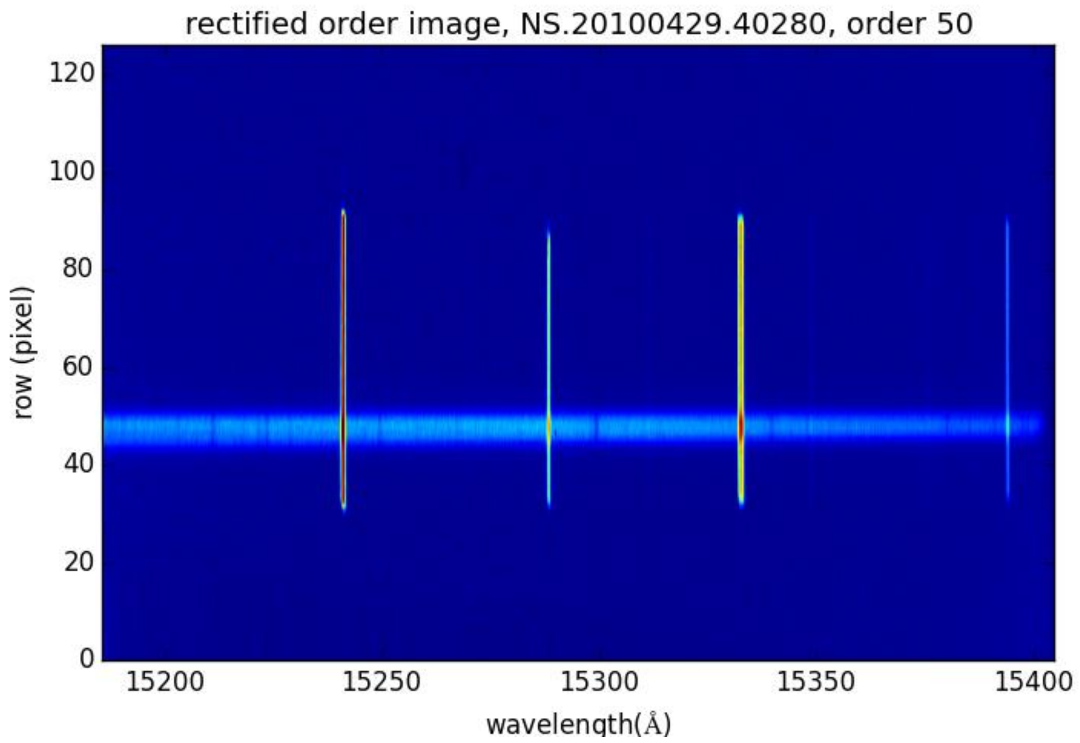

**Figure 7 The same flat-fielded, rectified order shown in the previous figure but with cosmic ray cleaning.**

## 6.3. Grating equation

The expected central wavelength falling on pixel column c is given by

$$\lambda(c,\theta,\phi,M) = \frac{k_1 \sin(\theta) + k_2 \left(\frac{w}{2} - c\right) cos(\phi)}{M}$$

where θ is Echelle angle, ϕ is the cross disperser angle and M is order number. $k_1$ and $k_2$ are constants the values of which have been determined empirically and w is the width of the detector in pixels along the dispersion axis.

Successive orders are stacked vertically (i.e. along the spatial axis) of the detector. The predicted location of the spatial center of an order in the zeroth pixel (shortest wavelength) column of the detector is given by

$$y_{center} = c_{f,0} + c_{f,1}\lambda_0 + c_{f,2}\sin(\phi)$$

where $c_{f,0}$, $c_{f,1}$ and $c_{f,2}$ are constants determined empirically for each filter and $\lambda_0$ is the wavelength of the zeroth pixel column.

Corrections are applied to the expected wavelengths by shifting the wavelength scale by an order-dependent constant value. Corrections are also applied to calculated order positions by adding slit- and filter-dependent constants. A change to the physical configuration of the detector or instrument occurred on May 24, 2004 so a date-dependent correction is also applied to calculated order positions.

It may be possible to improve the accuracy of the grating equation and avoid the many corrections that must be applied. Once a significant number of object frames have been reduced in a wide range of configurations, it may be possible to fit that data to the grating equation and find a better set of coefficients, or it may be necessary to add terms to the grating equation.

## 6.4. Order location, edge tracing and cutout

Order location and tracing proceeds in three successively more accurate steps:

1. Approximate location along the y-axis of the detector is determined by evaluation of the grating equation.
2. Actual order location is measured at the left (short wavelength, low column number) edge of the detector using the approximate positions predicted by the grating equation as a starting point and then using an edge detection algorithm to locate order edges along the left (low column numbers, short wavelength) edge of the detector.
3. A one-dimensional centroiding algorithm is used to trace the top and bottom edges of the order along the full spectral extent of the order. Measured order locations in the previous step are used as the starting point for centroiding.

Evaluation of the grating equation as well as its form and origin are discussed elsewhere. Order location estimated with the grating equation can differ from the actual measured order location by as much as 10 to 20 pixels so the grating equation alone cannot be used to locate orders.

Simple order width calculation computes order width from slit length (24 or 12 arc seconds) and plate scale (0.193 pixels/arc second).

Initial edge detection and edge tracing (steps 2 and 3) depend on two images derived from flats in which order edges are transformed from wide steps to narrow peaks.

Figure 8 shows flat field image where the bright areas are inside the orders and dark areas are between orders.

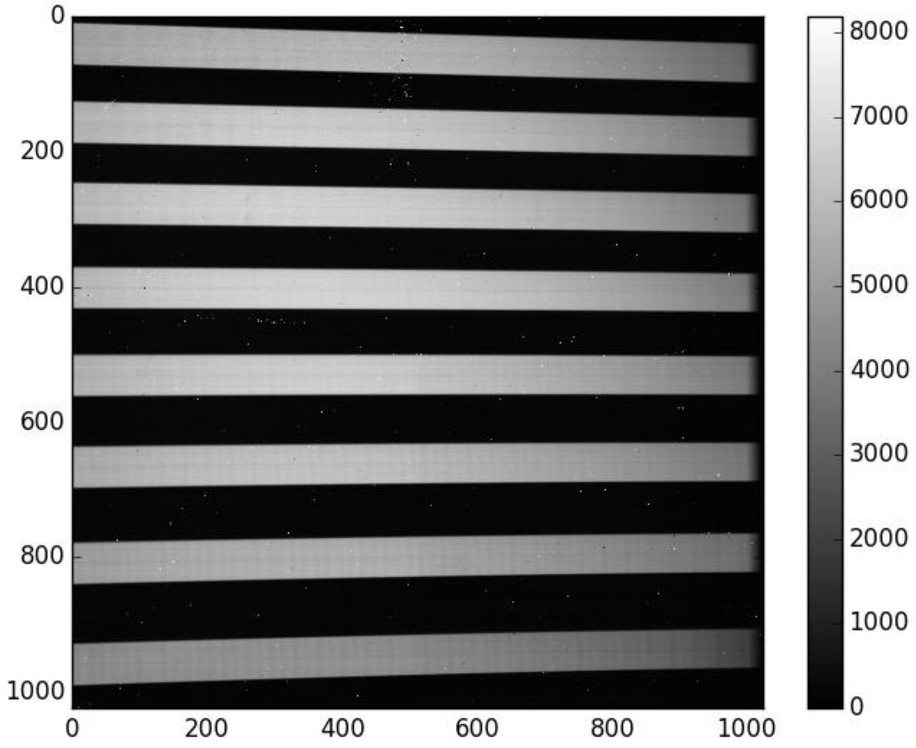

**Figure 8 Flat field image**

Figure 9 is a cross section of the image shown in Figure 8 near the left edge of the detector. The broad peaks correspond to the spectral orders on the detector and the valleys are the unexposed are between orders with steep steps between.

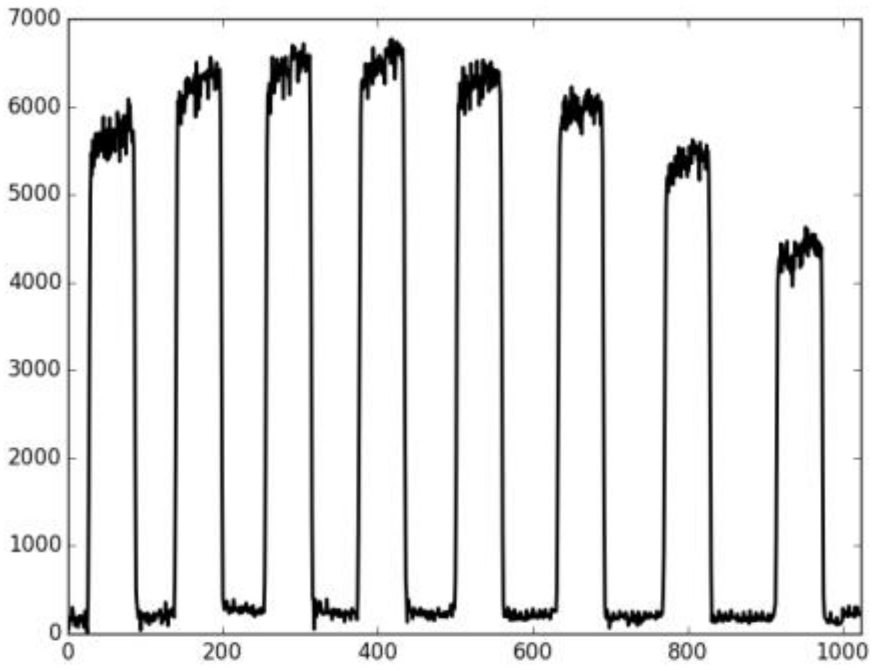

**Figure 9 Cross section of flat field image**

If the flat image is shifted in the +x direction and then the shifted image is subtracted from the original flat image, most of the image structure is cancelled out except for intensity peaks along the top order edges and negative peaks along the bottom edges. If the shift is in the negative then the pattern is reversed and the bottom order edges appear as peaks. These peaks indicate the location of the order edges at the column or column range of the crosscut.

The result of order shift and subtraction is shown in figure 9. In these images, red lines correspond to positive intensity peaks and blue lines to negative peaks.

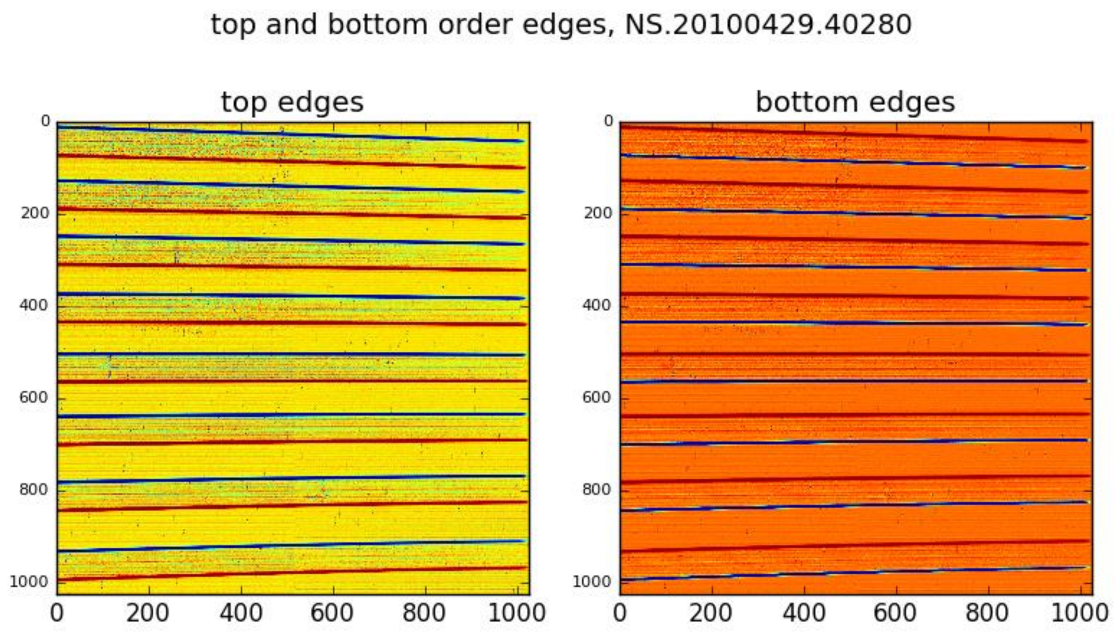

**Figure 10  Top and bottom order edge images**

Figure 11 shows a cross section along the y-axis of the top edge difference image shown in Figure 10; the peaks correspond to order top edge location in the column range of the cross section.

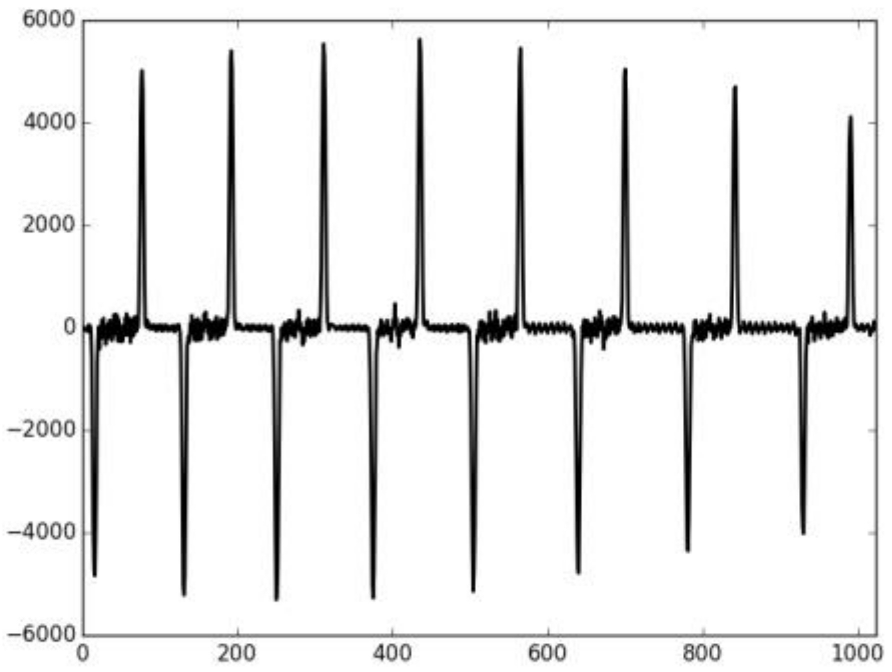

**Figure 11 Order top edge intensity cross section**

By shifting the flat in the other direction, intensity peaks along the inside of the bottom order edges are created and the result is shown in figure 12.

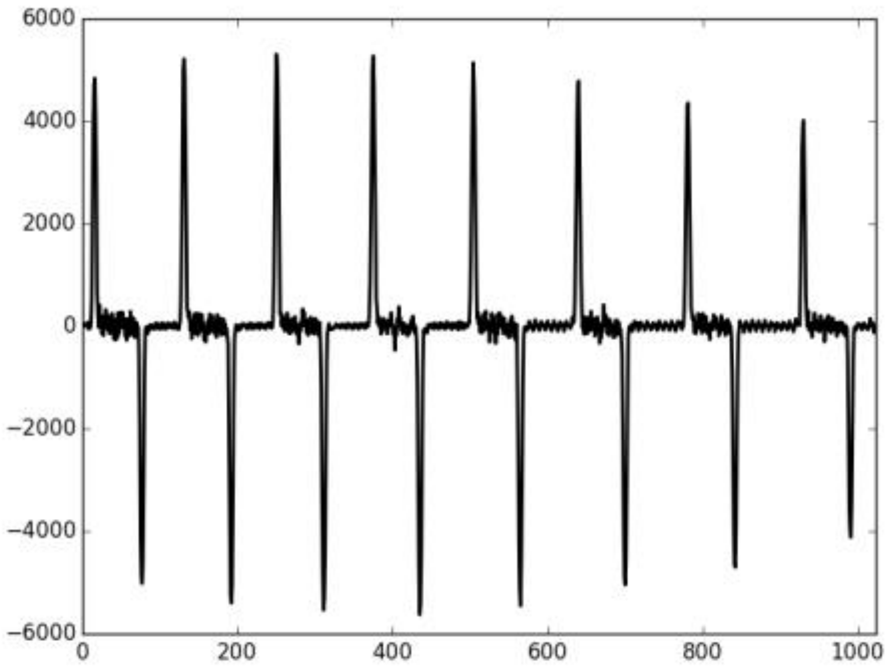

**Figure 12 Order bottom edge intensity cross section**

Peak locations are determined by finding relative extrema in the cross section using the argrelextrema() function of the scipy signal processing library. Peaks less than half the median peak intensity are ignored to filter out spurious edge detection caused by noise in low signal-to-noise images.

Top and bottom order edge locations in the cross section are used as the starting point for tracing the entire extent of the order. Edge tracing is based on the center_of_mass() function in the scipy.ndimage.measurements library where pixels are analogous to particles of mass, pixel intensity is particle 'mass' and pixel number is the one-dimensional coordinate of the particle.

The centroiding calculation is performed column by column to trace the edge across the detector. A simple recursive low-pass smoothing filter is implemented. If a measured point along the edge differs from the previous point by more than a preset threshold, then the last value in the filter is used for that point. A record is kept of the number of edge 'jumps' detected and if this number exceeds a preset threshold then edge tracing fails.

Figure 13 shows top and bottom edge traces for a single order (fourth from the top of the image) on the flat field image. Since the spectrometer was in the same configuration when the flat field frame was collected as it was when the object frame was collected, the same trace mapped onto the object frame delineates the spectral order in the object frame.

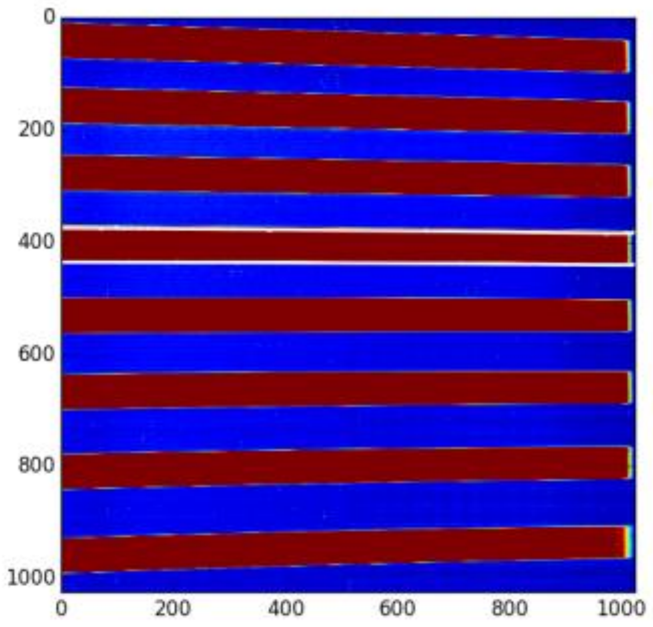

**Figure 13 Order edge traces on flat field frame**

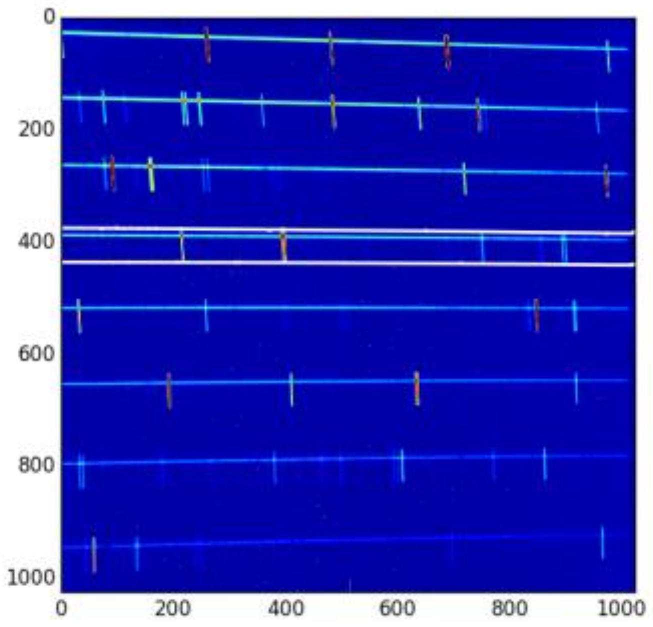

**Figure 14 Order edge traces on object frame**

The raw top and bottom traces are used to determine the boundaries of the order cutout. Each order is contained in the rectangular area defined by the lowest point of the bottom trace, the highest point of the top trace and the detector edges. For steeply sloped orders, the rectangular area may include portions of adjacent orders. To eliminate overlapping traces, the order cutout is trimmed to contain only the central order after spatial rectification.

For spatial rectification, the average of the top and bottom traces is used. The average trace is smoothed by fitting the raw average trace to a third order polynomial and discarding outlying points as shown in Figure 15.

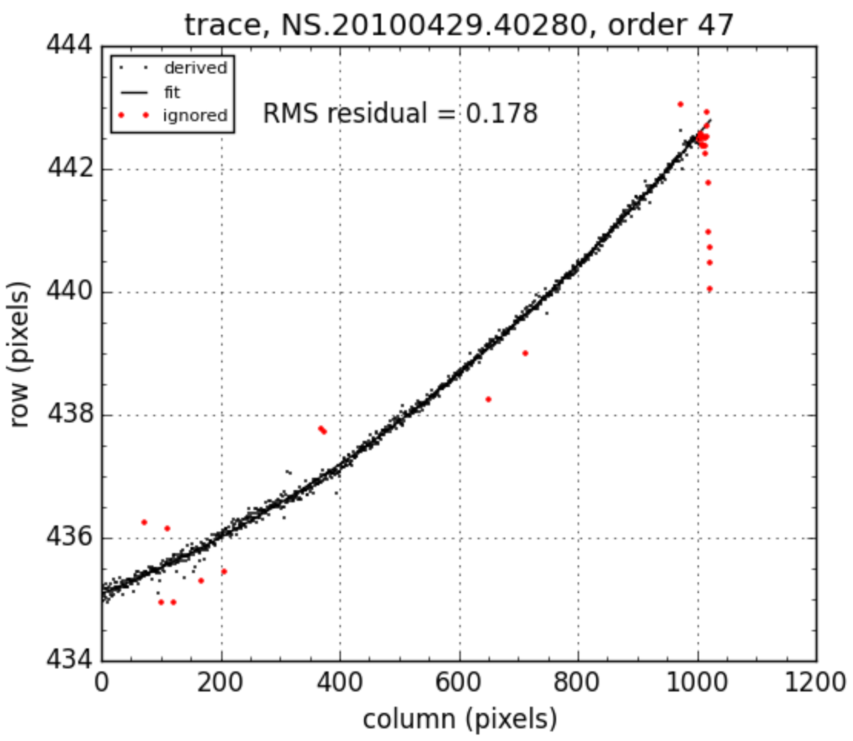

**Figure 15 Raw and smoothed spectral trace**

In the figure, the individual data points represent the raw average trace and the continuous line shows the result of the polynomial fit. Smoothing is accomplished in two steps. First the raw average trace is fit to a third order polynomial. Points with fit residuals greater than a threshold value are considered outliers and discarded. Then the remaining points are fit again to a third order polynomial.

Code having to do with order location and cutout is in the module extract_order.py. Spatial rectification is done by code in the reduce_order.py module.

## 6.5. Spectral rectification

Ideally, each column of pixels in an order would cover the same wavelength increment. In fact, there is a shear in the spectral dimension due to instrument geometry and each row of pixels is laterally shifted in wavelength from the adjacent row. If left uncorrected, this spectral shear would result in a broadening of spectral features.

Sky lines (OH rotation-vibration emission lines) illuminate the entire slit evenly. In the absence spectral shear, sky lines would appear at the same column location in every row of the spatially rectified order. In practice, sky lines appear as slightly tilted lines. By measuring the tilt angle of the sky lines, a corrective shift can be applied to each row of pixels such that in the resulting frame, all pixels in a column correspond more closely to the same wavelength increment.

Sky lines are first located by summing the intensity of each column in a slice of the order image far from the continuum which results in an approximate sky spectrum as shown in Figure 16.

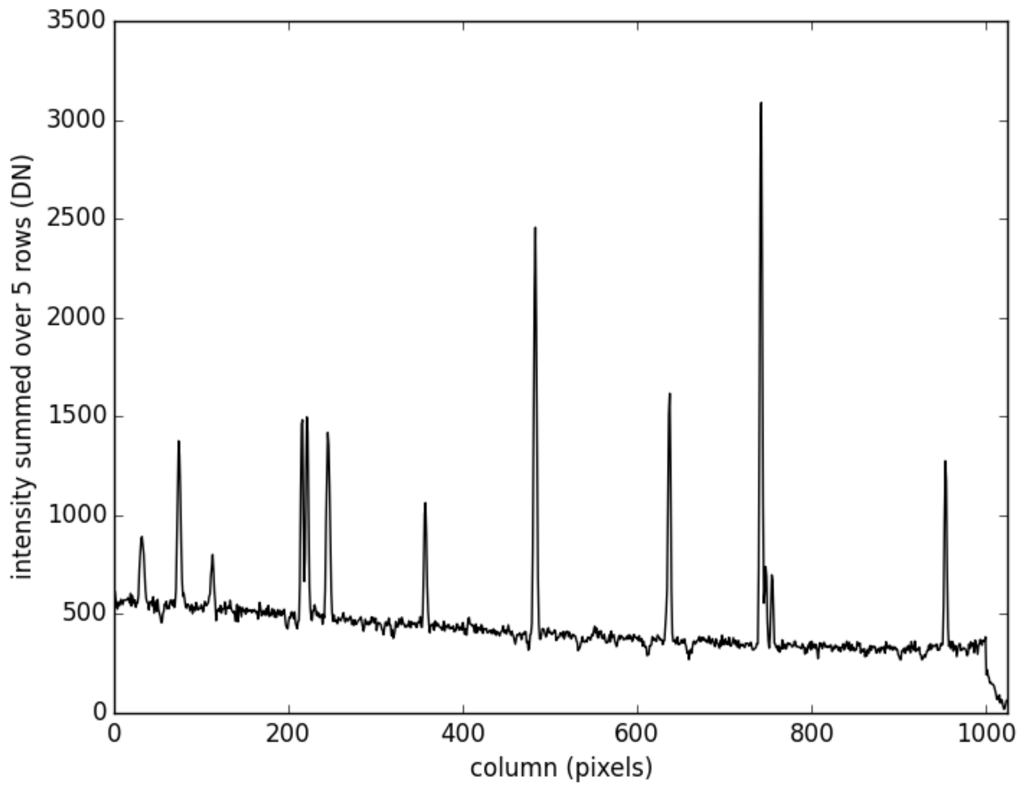

**Figure 16 Approximate sky spectrum used to locate bright sky lines.**

Closely spaced peaks are ignored as are peaks below a threshold intensity level. Starting with the brightest line, the line in the order image is traced along the spatial dimension using the center of mass algorithm described above. If tracing fails then the next brightest line is traced. This continues until three bright lines have been traced or no more lines are available. If three good lines were found, the trace of these three lines is shifted along the spectral axis so they have a common origin, averaged and then fit to a line. Spectral shift is assumed to be approximately linear. The slope of this line is the spectral tilt and is used to rectify the order in the spectral dimension.

Figure 17 shows a spectral order before (top) and after (bottom) spectral rectification. In this case, the slant is about 0.093 column pixels per row pixel, or about 0.9 Å from one end of the slit to the other.

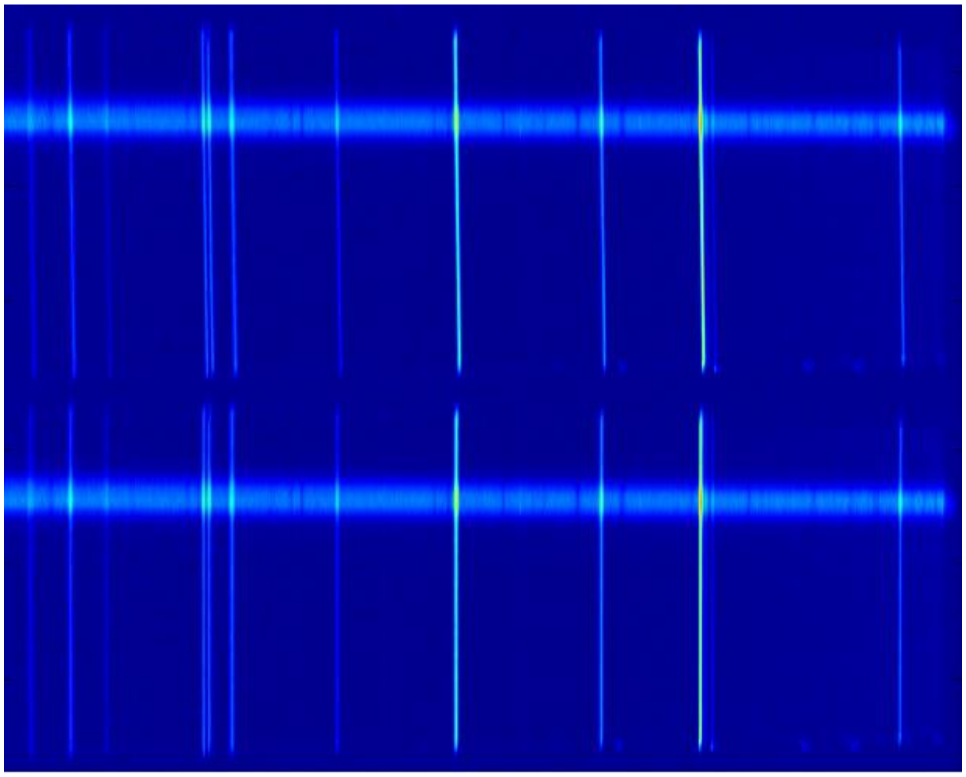

**Figure 17 Before (top) and after (bottom) spectral rectification.**

nirspec_lib.find_spectral_trace() is used to find the sky lines, trace them and average the traces. nirspec_lib.smooth_spectral_trace() is used to fit the data to a line.

## 6.6. Noise calculation

Three sources of noise are considered: shot noise due to the particle nature of light, read noise introduced by the readout amplifier, and dark current noise introduced by thermally generated electrons that collect in the pixels of the CCD.

Shot noise follows a Poisson distribution for which the standard deviation is $\sqrt{N}$ where N is the number of detected events. Shot noise in a single pixel is given by

$$\sigma_{photon}^2 = \frac{R}{G}$$

where R is the pixel value in data numbers (DN) after dark subtraction but before flat fielding and G is gain in units of $e^-/DN$. For NIRSPEC G is 5.8 $e^-/DN$

Read noise is given by

$$\sigma_{RN}^2 = \left(\frac{RN}{G}\right)^2 \frac{1}{n}$$

where RN is read noise in $e^-/pixel$ and $n$ is the number of reads. For NIRSPEC RN is 23 $e^-/pixel$.

Dark current noise is proportional to exposure time and is given by

$$\sigma_{DC}^2 = \frac{DC}{G}\, t$$

where DC is dark current in units of $e^-/\,sec/pixel$ and $t$ is exposure time in second. For NIRSPEC DC is 0.8 $e^-/\,sec/pixel$.

These noise sources also apply to the flat frame and since the noise sources are statistically independent they are added in quadrature so the total noise in each pixel is given by

$$\sigma_T^2 = \left(\sigma_p^2 hoton + \sigma_{RN}^2 + \sigma_{DC}^2\right)/flat^2$$

where $flat^2$ is the value of the corresponding pixel in the normalized flat frame.

In the DRP, noise values for each pixel are computed in a noise image where each pixel in the noise image corresponds to a pixel at the same coordinates in the object and flat frames.

Noise values for each spectral element are extracted from the noise image in the same way signal values are extracted from the object image. Thus, the noise for a given spectral element is given by

$$\sigma_{final}^2 = \sigma_{SP}^2 + \frac{SP^2}{(BK1 + BK2)^2}\,(\sigma_{BK1}^2 + \sigma_{BK2}^2)$$

where $\sigma_{SP}^2$ is the sum of $\sigma_T^2$ over the object extraction window, $\sigma_{BK1}^2$ and $\sigma_{BK2}^2$ are the sums of $\sigma_T^2$ over the two sky extraction windows and $SP$, $BK1$ and $BK2$ are the widths in pixels of the object and sky subtraction windows.

## 6.7. Sky line identification

Sky line identification is the association of observed sky lines with known OH emission features for wavelength calibration. Sky line identification proceeds in two main steps. First, a constant shift between the approximate wavelength scale from the grating equation and a synthesized sky spectrum with lines at known wavelengths is determined. Second, individual sky lines are identified.

The result of sky line identification is a list of (column location, known wavelength) pairs. Column location is the fractional pixel location of the sky line in the order determined by centroiding. Known wavelength is the accepted wavelength of the OH vibration-rotation transition from the sky line catalog. These data are fit to a line for the current order and the results are reported in the log but not used any further for wavelength calibration. The identified lines are also combined with sky lines identified in all orders and fit to the global wavelength equation.

In testing it was found that when there is a good fit on a per-order basis and sky lines were identified in each order, the global solution matches the per-order solution very closely. In cases where sky lines are only found in some orders, per-order wavelength calibration residuals may be lower than global fit residuals for the orders in which sky lines were found. This optimization is left as a potential future improvement to the DRP.

Wavelength shift is determined by generating a synthetic sky spectrum for the wavelength range of the order. Wavelength and relative intensity of sky lines are taken from the OH sky line atlas and used to generate a Gaussian of width equal to nominal spectral peak width for a point source. The Gaussians are summed to produce a synthesized sky spectrum as shown in the top green plot in Figure 18.

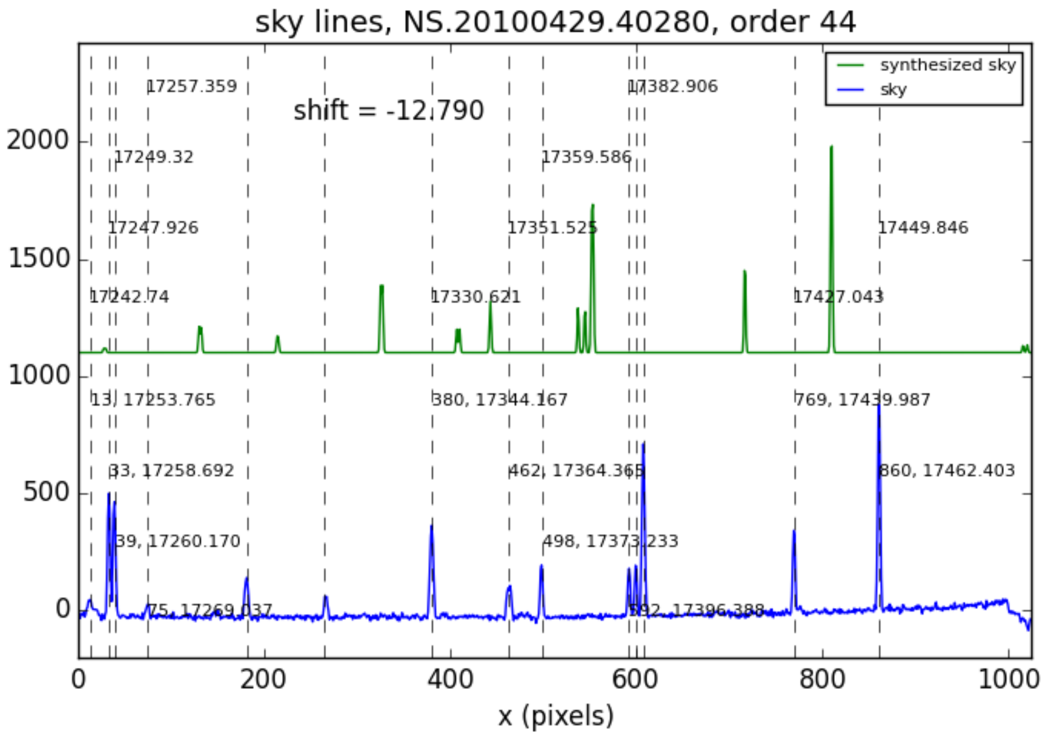

**Figure 18 Sky line identification using synthesized and real sky spectra.**

The real sky spectrum is then shifted over a range of values to maximize the convolution integral of the two spectra. The shift at which the convolution integral is greatest is taken as the wavelength shift between the approximate wavelength scale from the grating equation and the real wavelength scale. If this offset is greater than a threshold value then sky line identification fails.

Peaks in the real sky spectrum are then associated with peaks in the real sky spectrum. Closely spaced "doublets" are located first. Other strong peaks are then associated based on wavelength and intensity. The vertical dashed lines in Figure 18 indicate the lines identified in the sky spectrum and their accepted wavelengths.

## 6.8. Wavelength calibration

Known emission line wavelengths of identified OH sky lines are fit to a two-dimensional polynomial in column number and order number of the form

$$\lambda(c, M) = r_0 + r_1 c + r_2 c^2 + r_3 M^{-1} + r_4 c M^{-1} + r_5 c^2 M^{-1}$$

where λ(c, M) is the wavelength falling on column pixel c of order M and coefficients $r_0$..$r_5$ are determined by the method of least squares.

The fit is done iteratively with outliers deleted from the fit after each iteration until the RMS fit residual falls below a threshold. If too few points remain after dropping outliers then wavelength calibration fails.

Wavelength calibration failure is usually due to detection and identification of an insufficient number of sky lines in short exposures (< ~30 seconds). When wavelength calibration using sky lines fails the course of action taken depends on whether the program is running in command line mode where a single frame (or AB pair) is being reduced or if it is running in KOA mode where many frames are reduced in a single execution of the program.

When wavelength calibration fails in command line mode an approximate wavelength scale is computed from the grating equation. In this case, wavelength calibration errors can be as large as 5 – 10 Å (Is this estimate correct? Does the error depend on filter? Order?)

When wavelength calibration fails in KOA mode then multi-frame wavelength calibration is attempted. Multi-frame wavelength calibration takes advantage of the fact that long exposure frames with strong sky lines taken in the same temporally contiguous instrument configuration are usually available. For wavelength calibration in one frame to be applied to another frame, both frames must be taken with the same filter and slit and the grating and cross disperser angles must be identical. Furthermore, the instrument configuration must not have changed between the two frames. If these criteria are met then a successful wavelength calibration in one frame can be applied to an adjacent or nearly adjacent frame.

### 6.9. Background subtraction

The method used for background subtraction depends on whether a single frame or an AB pair is being reduced.

If a single frame is reduced then the background intensity level in each pixel column of the rectified order is estimated by computing the average pixel intensity in two sky windows on either side of the source spectral extraction window. This is illustrated in Figure 19 that shows the spatial profile of an order collapsed (by averaging) along the spectral axis. At each pixel column, the average intensity in the sky windows is subtracted from each pixel in the object extraction window. After background subtraction then pixel intensity is integrated along the extraction window to compute flux in the wavelength range falling on the pixel column. In the figure, the object window is denoted by a red bar centered on and at the level of the spatial peak. The two sky windows are denoted by blue bars on either side of the source extraction window and at the level of the average sky intensity.

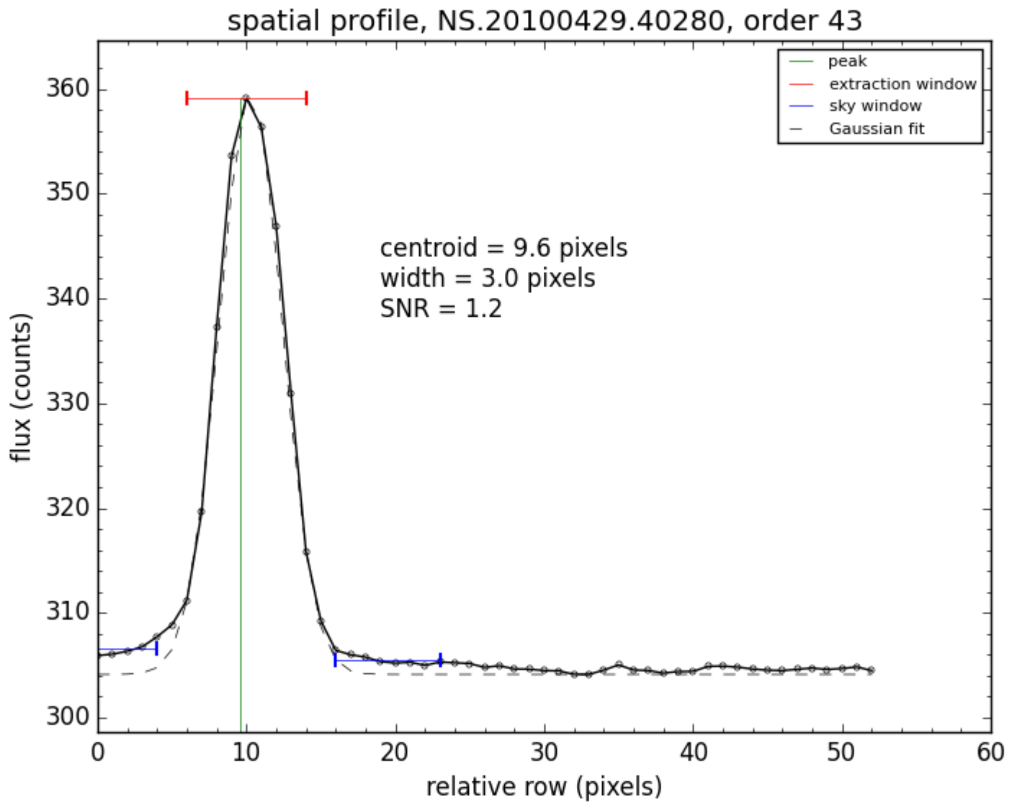

**Figure 19 Single frame spatial profile showing source and background extraction windows.**

Object and background window widths and the separation between object and background windows are set to values which are optimized for point sources; however, these parameters can be adjusted using command line options. In cases where a sky window extends beyond the edge of the detector, as in the figure, the sky window is truncated. [What happens if the object window is over the edge?]

In pair subtraction mode, where AB nod pairs are reduced, background subtraction using sky windows is not necessary. Instead, object frame B is subtracted from object frame A (after dark subtraction, before flattening) to produce a difference frame. Assuming the object is sufficiently displaced along the slit between the two frames, the difference frame will contain a positive and negative spatial peak the sky background removed (within limitations imposed by noise, temporal drift, etc.). Figure 20 shows the spatial profile of a difference frame. For each pixel column, pixel intensity is integrated along the object window and the result is used directly to compute flux without further background subtraction.

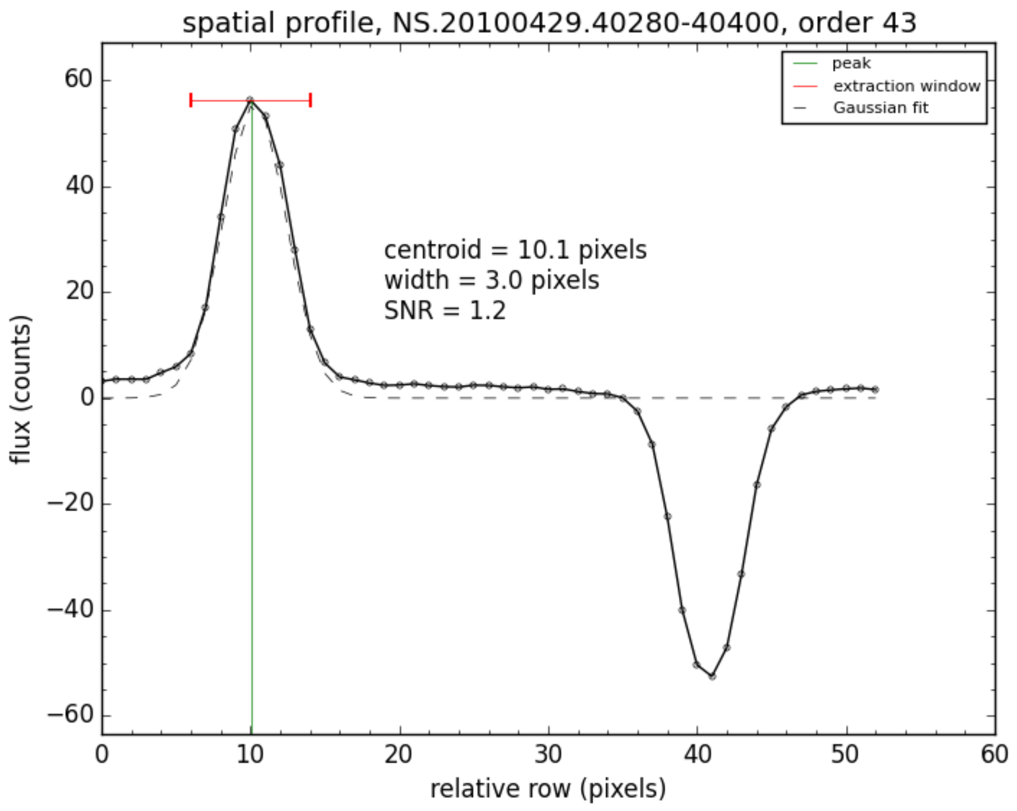

**Figure 20 Spatial profile of AB difference frame.**

It would be possible to also extract a spectrum along the negative spatial peak but the negative peak is currently ignored by NSDRP.

## 6.10. Logging

NSDRP generates two types of log files. For each execution of the program a main log file is created. For each object frame a per-object frame log file is created. Summary information is stored in the main log file and details about the reduction of each individual frame is stored in the per-object log file.

Loggers are identified in the program by logger name. Logger names are fixed and hardcoded. The main logger is called 'main' and the per-object logger is called 'obj'. The logging system provides a facility to get a handle to the desired logger by name and this alleviates the need to pass references to it from one function to another.

## 6.11. Per-Night Log

Summary information about reduction of all object frames in the input directory, typically one night's data, is stored in the main log file. The main log file is in nsdrp.log in the root output directory.

The main log file contains:

- Configuration information such as the current working directory, input and output directory paths, the number of FITS files found in the input directory.
- An explanation is given for each FITS file with IMAGETYP = 'object' that cannot be reduced, e.g. because the image was in low-resolution mode.
- For each object file that can be reduced:
    - name of object file
    - name of darks and flats associated with the object file
    - filter name and slit name
    - the number of orders on the detector and the number of orders successfully reduced
    - mean and minimum signal-to-noise ratio over all reduced orders
    - mean and maximum spatial peak width over all reduced orders
    - number of lines used in wavelength fit and RMS fit residual
    - total number of data products generated

## 6.12. Per-Object Frame Log

Details about reduction of an individual object frame are stored in the per-object log file. Per-object log files are stored in the root output directory unless the –subdirs flag is set in which case they are stored in the output directory for the frame. Per-object log files are named according to the KOA ID of the object file plus a .log extension, e.g. NS.20100429.40280.log.

Some of the information in the per-object log is redundant with information in the main log.

The per-object log file contains:

- Name of the object file and date of observation
- Filter name, slit name, Echelle angle and cross disperser angle
- The names of flats and darks used.
- An indication of whether cosmic ray artifact rejection was performed
- For each order:
    - predicted and measured order locations on the detector
    - confirmation of flat fielding
    - number of points ignored in measured trace
    - column location of spatial peak
    - number of sky lines used for spectral rectification and measured spectral tilt
    - extraction window parameters
    - signal to noise ratio
    - shift between wavelength scale determined by fitting identified sky lines in the current order and wavelength scale predicted by grating equation.
    - results of fitting measured vs. accepted wavelength of sky lines identified in the current order to a line

- Number of orders on the detector and number of orders successfully reduced
- Total number of lines used in wavelength calibration and fit residual
- Wavelength equation coefficients from fit
- Number of data product files produced.

## 6.13. Log levels

All messages written to the log files are assigned one of five levels. The meaning of each log level is described in this section.

INFO
Most log messages are at the INFO log level. These messages describe values, conditions or events that occur during normal DRP processing. There are usually many INFO messages in a log file. INFO messages include a description of the object file and associated flats and darks, location of spatial peak, wavelength fit RMS residual, etc.

WARNING
These messages indicate the occurrence of a value, condition or event that will compromise the quality of the reduction. In most cases the reduction can still proceed and useful results will be produced. WARNING message are generated, for example, when an order edge cannot be traced on the flat or the width of the spatial peak cannot be computed.

ERROR
These are conditions that should normally not occur and indicate that the reduction of the current object frame will be significantly compromised or will fail. For example, if a suitable flat cannot be found for the current object frame than an message at log level ERROR will be produced. Although reduction of the current frame will be affected by the error, it still makes sense to proceed with reduction of the next object frame.

CRITICAL
Log level CRITICAL indicates that a severe error has occurred which makes data reduction or data product generation impossible. CRITICAL errors require operator intervention. After writing a message about a CRITICAL error to the log file the DRP terminates even if additional object frames are present in the input directory. CRITCAL errors are generated, for example, if the OH emission line atlas cannot be found or the input directory cannot be accessed.

DEBUG
These messages are only written to the log file if the –debug flag is set on the command line. DEBUG messages reflect events, conditions or values that are typically only of interest to a programmer or a user troubleshooting a problem

## 7. Usage

```
usage: nsdrp.py [-h] [-debug] [-verbose] [-subdirs] [-dgn] [-npy][-cosmic] [-
products] [-obj_window OBJ_WINDOW]
[-sky_window SKY_WINDOW][-sky_separation SKY_SEPARATION]
[-oh_filename OH_FILENAME] [-int_c] in_dir out_dir
```

The following positional arguments are required:

**in_dir**

Path to directory containing the raw FITS files to be reduced.

**out_dir**

Path to the root of the output directory to be used for storage of generated data products.

The following are optional arguments.

**-h, --help**

Displays description of command line arguments.

**-debug**

Enabled additional logging for debugging. Log messages at log level DEBUG are only generated when the –debug flat is set. Also, when –debug is set, each log message includes the source code file name and line number where the message originated.

**-verbose**

Enables output of all log messages to stdout. Note that some messages are sent to both the main log file and the per-object log file; these will appear as duplicates on stdout.

**-subdirs**

Enables creation of one subdirectory per object frame. These directories are immediately subordinate to the output directory.

**-dgn**

Enables storage of intermediate data products for diagnostic purposes.

**-npy**

Enables storage of certain arrays as numpy text files for diagnostic purposes.

**-no_cosmic**

Inhibits cosmic ray artifact rejection.

**-no_products**

Inhibits generation of data products.

**-obj_window w**

Specifies object extraction window width in pixels. The default value is 9 pixels.

**-sky_window w**

Specifies background extraction window width in pixels. The default value is 8 pixels.

**-sky_separation w**

Specifies the seperation between the object extraction window and the background extraction windows on either side of it. The default value is 2 pixels.

**-oh_filename**

Specifies the location of the file containing the OH emission line atlas. The default value is ./ir_ohlines.dat.

**-int_c**

Forces use of integer column numbers for identified OH emission lines in wavelength fit.

**-lla LLA**

Calibration lines location algorithm, 1 or [2].

**-pipes**

Enables pip character separators in ASCII table headers.

**-shortsubdir**

Use file ID only, rather than full KOAID, for subdirectory names. Ignored in command line mode.

**-ut UT**

Specify UT to be used for summary log files. Overrides automatic UT determination based on UT of first frame.

**-gunzip**

Forces decompression of compressed FITS files. Leaves both .gz and .fits files in the source directory. Note that the compressed files can be read directly so it is not necessary to decompress them.

**-spatial_jump_overide**

Inhib rejection of order edge traces based on "jump" limit.

**-out_dir OUT_DIR**

Output directory path used in command line mode. Default is current working directory. Ignored in KOA mode.

**-b B**

Specifies filename of frame B in AB pair.

**-jpg**

Create preview plots in JPG format instead of PNG.

**-sowc**

Enable simple order width calculation