`flame` user manual

Sirio Belli

July 13, 2016

1 Introduction

`flame` is a pipeline for reducing near-infrared multi-object spectroscopic data, and is written in IDL. Although written specifically for the LUCI instrument at LBT, it has been designed in a modular way and can be easily adapted to work with the data produced by different instruments.

2 Installation

2.1 Requirements

The following three external IDL libraries must be installed before running `flame`.

1. NASA's IDL Astronomy User's Library;
2. David Fanning's Coyote Library;
3. Craig Markwardt's mpfit (only the mpfit.pro and mpfit2dfun.pro files are needed).

Make sure to have the most recent versions of these libraries (at least more recent than the January 2016 release).

Other third-party routines used by `flame` are included in the distribution, in the flame/lib/ directory. These are the B-spline routines from the `idlutils` library.

2.2 Download

Download the `flame` IDL code from `http://siriobelli.github.io/flame/`. Either save it in the directory where you keep your IDL code, or add the flame/ directory to the IDL path. No other step is necessary.

3 Setting up the Data Reduction

Go to the directory that you want to use for the reduction of a data set (it is recommended to create a new directory for this purpose). Then:

- Copy the file `flame_driver.pro` from the flame/pro/ directory to the current directory.

- *(optional)* Create a directory that will contain the input files (e.g., input/)
- Create an ASCII file with, in each line, the full name (including absolute path) of a raw science file (e.g., input/science.txt)
- *(optional)* Create an ASCII file with the names of the flat field frames (e.g., input/flats.txt)
- *(optional)* Create an ASCII file with the names of the dark frames (e.g., input/darks.txt)

3.1 Standard options

The data reduction is entirely controlled via the driver file `flame_driver.pro`. Open the local copy with a text editor and have a look at the code. The driver file is divided into two parts. The first part is the preamble, and sets the input parameters for this particular data set. The second part is the data reduction, and does not require any input from the user.

The data reduction in `flame` consists of a series of steps, which are represented by individual routines in the driver. The information is carried from one routine to the next via the use of only one structure, named `fuel`, which contains all the input parameters set by the user in the preamble, together with all the internal variables. The very first step is to create the `fuel` structure:

```
fuel = flame_create_fuel()
```

We now need to populate the fields in the `fuel` structure with values that are appropriate for the data set that we are reducing. We start by specifying the ASCII files containing the FITS file names:

```
fuel.science_filelist = 'input/science.txt'
fuel.darks_filelist = 'input/darks.txt'
fuel.flats_filelist = 'none'
```

In this example the flat field frames are not available and the corresponding field is set to 'none'. The same can be done for the dark frames. The science frames must, of course, always be specified. Next, we need to define the directories for intermediate and final outputs:

```
fuel.intermediate_dir = 'intermediate/'
fuel.output_dir = 'output/'
```

If these directories do not exist, they will be created by `flame`. Note that these definitions use relative paths, and will work only if you run the driver from the directory immediately above intermediate/ and output/. It is safer to always use absolute paths (e.g., /scr/observing/run_1/mask_1/output/). Finally, the last required input is the y pixel position of a reference star in both the A and the B position. These coordinates do not need to be exact, and can be typically estimated from a single frame. If the trace is too faint, a simple A-B subtraction is needed. The A and B coordinates need to be specified as the two elements of an array, for example:

```
fuel.startrace_y_pos = [547, 560]
```

When no reference star is available, the dithering pattern must be specified manually. These and other advanced features are explained in the next section.

3.2 Advanced options

3.2.1 Specify dithering pattern

It is advisable to always have a reference star on the multi-slit mask. When this is not possible, `flame` cannot measure the dithering length for each frame, and the dithering pattern needs to be saved in an ASCII file. The file must contain a number of lines equal to the number of science frames. Each line will have the value of the vertical offset, in arcseconds, corresponding to that frame (for example 5, -5, 5, -5, etc). Then the file name needs to be input in the fuel structure:

```
fuel.startrace_y_pos = [0, 0]
fuel.dither_filelist = 'input/dither.txt'
```

3.2.2 Specify the x coordinates for the reference star

When the reference star is present, its trace is extracted and used as a diagnostics tool (see Section 4.1). Only the central part of the trace is used, in the interval [1000, 1200] in x-pixel coordinates. If the trace is not present in this central region, it is possible to define a different range for the extraction:

```
fuel.xrange_star = [100, 500]
```

3.2.3 Reduce only one slit

Because reducing the entire slitmask can take a long time, especially for deep observations with a large number of frames, `flame` offers the possibility to reduce only one slit. This is particularly useful when observing, so that quick decisions can be made based on the final reduction of a representative object. To reduce only one slit we need to specify the slit index (starting from 1). For example if we are interested in the third slit:

```
fuel.reduce_only_oneslit = 3
```

The default value for fuel.reduce_only_oneslit is zero, which means that all the slits will be reduced.

3.2.4 Longslit observations

The reduction of longslit observation is slightly different from the reduction of a slit in a multi-object mask because the edges of the longslit are not well defined. This means that the slit curvature cannot be traced, and therefore it is advisable to reduce only the region of interest, possibly at the center of the detector. The longslit flag must be set, and the y pixel coordinates delimiting the region of interest must be provided:

```
fuel.longslit = 1
fuel.longslit_edge = [1150, 1250]
```

3.3 Initialization

Once all the inputs and options have been specified, the fuel structure must be initialized:

```
flame_initialize_luci , fuel=fuel
```

This will check for potential errors in the inputs, create the necessary directories, and read the relevant fields from the FITS header of the science files, such as band, central wavelength, and slit positions. This is the only step during the data reduction where instrument-specific operations are performed. Since the remaining steps are common to all near-infrared MOS data, it is in principle easy to adapt `flame` to a different instrument by simply writing the appropriate initialization module.

4 Running the Data Reduction

Once the first part of the driver file has been edited, the data reduction is fully automatic. One way to run `flame` at this point would be to execute the driver directly from the command line:

```
idl flame_driver.pro
```

Another possibility, which allows some interaction and testing, is to copy line-by-line the content of the driver file onto an interactive IDL session. This way one can explore the output files and the fuel structure after every step.

The second part of the driver file contains eight steps or modules, which we are going to examine in detail in this section. Every module is an IDL routine whose name starts with `flame_` and which is saved in a .pro file with the same name in the directory flame/pro/. All modules accept (and require) only one argument, which is the `fuel` structure, so that the IDL code for each module looks the same:

```
flame_modulename, fuel=fuel
```

Every module edits the fuel structure and/or outputs files in the intermediate directory. Only the last module, which creates the final results, saves the output files in the output directory.

4.1 `flame_diagnostics`

The first step produces important diagnostics which will be used by the successive modules. These diagnostics can also be helpful to the user to assess the quality of the observations, and can be run in real time while observing, ideally after every new frame or frame pair has been taken.

Starting with the y coordinates specified by the user in fuel.startrace_y_pos, this module detects the reference star position on each frame, and identifies which frames belong to the *A* and which to the *B* position (and assign an *X* position when no star trace is found). Then, for each frame a Gaussian profile is fit to the star trace, from which the flux, vertical position and FWHM are measured. In addition, the airmass value is measured from the FITS header of each frame.

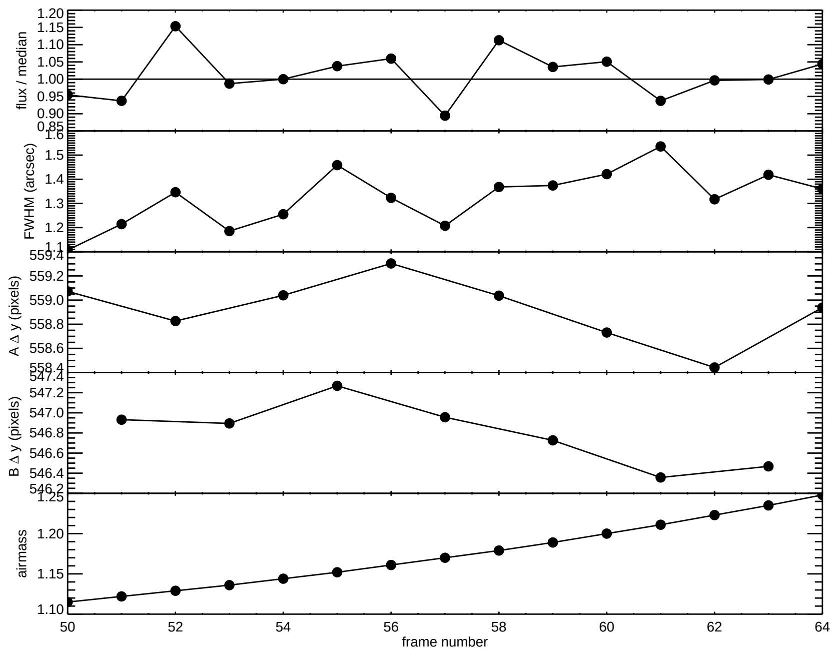


Figure 1: Example of diagnostic plots.

All these quantities are saved in the substructure fuel.diagnostics. They are also output in the ASCII file diagnostics.txt, and plotted as a function of the frame number in the file diagnostics.ps. One example of such file is shown in Figure 1. Becasue of the dithering, the vertical position is shown in different panels for the *A* and the *B* frames.

The diagnostics plots can be used to judge the trend of transmission (including the effect of cloud cover), seeing, and drift. In addition to help the user make informed decisions while observing, these measurements are also useful to identify the bad frames that should be excluded from the final data reduction.

4.2 `flame_quickstack`

The second step produces a FITS file (quickstack_A-B.fits) which is the difference of a stack of the *A* frames and a stack of the *B* frames (as defined in the previous module). This step is also useful while observing, since the simple $A - B$ subtraction often yields a decent sky removal which allows the detection of faint continuum traces or emission lines.

The sole goal of this module is to allow a quick look at the data. This step can be skipped with no consequences for the remaining parts of the data reduction.

4.3 `flame_correct`

This module applies a number of correction to the raw science frames. First, the bad pixel mask is generated from the input dark frames. If no dark frames were supplied, then the default bad pixel mask, stored in flame/data/, is used.

The following corrections are applied:

- Linearity correction: the value of each pixel is corrected to take into account the non-linear response of the detector. The correction is a polynomial function with instrument-specific coefficients, which are determined during the initialization.
- Bad pixels are set to NaN.
- The flux ofeach pixel is multiplied by the gain. This converts the flux units from ADUs to electrons.

Each of the corrected science frame is then saved in the intermediate directory.

4.4 `flame_getslits`

The module `flame_getslits` is responsible for detecting and extracting the individual slits from the multi-object slitmask. The vertical coordinates corresponding to the slit edges are calculated during the initialization are used. However, the physical position of the mask with respect to the detector is typically difficulat to predict and varies slightly from one night to the other. To account for this, a vertical shift is calculated by identifying the slit edges.

Once the approximate pixel position of each slit has been calculated, a second, more refined slit edge detection is run. This is based on the identification of individual OH emission lines, which are typically very bright and can be reliably used to trace the edge of a slit. For each slit, a low-order polynomial is fit to the y coordinate of the edge as a function of x position, and the results are saved to the fuel structure. Note that this procedure is more likely to fail when the slits are short or when there is not much space between one slit and the next one.

The output file slitim.fits is a mask image where the value of each pixel is an integer indicating the slit number to which that pixel belongs to, with zero indicating pixels that do not belong to any slit. Also, the file slits.reg is saved, which is a ds9 region file that can be loaded on top of any science frame and will show the slit edges (see Figure 2). This can be used both to check the slit detection and also to identify which slit on the science frame corresponds to which target.

Finally, all the slits are extracted from each corrected frame and saved as FITS files. All the files corresponding to the same slit are saved in the directory slitxx, where xx is the slit number.

4.5 `flame_wavecal`

This is the most important and difficult step in the data reduction. An accurate wavelength calibration is not only useful for the scientific interpretation of the data, but is also necessary for a correct sky subtraction.

For each slit, the wavelength calibration consists of two successive steps.

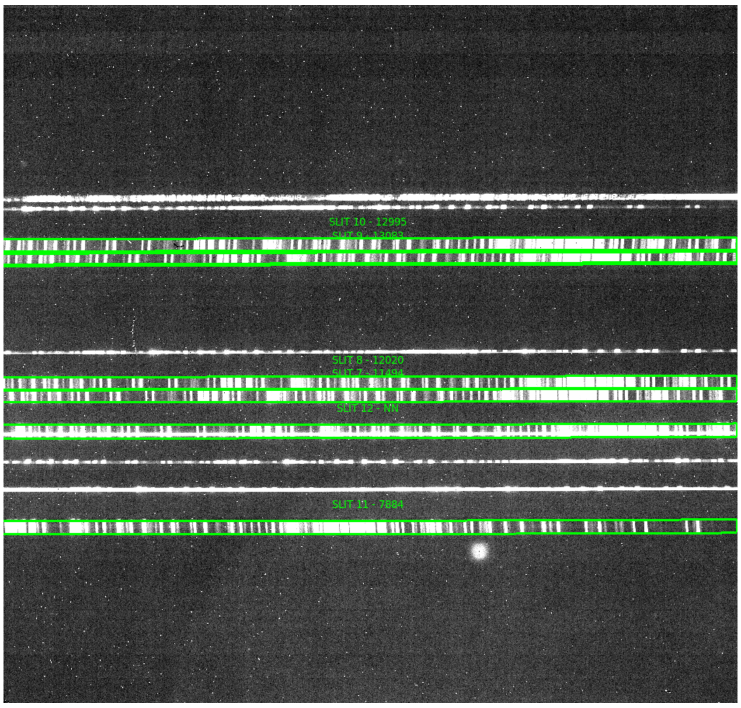


Figure 2: Aaa.

4.5.1 Approximate Wavelength Calibration

The first step finds an approximate wavelength calibration for the entire slit, using only the first science frame. The sky spectrum is extracted from the central five pixels of the slit, and is compared to a model spectrum. A very rough wavelength range is calculated from the header during the initialization, and the model sky is trimmed to a reasonable range. The comparison consists of two successive steps:

1. Both the observed and the model sky spectra are smoothed, and a series of values for the pixel scale (in micron per pixel) are tested. At each loop, cross-correlation is used to find the initial wavelength, and the pixel scale that gives the largest value of the cross-correlation is selected.

2. Once the pixel scale and the initial wavelength are roughly known, a finer comparison is necessary. In this case the spectra are not smoothed, so that the narrow OH lines can be used for a very effective comparison. The relation between x pixel coordinate and wavelength is typically not linear, and a second-order polynomial is used: $\lambda(x) = a_0 + a_1 x + a_2 x^2$. A fine grid of values for a_1 and a_2 is used, and at each loop cross-correlation is used to determine the best value for a_0.

Two plots, illustrating the two comparisons of observed and model sky spectra, are saved as ps files (estimate_wavelength_solution_slitxx.ps). An example is given in Figure 3.

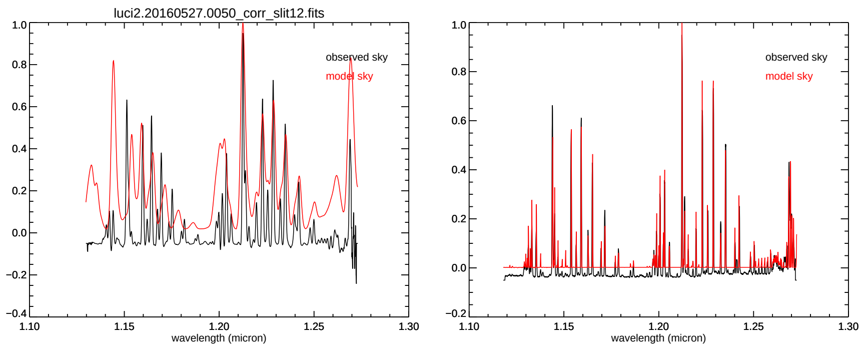


Figure 3: Approximate wavelength calibration.

4.5.2 Accurate Wavelength Calibration

The previous procedure gives only a rought wavelength calibration that is valid for the pixel rows at the center of the slits. Since the wavelength solution typically varies with spatial coordinate within the slit (particularly for tilted slits), the calibration needs to be calculated for each individual pixel row.

Initially, the spectrum of the central pixel row is extracted, and the individual OH emission lines are identified using the approximate wavelength solution. The x coordinate (in pixels) of the center of each line is calculated via fitting of a Gaussian profile. The relation between the x coordinate and the expected wavelength is then fit with a low order polynomial (see example in Figure 3). This procedure is then repeated for the next pixel row, using the wavelength solution from the previous pixel row as first guess. This is always a good assumption as long as the wavelength solution varies smoothly with position, without abrupt changes from one pixel position to the next.

After all the pixel rows have been processed, the final result is a number of x, y coordinates of detected OH lines, with the corresponding expected wavelength. A ds9 region file (OHlines.reg) showing the location of the OH lines is produced.

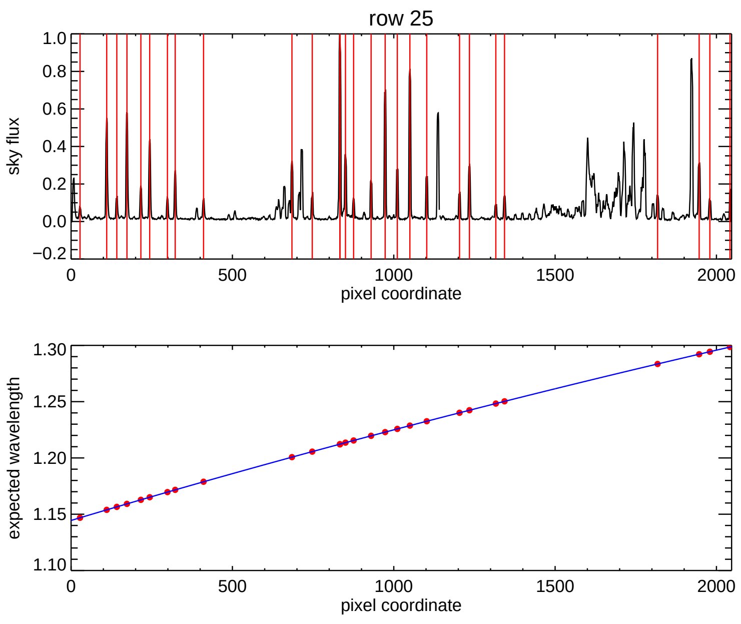
4.6 `flame_skysub`

4.7 `flame_rectify`

4.8 `flame_combine`

Credits

`flame` was written by Sirio Belli with substantial help from Alessandra Contursi for development and testing. Many features were inspired by code written by Eva Wuyts and Ric Davies for the early LUCAS pipeline, and by Nick Konidaris' MOSFIRE DRP. Dave Thompson provided useful information particularly for the LUCI-specific parts.



row 25
sky flux
pixel coordinate
expected wavelength
pixel coordinate

Figure 4: Aaa.