`flame` user manual

Sirio Belli

February 24, 2017

1 Introduction

`flame` is a pipeline for reducing near-infrared multi-object spectroscopic data, written in IDL. Although created 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.
- Create an ASCII file with, in each line, the full name (including absolute path) of a raw science file (e.g., science.txt)
- *(optional)* Create additional ASCII files with the names of the calibration frames (e.g., darks.txt, flats.txt, etc.)

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 where the input parameters are set. The second part is the data reduction, and does not require any input from the user.

The very first step is to create the `input` structure:

```
input = flame_create_input()
```

We now need to populate the fields in the `input` 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; for example like this:

```
input.science_filelist = 'science.txt'
input.dark_filelist = 'dark.txt'
input.pixelflat_filelist = 'default'
input.illumflat_filelist = 'none'
input.arc_filelist = 'none'
```

The science frames must always be specified, while for the other fields there are three possibilities: they can be set to 'none' (no calibration applied), to 'default' (the default calibration will be applied), or can be set to a text file containing the user-supplied calibration frames (such as 'dark.txt').

Finally, the y pixel position of a reference star in both the A and the B position must be provided:

```
input.star_y_A = 547
input.star_y_B = 560
```

These coordinates do not need to be exact. If the trace is very faint in a single frame, a simple A-B subtraction is generally sufficient for a visual detection. 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

The `input` structure contains additional fields that can be used to specify advanced options. We now discuss the most useful ones; the full list is reported in Table 1.

Table 1. Content of the `input` structure

Field	Default Value	Description
science_filelist	science.txt	name of ASCII file containing the list of science frames
dark_filelist	none	name of ASCII file containing the list of dark frames
pixelflat_filelist	none	name of ASCII file containing the list of frames for pixel flat field
illumflat_filelist	none	name of ASCII file containing the list of frames for illumination flat field
arc_filelist	none	name of ASCII file containing the list of lamp arc frames
dither_filelist	none	name of ASCII file containing the list of dither positions
intermediate_dir	intermediate/	directory that will contain the intermediate products
output_dir	output/	directory that will contain the final output
input.star_y_A	0	pixel y-coordinate of the A position of the star trace
input.star_y_B	0	pixel y-coordinate of the B position of the star trace
reduce_only_oneslit	0	if non-zero, reduce only that one slit
longslit	0	set to one to reduce long-slit data
longslit_edge	[0, 0]	when reducing long-slit data, only consider this range of pixel y-coordinate
star_x_range	[1000, 1200]	range of star trace (in pixel x-coordinate) that will be extracted
rough_wavecal_R	500	target spectral resolution for the rough wavecal calibration
use_sky_edge	0	if set, the slit detection is based on the sky continuum instead of OH lines

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 offset 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 `input` structure:

```
input.startrace_y_pos = [0, 0]
input.dither_filelist = '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 diagnostic 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 too faint in this central region, it is possible to define a different range for the extraction:

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

3.2.3 Reduce only one slit

Because reducing the entire slitmask can take a long time, especially for 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:

```
input.reduce_only_oneslit = 3
```

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

3.2.4 Detection of slit edges with sky background

In some cases, particularly in the K band, there are not enough bright OH lines for identifying the slit edges (see Section 4.4). It is possible to use the sky continuum emission instead, by specifying

```
input.use_sky_edge = 1
```

3.2.5 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:

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

3.3 The `fuel` structure

The data reduction in `flame` consists of a series of steps, which are represented by individual routines. 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, together with all the internal variables.

Once the `input` fields have been specified, the `fuel` structure must be created:

```
fuel = flame_create_fuel(input)
```

This will also check for potential errors in the inputs, and create the necessary directories.

The `fuel` structure consists of five substructures:

- fuel.input: a copy of the `input` structure created in the previous step;
- fuel.util: utility variables needed by `flame`;
- fuel.instrument: instrument-specific information;
- fuel.diagnostics: diagnostics such as seeing and transparency for each of the science frames;
- fuel.slits: information relative to each slit, such as position on the detector and expected wavelength range.

Initially, only fuel.input and fuel.util are populated; the other fields are set to null pointers and will be created by successive steps.

3.4 Initialization

Once the `fuel` structure has been created, it is still mostly empty. The next step creates the substructures fuel.instrument and fuel.slits:

```
flame_initialize_luci, fuel=fuel
```

During the initialization, relevant fields from the FITS header of the science files, such as band, central wavelength, and slit positions are read and saved. 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. In this way one can explore the output files and the fuel structure after every step.

The second part of the driver file contains nine 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 other 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 read from the FITS header of each frame.

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

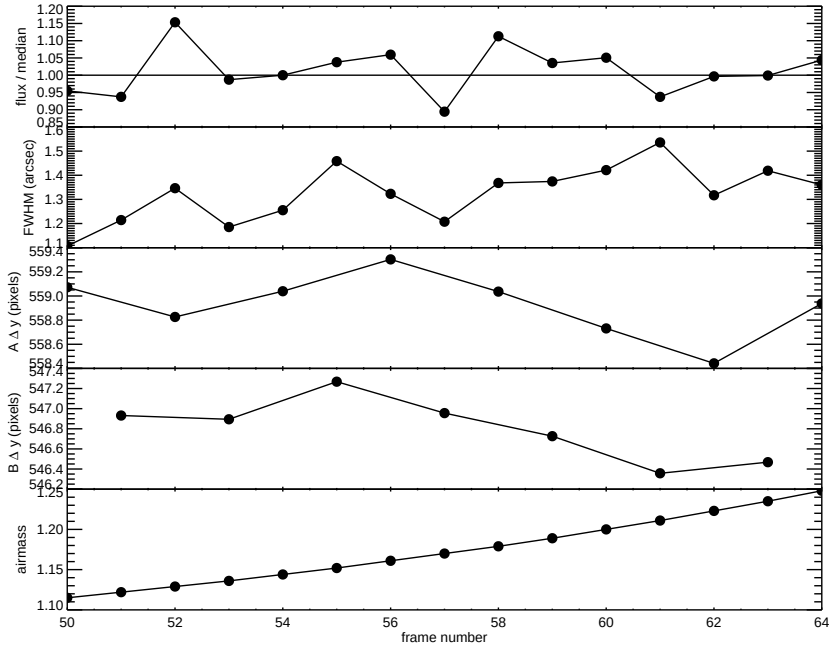

Figure 1 Example of diagnostic plots.

frame number in the file diagnostics.ps. One example of such file is shown in Figure 1. Because of the dithering, the vertical position is shown in different panels for the A and the B frames. If there are multiple frames with same frame number, which can happen when combining data from differen nights, then the x axis of the plot will show a sequential number (but the original frame numbers are used in the ASCII file).

The diagnostic 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 of each 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 of the slit edges calculated during the initialization are used. However, the physical position of the mask with respect to the detector is typically difficult 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. When observing in the K, the low number of OH lines and the strong continuum sky emission makes it convenient to use the sky background instead (see Section 3.2.4)

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 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_rough`

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

In `flame`, the wavelength calibration is split into two steps. The first step finds an approximate wavelength calibration for the central part of the slit, using

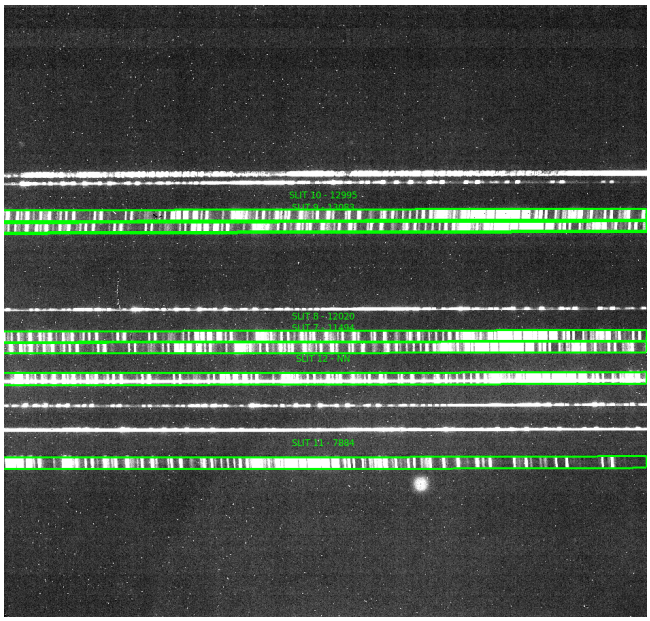

Figure 2 An example of LUCI raw frame, with the regions marking the slits identified by `flame` shown in green. The slits that are not marked are the boxes for the alignment stars, which are ignored by the pipeline.

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. The comparison consists of two successive steps:

1. Both the observed and the model sky spectra are smoothed to an intermediate spectral resolution ($R = 500$, but this value is stored in the input structure and can be changed by the user), 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 performed. In this case the spectra are not heavily 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 (wavelength_solution_estimate.ps). An example is given in Figure 3.

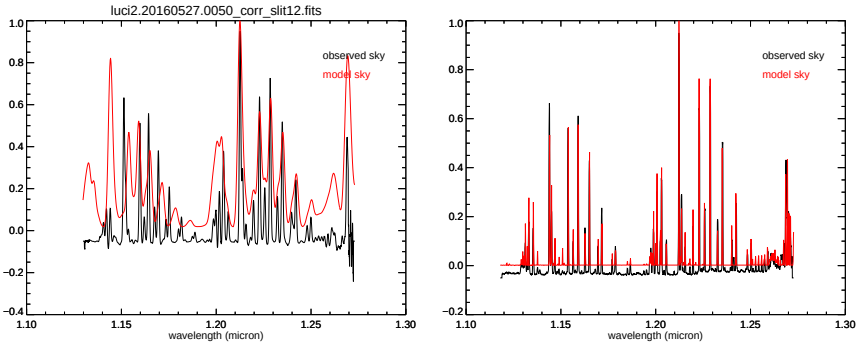

Figure 3 Approximate wavelength calibration. Left: the sky spectrum (in black) is extracted from the central region of the slit, and is compared to a template spectrum (in red). A very rough alignment is obtained by shifting the spectra and cross-correlating them. Right: A better level of approximation is then obtained by allowing a non linear transformation between pixel position and wavelength, and cross-correlating the spectra without smoothing.

4.6 `flame_wavecal_accurate`

The previous procedure gives only a rough 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, which is done in this step.

Initially, the spectrum of the central pixel row is extracted, and the individual OH emission lines are identified using the rough wavelength solution. A Gaussian fit is performed on each line, and the x pixel coordinate corresponding to the center is stored. It is therefore necessary that the calibration found in the previous step is accurate to within the window used in the Gaussian fit, otherwise the identification of the OH lines will not be possible. The relation between the x coordinate and the expected wavelength of each OH line is then fit with a low order polynomial (see example in Figure 4). 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, and the corresponding theoretical wavelength. A ds9 region file (OHlines.reg) showing the detected location of the OH lines is produced. These coordinates are used to construct the transformation between the observed (x, y) pixel position and the wavelength value. The IDL procedure POLYWARP is used to calculate the polynomial coefficients that describe the transformation.

4.7 `flame_skysub`

Because the OH emission lines vary on short timescales, the A-B subtraction typically leaves strong residuals. In order to improve the sky subtraction, `flame`

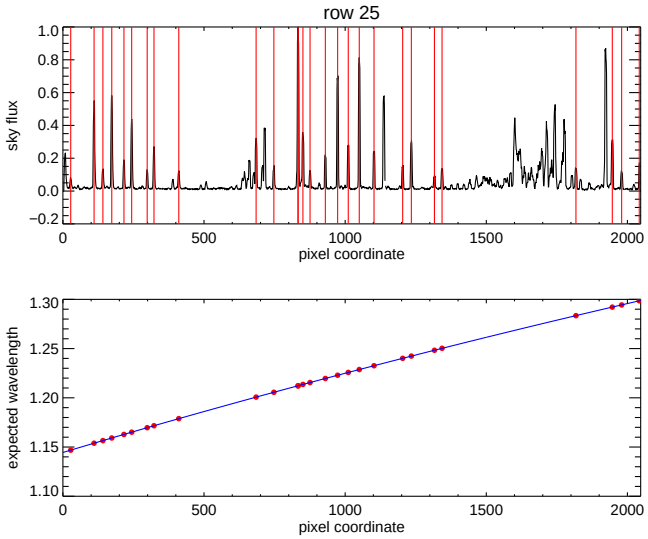

Figure 4 Accurate wavelength calibration. Top panel: The sky spectrum is extracted from a single pixel row, and using the approximate wavelength calibration the OH lines are identified and fit with a Gaussian profile. Bottom panel: the measured position of each OH line, together with their known wavelength, allow an accurate wavelength solution, shown in blue.

adopts the method of Kelson 2003. Briefly, the wavelength solution is used to map each pixel from the observed, distorted frame to the output, rectified frame. The distortion in the observed frame (which is produced by the optics and/or by the tilted slits) produces a very fine sampling of the sky lines in the rectified frame. Because of this sub-pixel sampling, it is possible to obtain an excellent B-spline fitting of the emission lines. `flame` performs this operation on each frame for every slit, and saves a FITS file of the sky-subtracted data. A graphic window shows, during this step, the sky emission spectrum (collapsed onto one dimension) and the B-spline fit for every frame.

4.8 `flame_rectify`

This simple step produces a rectified FITS file for each slit and each frame. The polynomial warping is performed using the IDL procedure POLY_2D, where the polynomial coefficients are those saved in the `fuel` structure by `flame_wavecal`.

4.9 `flame_combine`

The last step consists in the stacking (after sigma-clipping) of all the A and B frames. The individual A and B stacks, and the difference A-B, are output as FITS file for each slit. Non-skysubtracted versions of the stacks are also saved. Also, an ABcombined FITS file is produced by summing A-B to the shifted B-A images, so that one positive central trace, corresponding to the total observing time, can be extracted for the science analysis.

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 LUCAS pipeline, and by Nick Konidaris' MOSFIRE DRP. Dave Thompson provided useful information particularly for the LUCI-specific parts.