`flame` user manual

Sirio Belli

June 30, 2016

1 Introduction

`flame` is an IDL data reduction pipeline for near-infrared multi-object spectroscopy. 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 for `flame`).

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 bspline routines from the `idlutils` library.

2.2 Download

Download the `flame` IDL code from `https://github.com/siriobelli/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` directory to the current directory.

- Create a directory where the intermediate files produced by the pipeline will be saved (e.g., intermediate/)

- Create a directory where the final output of the data reduction will be saved (e.g., output/)

- *(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 parts. The first part is the preamble, and set 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.

First we need to create fuel!

First, we need to define the directories for intermediate and final outputs:

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

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/).

Now we define the input files:

```
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.

Reference star etc.

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

In alternative, need to specify the dithering pattern:

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

3.2 Advanced options

3.2.1 Reduce only one slit

Because reducing one 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.

3.2.2 Specify the x coordinates for the reference star

3.2.3 Longslit observations

3.3 Initialization

```
flame_initialize_luci, fuel=fuel
```

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.
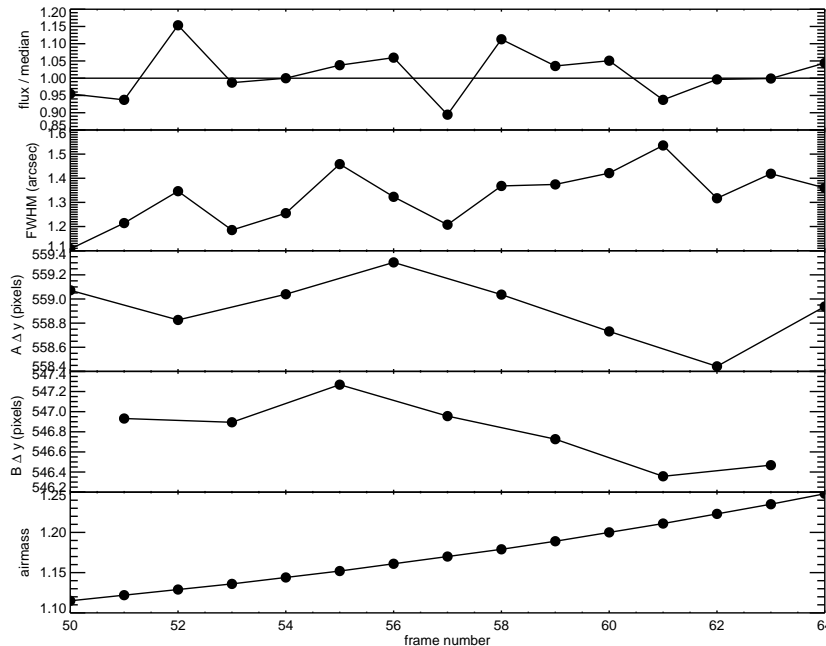
Figure 1: Example of diagnostics plot.

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.

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

4

reduction.

4.3 `flame_correct`

This module apply 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`

4.5 `flame_wavecal`

4.6 `flame_skysub`

4.7 `flame_rectify`

4.8 `flame_combine`