Flame user manual

Last updated April 6, 2018

Sirio Belli
sirio@mpe.mpg.de

Contents

1 Introduction **2**

2 Installation **2**
2.1 Requirements . . . 2
2.2 Download . . . 2

3 Setting Up the Data Reduction **3**
3.1 Basic input . . . 3
3.1.1 Science frames . . . 3
3.1.2 Nodding . . . 3
3.1.3 Reference star . . . 3
3.1.4 Slits to be reduced . . . 4
3.1.5 Longslit observations . . . 4
3.2 Optional input . . . 4
3.2.1 Calibration frames . . . 5
3.2.2 Slit edge positions . . . 5
3.2.3 Dithering positions . . . 6
3.2.4 Directories . . . 6
3.3 Initialization . . . 6

4 Running the Data Reduction **7**
4.1 `flame_diagnostics` . . . 9
4.2 `flame_quickstack` . . . 10
4.3 `flame_calibrations` . . . 10
4.4 `flame_slitid` . . . 11
4.5 `flame_cutouts` . . . 12
4.6 `flame_spatialcal` . . . 13
4.7 `flame_roughwavecal` . . . 13
4.8 `flame_findlines` . . . 14
4.9 `flame_wavecal` . . . 15
4.10 `flame_illumcorr` . . . 17
4.11 `flame_skysub` . . . 17
4.12 `flame_rectify` . . . 18
4.13 `flame_combine` . . . 18
4.14 `flame_checkdata` . . . 19
4.15 `flame_extract` . . . 19

5 Supported Instruments **20**
5.1 LUCI at LBT . . . 21
5.2 LRIS at Keck . . . 21
5.3 MOSFIRE at Keck . . . 21
5.4 Diagnostics for unsupported instruments . . . 21
5.5 How to add support for a new instrument . . . 21

1 Introduction

Flame is a pipeline for reducing near-infrared and optical multi-object spectroscopic data, written in IDL. The pipeline design and the algorithms used are described in Belli, Contursi & Davies (2017, arXiv:1710.05924), which users should read before reading the present manual.

Although created specifically for the LUCI instrument at the Large Binocular Telescope, *Flame* has been designed in a modular way and can be easily adapted to work with the data produced by different instruments. For simplicity, the present document describes in detail the reduction of LUCI data. Section 5 explains how to use *Flame* with other supported instruments, and how it is possible to implement support for new instruments. There is also a minimal version of *Flame* (see Section 5.4) that can be used to run simple diagnostics on data from virtually any instrument, and that can be particularly useful while observing.

If you made use of *Flame* in your work, please cite Belli, Contursi & Davies (2017). For comments, bug reports, or suggestions please contact `sirio@mpe.mpg.de`.

2 Installation

2.1 Requirements

The pipeline has been extensively tested in Linux and Mac. Limited testing was done in Windows.

The following three external IDL libraries must be installed before running *Flame*:

1. NASA's IDL Astronomy User's Library[1];
2. David Fanning's Coyote Library[2];
3. Craig Markwardt's mpfit[3] (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, the L.A.Cosmic routine written by Joshua Bloom and Pieter van Dokkum, and the readmhdufits routine written by Marc Kassis.

2.2 Download

Download *Flame* from `http://siriobelli.github.io/flame/`. The parent directory `flame/` contains the IDL code, documentations, and data needed to run the pipeline. You can either save this in the directory where you keep your IDL code, or save it somewhere else and then add the `flame/` directory to the IDL path. No other step is necessary.

[1] `http://idlastro.gsfc.nasa.gov/`
[2] `http://www.idlcoyote.com/code_tips/installcoyote.php`
[3] `https://www.physics.wisc.edu/~craigm/idl/fitting.html`

3 Setting Up the Data Reduction

3.1 Basic input

Create a new, empty directory that will be used for the reduction of a data set. Then:

- Copy to this directory the appropriate driver file (for LUCI observations this would be `flame_driver_luci.pro`) from the `flame/drivers/` directory. If you want to see all the advanced options, use the "complete" driver file instead (`flame_driver_complete.pro`).
- Create a text file (e.g., `science.txt`) with, in each line, the full name, including the absolute path, of a raw science frame (either FITS files or gzipped FITS files).
- *(optional)* Create additional text files with the names of the calibration frames (e.g., `darks.txt`, `flats.txt`, etc.)

The data reduction is controlled entirely via the driver file. Open the local copy with a text editor and have a look at the code. First the input parameters are set, the pipeline is then initialized, and finally the actual data reduction is run.

The first step in the driver file creates the `input` structure:

```
input = flame_create_input()
```

and the following steps populate the fields in the input structure. The data reduction is set up by modifying few lines of code in the driver file.

3.1.1 Science frames

We start by specifying the name of the file containing the science frames; for example:

```
input.science_filelist = 'science.txt'
```

If the file is not in the current working directory, then the absolute path must be provided.

3.1.2 Nodding

Near-infrared observations typically use spatial nodding for an effective sky subtraction. *Flame* supports a two-point nodding, also called A-B nodding, which is activated via the flag:

```
input.AB_subtraction = 1
```

3.1.3 Reference star

It is always recommended to place a slit on a relatively bright star (hereafter called the reference star) from which the observing conditions and the spatial nodding can be accurately measured. In the A-B nodding scheme, the star trace will appear on two different positions, which may be in the same slit or in two different slits. The y pixel position of the reference star in both the A and the B position must be provided:

```
input.star_y_A = 520
input.star_y_B = 560
```

These coordinates do not need to be exact. If the trace of the reference star 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 nodding pattern must be specified manually (see Section 3.2.3). If no AB nodding is performed, the B star position should be set to zero.

3.1.4 Slits to be reduced

While the default behavior is to reduce all the slits in the mask, it is sometimes preferable to reduce only one slit, for example to obtain a fast reduction for one representative object while observing. This can be achieved by specifying:

```
input.reduce_only_oneslit = 2
```

where in this case we would get the reduction of only the second slit. The default value is zero which means that all the slits will be reduced.

3.1.5 Longslit observations

Finally, to reduce longslit observations, it is sufficient to set the longslit flag:

```
input.longslit = 1
```

Since the longslit typically spans the whole detector in the vertical range, the edges of the longslit are not well defined. The region of interest must be specified by providing the y pixel coordinates of the top and bottom limits:

```
input.longslit_edge = [1050, 1350]
```

If `longslit_edge` is set to [0,0], the default is to reduce the full frame excluding 10% of pixels at the bottom and 10% at the top. For example, for a frame that is 2048 pixels tall, this would correspond to the [205, 1843] range. The spatial distortion is not rectified for longslit data, because it cannot be traced using the slit edges.

3.2 Optional input

The `input` structure contains additional fields that can be used to specify optional inputs. The full list is reported in Table 1, and the content of the `input` structure can be explored at any time by typing

```
help, input
```

in the interactive IDL session.

The extended driver file `flame_driver_complete.pro` includes all the possible types of input and settings: the relevant lines can be copied and pasted in the local copy of the driver.

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
AB_subtraction	0	flag to enable AB subtraction
star_y_A	0	pixel y-coordinate of the A position of the star trace
star_y_B	0	pixel y-coordinate of the B position of the star trace
reduce_only_oneslit	0	if equal to n and non-zero, reduce only the n-th slit
longslit	0	set to 1 to reduce long-slit data
longslit_edge	[0, 0]	when reducing long-slit data, only consider this range of pixel y-coordinate
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
illumflat_pixelshift	0	vertical pixel shift required to align the illumination flat with the science frames
slitflat_filelist	'none'	name of ASCII file containing the list of slit flats
slitflat_pixelshift	0	vertical pixel shift required to align the slit flat with the science frames
arc_filelist	'none'	name of ASCII file containing the list of lamp arc frames
arc_pixelshift	0	vertical pixel shift required to align the arc with the science frames
slit_position_file	'none'	name of ASCII file containing the approximate y-coordinates of the slit edges
dither_file	'none'	name of ASCII file containing the list of dither positions
max_slitwidth_arcsec	0	threshold value of the slit width to identify the alignment boxes
intermediate_dir	'intermediate/'	directory that will contain the intermediate products
output_dir	'output/'	directory that will contain the final output

3.2.1 Calibration frames

The calibration frames used by *Flame* are the dark frames, pixel flats, illumination flats, slit flats, and arcs. Each of these types of calibrations can be supplied to the pipeline via text files containing a list of FITS files, similarly to what done for the science data. The only difference is that all the calibration files are optional, and the default value for the filelist parameter is always `'none'`. Here is an example:

```
input.dark_filelist = 'darks.txt'
input.pixelflat_filelist = 'pixelflats.txt'
input.illumflat_filelist = 'none'
input.slitflat_filelist = 'twilightflats.txt'
input.arc_filelist = 'none'
```

For the illumination flats, slit flats, and arcs, it is important that the calibration frames are spatially aligned with the science frames. However, it is often the case that calibrations taken in the afternoon show a vertical displacement compared to science observations because of flexure. In this case, it is possible to correct for this by specifying the vertical offset, in pixel, that must be applied to the calibration frames:

```
input.illumflat_pixelshift = 7
input.slitflat_pixelshift = 8
input.arc_pixelshift = 0
```

Warning: the use of arcs with LUCI observations has not been tested.

3.2.2 Slit edge positions

Normally, the expected positions of the slits on the detector are derived from the values in the FITS header. However, sometimes this can fail, particularly

when the slits are tilted or curved. In these cases it is possible to specify the list of approximate pixel positions of each slit edge as an ASCII file:

```
input.slit_position_file = 'slit_edges.reg'
```

The easiest way to generate this file is to open the slit flat field (or a science frame) with ds9, make a simple circular region at each slit edge (both top and bottom edges for each slit must be specified), and save the regions in a file with format 'XY' and coordinate system 'Image'.

3.2.3 Dithering positions

It is advisable to always have a reference star on the multi-slit mask. When this is not possible, *Flame* cannot measure the nodding and dithering offset for each frame, and this 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.dither_file = 'dither.txt'
```

3.2.4 Directories

Finally, it is possible to specify the directories for intermediate and final outputs:

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

If these directories do not exist, they will be created by *Flame*. Note that these definitions use relative paths, but absolute paths can also be used.

3.3 Initialization

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_initialize_luci(input)
```

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

The `fuel` structure consists of six substructures:

- fuel.input: a copy of the `input` structure created in the previous step;
- fuel.settings: a series of settings and parameters that control the data reduction;
- 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.

Some of the fields in the `fuel` structure are initially set to null pointers and will be created by successive steps.

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.

After the initialization, the user has complete control over the `fuel.settings` structure. We will describe some of the settings in the next section; the complete list together with a brief description for each of them is shown in Table 2. Normally, the user should not edit the other five substructures contained in `fuel`.

4 Running the Data Reduction

Once the first part of the driver file has been edited, the data reduction can be executed. Broadly speaking there are two ways to run *Flame*:

1. The user can copy line-by-line the content of the driver file into an interactive IDL session. This allows the user to check the result of each individual step, exploring the `fuel` structure and the output files and plots.
2. The driver can be executed automatically, by typing:

   ```
   .run flame_driver_luci.pro
   ```

   in the IDL command line (assuming the driver file is located in the current directory). The `.run` command executes a file as an IDL *main program*, which means that the driver file must have an END statement but does not need a PRO statement at the beginning, as opposed to regular routines[4].

The driver file contains fourteen steps or modules. 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
```

Every module edits the fuel structure and/or outputs files in the intermediate directory. Only the last two modules, which create the final results, save the output files in the output directory.

In this section for each module we briefly describe its goal, the output produced, and how to solve potential problems encountered while running *Flame*.

[4]Technically, the driver file can also be executed as a *batch file*, by running `idl flame_driver_luci.pro` at the UNIX command line, although this will produce an error message when the END statement is encountered. This method is discouraged because if one of the modules produces an error, the execution will not stop but will continue with the remaining modules, making it difficult for the user to understand the source of the problem.

Table 2: Content of the `fuel.settings` structure

Field	Default Value (for LUCI)	Routine	Description
sky_emission_filename	-	roughwavecal	None
linelist_filename	-	(many)	None
star_x_range	[1000, 1200]	diagnostics	range of pixel x-coordinate to consider for the reference star trace
star_y_window	≃4 arcsec	diagnostics	range of pixel y-coordinate to consider for the reference star trace
clean_individual_frames	0	calibrations	identify and mask cosmic rays in each frame
badpix_useflat	1	calibrations	use pixel flat field (if provided) to identify bad pixels
badpix_usedark	1	calibrations	use dark frames (if provided) to identify bad pixels
badpix_sigma	7.0	calibrations	sigma clipping to identify bad pixels in the master dark frame
badpix_flatcorrection	0.20	calibrations	values beyond this deviation in the master pixel flat are bad pixels
flatfield_data	1	calibrations	divide the science data by the master pixel flat field (if provided)
darksub_data	1	calibrations	subtract the master dark (if provided) by the science data
trace_slit_with_emlines	1	slitid	use arcs or sky lines to trace slit edges
trace_slit_xmargin	20	slitid	horizontal margin (in pixels) to consider around each emission line
trace_slit_ymargin	12	slitid	vertical margin (in pixels) to consider beyond the expected slit edge
trace_slit_polydegree	2	slitid	degree of the polynomial used to describe the slit edge
roughwavecal_R	[500, 1000, 3000]	roughwavecal	spectral resolution to be used for the rough wavelength calibration
roughwavecal_smooth_window	20	roughwavecal	window, in pixels, used to calculate and subtract the sky continuum
roughwavecal_split	0	roughwavecal	in the last step of rough_wavecal, split the wavelength range in two
findlines_stack_rows	0	findlines	number of pixel rows to stack when identifying arcs or sky lines
findlines_poly_degree	5	findlines	polynomial degree for the row-by-row wavelength solution
findlines_Nmin_lines	6	findlines	minimum number of lines needed to consider a pixel row
findlines_linefit_window	6.0	findlines	window used for Gaussian fitting, in units of expected line width
wavesolution_order_x	3	wavecal	polynomial degree for the wavelength solution, along x
wavesolution_order_y	2	wavecal	polynomial degree for the wavelength solution, along y
shift_arcs	1	wavecal	apply wavelength shift calculated from sky lines to the arcs solution
shift_arcs_Nmin_lines	1	wavecal	minimum number of sky lines needed for shifting the arcs solution
illumination_correction	1	illumcorr	apply illumination correction derived from sky lines
skysub	1	skysub	construct and subtract a model of the sky
skysub_plot	0	skysub	while fitting the sky, show pop-up windows with the b-spline fit
skysub_plot_range	[0.4, 0.6]	skysub	fractional range of the x axis to plot during sky subtraction
skysub_bspline_oversample	1.0	skysub	breakpoints separation for the b-spline model, in units of pixels
skysub_reject_fraction	0.10	skysub	fraction of pixels to reject at each loop
skysub_reject_loops	3	skysub	number of loops for outlier rejection
skysub_reject_window	2.0	skysub	window for outlier rejection, in units of breakpoint separation
interpolation_method	'NaturalNeighbor'	rectify	method used to resample frames during rectification
frame_weights	'None'	combine	weights to use for the final frmae stacking
combine_sigma_clip	2.0	combine	sigma-clipping to use when combining frames
combine_min_framefrac	0.49	combine	minimum fraction of frames that must contribute to a pixel
stop_on_error	1	(all)	halt execution when an error occurs

4.1 `flame_diagnostics`

The first step produces important diagnostics which will be used by 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.

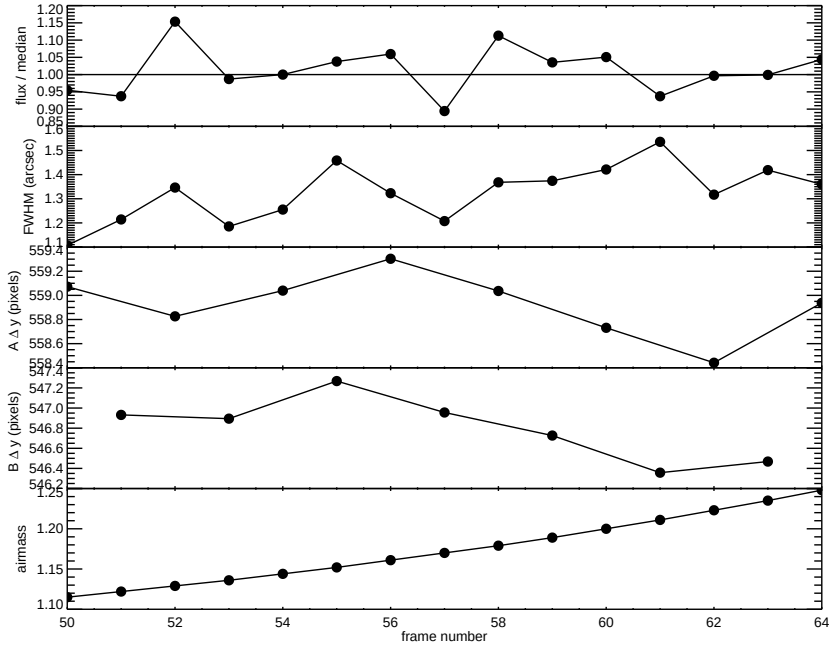

Figure 1: Example of diagnostic plots. From the top: Flux of the star trace, normalized by the median value; FWHM in arcsec; vertical position of the trace in A frames; vertical position of the trace in the B frames; airmass value read from the FITS header.

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 assigns 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` and plotted as a function of the 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.

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.

Outputs. The main output file is `diagnostics.ps`. The same information is also available in tabular form in `diagnostics.txt`. It is a good

idea to check that each frame has been assigned to the correct nodding position, and that the measured seeing, flux variation, and position are reasonable. If one or more frames have unexpected values, additional plots written to `startrace_identify_AB.ps` and `startraces.ps` should be checked.

Troubleshooting. If some of the fits to the stellar trace fail, there are three things that the user can do to improve the fits:

1. The input values (`input.star_y_A` and `input.star_y_B`) can be slightly adjusted. Particularly if the first frame was used as a reference, and there is substantial drift among frames, choosing a value that is more representative of all frames can help.

2. The stellar trace is extracted from a relatively narrow range of x pixels, which can be changed via `settings.star_x_range`. Sometimes the stellar spectrum only covers part of the detector, for example if the slit was placed at the left or right edge of the mask. If sky lines are strong, it is advisable to choose a relatively clear, narrow range; if the star is very faint, it may be better to use a wider range.

3. The vertical window used to extract the stellar profile can also be changed, via `settings.star_y_window`. The stellar trace should be within this window in all frames, so a wider window should be used when dealing with large drift or dithering. Sometimes fits can fail because the spatial profile is disturbed by the presence of the slit edge; in these cases the size of the window should be decreased.

4.2 `flame_quickstack`

The second step produces a FITS file 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.

This step and the previous ones can be run on virtually any data set without any knowledge on the instrument except for the spatial pixel scale. This minimal version of *Flame*, designed for on-the-fly use, is described in Section 5.4.

Outputs. The only output is the file named `quickstack_A-B.fits` or similar (according to the nodding pattern).

4.3 `flame_calibrations`

This module applies a number of calibrations to the raw science frames. First, the bad pixel mask is generated from the input dark and/or flat frames. If neither dark nor flat field frames were supplied, then the default bad pixel mask, stored in flame/data/LUCI, is used. The flat field correction and dark subtraction are then performed.

In order to offer the maximum flexibility, individual corrections can be turned on or off *even when the corresponding frames have been supplied to* Flame. For example, if the pixel flat field frames are provided, the user can choose whether they should be used for the flat field correction, for the bad pixel identification, or both, via the `settings.flatfield_data` and `settings.badpix_useflat` options. Analogous settings apply to the dark frames.

When using the flat field to construct the bad pixel mask, the option `settings.badpix_flatcorrection` controls the threshold used to identify bad pixels. A similar option exists for when the darks are used. If darks and flats are not used then the default bad pixel mask will be used. Users that wish to skip the bad pixel flagging altogether need to additionally set the value of `fuel.instrument.default_badpixel_mask = 'none'`

The user can also choose to clean individual frames of cosmic rays using L.A.Cosmic (this is generally not needed for LUCI data, as long as the number of frames is enough for sigma-clipping).

Outputs. After all the calibrations have been applied, the corrected science frames are saved in the intermediate/frames/ directory. Each FITS file contains a second extension with the error spectrum, which is calculated as the combination of read-out noise and Poisson noise. The `master_slitflat.fits` file is saved in the intermediate directory. Additional files may be saved, depending to the options used. These include the master flats and darks, the median pixel flat (which is a simple median stack, not normalized, and is used to obtain the master file), the bad pixel mask, and ps files showing the distribution of pixel values in the master flats and/or darks and the threshold used to identify the bad pixels. If present, the master pixel flat and bad pixel mask should be checked to make sure they look reasonable.

4.4 `flame_slitid`

The module `flame_slitid` is responsible for detecting the slits from the multi-object slitmask, using the master slit flat file. 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. The order of the polynomial can be specified via the settings.

In some cases, particularly in the K band, there are not enough bright OH lines for identifying the slit edges. It is possible to use the continuum emission instead, by specifying `settings.trace_slit_with_emlines`= 0. Using lamp flats or arc frames as slit flat field is also an option.

Outputs. This module creates the file file slits.reg, which is a ds9 region file that can be loaded on top of the slit flat (or any science frame) and will show the slit edges (see Figure 2). This can be used to check the slit detection and

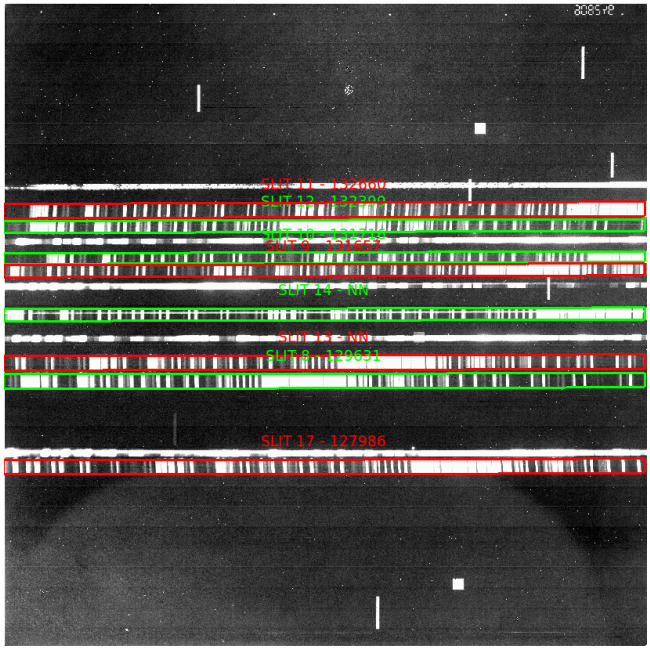

Figure 2: An example of LUCI raw frame, with the regions marking the slits identified by *Flame* shown in green. The slits that are not marked are the boxes for the alignment stars, which are ignored by the pipeline.

also to identify which slit on the science frame corresponds to which target. The raw detection (i.e., before the polynomial fit) of the slit edges are saved in slits_raw.reg.

Troubleshooting. If the slit identification fails or is not accurate, the user can change the size of the cutout used for the slit identification. If emission lines are used, a cutout is produced around each line, and the horizontal dimension of the cutout can also be changed. In these cases the slits_raw.reg file is more helpful because it shows the individual detections, and can be used to guide the tweaking of the parameters.

In very difficult cases, an alternative approach is to use the longslit option to manually select the region within one slit to reduce. By default the longslit option assumes that the slit is *spatially* centered on the mask. If this assumption causes a wildly wrong initial guess for the wavelength range, it is possible to override this value by changing the `fuel.slits.range_lambda0` variable, which is a 2-element array containing the minimum and maximum values to be considered for the wavelength of the first pixel.

4.5 `flame_cutouts`

Using the slit definition from the previous step, the slits are extracted from each corrected frame and saved as FITS files.

Outputs. For each slit, all the cutouts are saved in the directory slitxx, where xx is the slit number.

4.6 `flame_spatialcal`

The spatial rectification is calculated starting from the slit edges, and is corrected for nodding and dithering using the results of the diagnostics. The coefficients of the coordinate transformation $\gamma(x, y)$ are stored for each cutout in the fuel.slits structure.

No outputs are created during this step.

4.7 `flame_roughwavecal`

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 three steps. The first step finds an approximate wavelength calibration for the central part of the 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. 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 can be controlled via settings.roughwavecal_R), 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.

3. Finally, another cross-correlation is performed at the native spectral resolution.

Outputs. For each slit, plots illustrating the successive comparisons of observed and model sky spectra are saved as ps files (rough_wavelength_calibration.ps) in the slit directory. An example is shown in Figure 3.

Troubleshooting. The spectral resolution R used in the three steps is stored in `settings.roughwavecal_R` and can be tweaked when the default values are not satisfactory. When the wavelength solution is highly non-linear, or the wavelength range is extended, it may be difficult to obtain an accurate fit using a second-order polynomial. In these cases the user can set `settings.roughwavecal_split`, and the third step will be performed on the two halves of the spectrum separately, with a substantial increase in the flexibility of the wavelength solution.

If the sky spectrum includes a strong contribution from continuum emission, as is often the case for the thermal emission in the K band, it may be

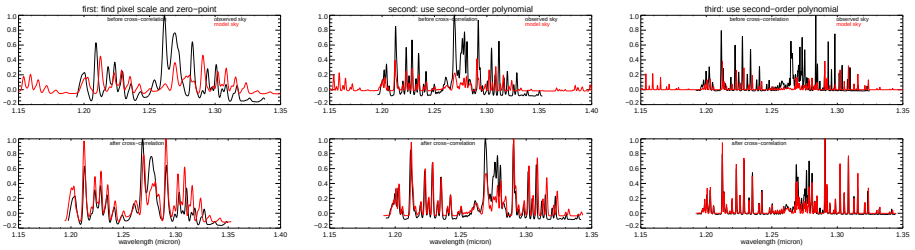

Figure 3: Approximate wavelength calibration. The sky spectrum (in black) is extracted from the central region of the slit, and is compared to a template spectrum (in red). The top panels show the initial guess, and the bottom panels show the results of the cross-correlation. The cross-correlation is repeated three times using successively higher spectral resolution.

necessary to fit and subtract a low-order polynomial to the observed spectrum before comparing it to the model. This is controlled by the parameter `settings.roughwavecal_smooth_window`.

4.8 `flame_findlines`

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 5). 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, which is stored in the `fuel` structure.

Outputs. In the slit directories, the file `summary_line_identification.ps` should be checked to get an idea of the quality and quantity of line identification. From these plots it is possible to see if there are frames with a low number of line identification, if the lines cover the 2D spectra uniformly, and what is the frame-to-frame drift along the spectral direction. A corresponding text file lists the wavelength of all the lines that were identified so that problematic lines can be easily identified and, if needed, removed from the line list. For a more thorough examination of individual frames, the `*speclines.ps` files show the identifications and the temporary wavelength solution for each individual pixel row in each frame. The `*speclines.reg` files can be loaded in ds9 on top of

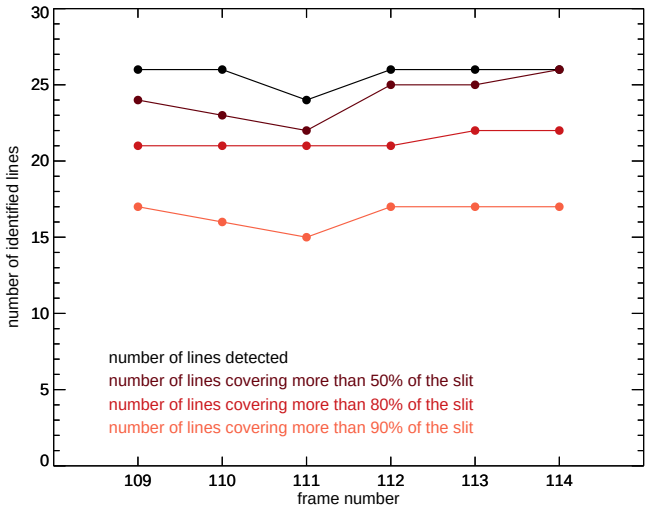

Figure 4: Number of sky lines that were identified and fit with a Gaussian, as a function of frame number. The different colors correspond to different definition of "identified lines". The black line correspond to the total number of sky lines that were detected in at least one pixel row; the other lines show increasingly stricter definitions.

the corresponding cutout to see the individual detections, as shown in Figure 5. Line identification are shown at each pixel row with red crosses; blue crosses mark those lines for which the wavelength is not accurately known, and that will be used for the rectification but not for the absolute wavelength calibration.

When using arcs, the summary files is not generated, but the `*speclines.ps` file for the arcs should be checked instead.

Troubleshooting. The first thing to check in case of poor line identification is that the previous step was successful. Without a reasonable starting guess for the wavelength solution it will not be possible to correctly identify and measure the lines. If the lines are far apart and there is no risk of mis-identifying them, it is possible to widen the window around the expected wavelength used to search for the line. This helps particularly if the quality of the rough wavelength calibration is poor.

In cases where the lines are very weak or infrequent (for example in some spectral regions in the K band), it may be necessary to tweak the settings in order to change the minimum number of lines identified on a single pixel row required to fit a wavelength solution, or the polynomial degree of the solution itself. It is also possible to stack multiple contiguous rows before attempting the line identification. This can be helpful when the lines are faint and a single pixel row does not have enough signal for a Gaussian fit.

4.9 `flame_wavecal`

For each cutout, the complete set of line identifications is used to find the best-fit wavelength solution. This is a 2D polynomial function that describes the coordinate transformation $\lambda(x, y)$. The coefficients describing this transformation are saved in the `fuel` structure.

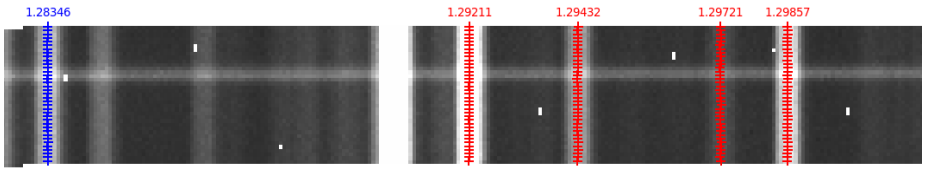

Figure 5: Identification of sky lines. At each pixel row and for each sky line a red cross mark the peak of the Gaussian fit. Lines shown in blue lack a precise wavelength measurement, and will be used only to constrain the rectification.

Outputs. In the slit directories, the file `summary_wavecal.ps` gives an overview of the wavelength calibration for all cutouts. The first plot shows the residuals asa function of wavelength, which is useful for two reasons: 1) lines that are clearly off the wavelength solution can be easily identified and removed from the line list; 2) if the residuals show a systematic trend with wavelength, a higher polynomial degree may be needed for an accurate wavelength solution. The second plot shows the line widths as a function of frame number, to test that the spectral resolution is constant throughout the observations. The third plot (see Figure 6) shows the wavelength residuals as a function of frame number. At each frame, the median and the central 68% levels are shown in red, while the blue bars mark the median absolute deviation.

For each slit and cutout, the `*wavecal.ps` files show a map of the line identifications, a plot of the measured line widths, and one of the wavelength residuals. In all these plots, the line identifications that have been discarded during the fit of the wavelength solution are marked in red. When using arcs, the `*wavecal.ps` files show the fit to the sky lines from which the wavelength shift is calculated.

When using arcs, the summary files is not generated, but the `*wavecal.ps` file for the arcs should be checked instead.

Troubleshooting. If the residuals show strong trends with wavelength, it is possible that a higher order polynomial is needed for an appropriate description of the wavelength solution. The default orders are 3 along the x axis and 2 along the y axis, and can be changed using the corresponding settings.

When using arcs, the default behavior is to extract the sky spectrum anyway, identify sky lines of known wavelength, and calculate the shift in wavelength with respect to that solution derived from the arcs (and additional figures are saved in the `*wavecal.ps` files. This shift is calculated for each cutout and applied to the wavelength solution, even in those cases where only one sky line is present in the observed spectrum (this situation is rarely seen in the near-infrared but common in the optical). It is possible to specify a minimum number of sky lines that should be identified before applying this shift. If not enough sky lines are identified, the shift is calculated by cross-correlating the observed sky spectrum with a model. This procedure, however, is not very robust when the sky emission is faint, especially at very blue wavelengths ($\lambda < 6000$Å). It is also possible to switch off the wavelength shift by setting `shift_arcs`$= 0$ (the shift is still calculated but is not applied).

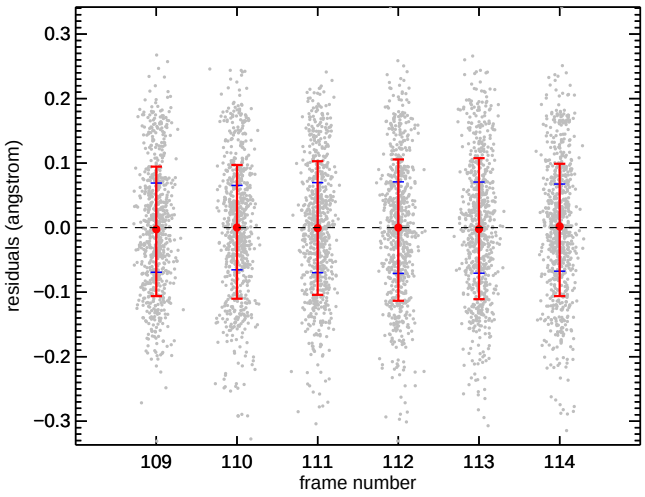

Figure 6: Residuals of the wavelength calibration. For each frame, at each pixel row each sky line is fit with the Gaussian and its position is measured. Then its wavelength is calculated using the best-fit 2D wavelength solution. The difference between this value and the true wavelength of the sky line is then shown here as a gray point. For each frame the median and 68% levels are shown in red. The MAD (median absolute deviation) is a better estimate of the level of wavelength precision that can be expected, and is shown in blue (in both positive and negative values; i.e. the distance between the two blue bar is twice the measured MAD).

4.10 `flame_illumcorr`

The illumination correction is calculated from the sky lines, the arcs, or the illumination flat, and is applied to each cutout. The user can choose to skip this step by setting the parameter `illumination_correction`= 0.

Outputs. For each cutout, the corresponding illumination-corrected FITS file (`*illcorr*.fits`) is created. From now on, this is the file that will be used in the data reduction. A figure is also saved in the `*illumcorr.ps` files, showing the measured flux across the slit in black, and the smooth model in red.

4.11 `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* constructs and subtracts a model of the sky, following the method of Kelson 2003. *Flame* performs this operation on each frame for every slit, and saves a FITS file of the sky-subtracted data.

This step improves substantially the quality of the data reduction, however it can be demanding from the computing point of view, particularly for very tall slits. If the user wants to skip this step to obtain a quick reduction, it is sufficient to set `skysub=0`. Also, if the target is extended and covers most of the slit, an accurate model of the sky cannot be constructed, and this step may well be skipped altogether.

It is possible, by setting `skysub_plot=1`, to have a pop-up window showing the sky emission spectrum (collapsed onto one dimension) and the B-spline fit

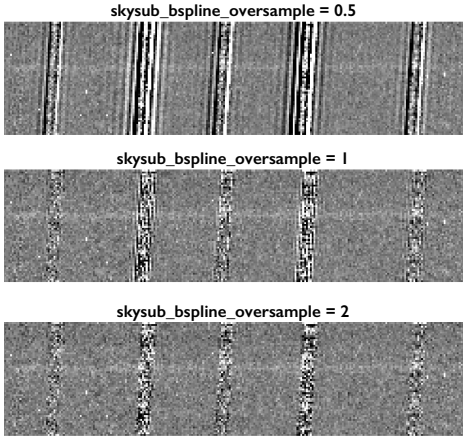

Figure 7: Aaa.

for every frame. This can be a useful diagnostic to check the quality of the sky model. It is usually preferable to have the plot zoomed in on a region rich of sky lines; the region can be set via the keyword `skysub_plot_range`. By default this pop-up window is not activate because in some machines it can significantly slow down the data reduction.

Outputs. IN each slit directory, the `*skymodel.fits` files contain a 2D model of the sky emission for each cutout. These models are subtracted from the observed frames and the results are stored in the `*skysub.fits` files.

Troubleshooting. The most common problem with the sky subtraction is a wavy shape of the sky line residuals in the sky-subtracted frames. However, when this happens the problem does not lie in the sky subtraction itself, but in the inadequate wavelength solution. Users in this case should go back to the wavelength calibration step or even the line identification step and understand what went wrong.

Another common problem is the over-subtraction due to bright emission (either line or continuum) from the target. In order to avoid subtracting the target itself, at each wavelength a generous outlier rejection is performed before fitting the sky model. Here is how the rejection works: within a given wavelength window, pixels are ranked by brightness; a fixed fraction of pixels with the most extreme values are rejected; then this procedure is repeated. For bright or extended targets this rejection should be more aggressive. This can be obtained by increasing the rejection fraction and/or the number of loops. Note that the total number of rejected pixels is uniquely determined (within rounding effects due to the relatively small number of pixels) by these two quantity. For example, the default values are 3 loops in which 10% of pixels are rejected, therefore $(1 - 0.10)^3 = 0.729$ is the remaining fraction, corresponding to a total rejection fraction of 27%. The window used for the rejection can also be tweaked to the typical wavelength over which the target and/or the sky spectrum vary.

Finally, an important aspect of the

4.12 `flame_rectify`

Using the 2D rectification coefficients calculated in `flame_wavecal_accurate`, this step performs both the wavelength calibration and the spatial rectification simultaneously, using only one interpolation. The interpolation method can be set to 'NaturalNeighbor', 'Linear', 'NearestNeighbor', or 'Quintic'.

4.13 `flame_combine`

This step consists in the stacking of all the A and B frames. Sigma-clipping is first performed to remove cosmic rays and bad pixels, the default value is 2-sigma but this can be changed. 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.

If the dithering length, as calculated from the star traces, matches the distance between any two slits, then these are assumed to be matching slits and their A-B stacks are combined together. It is worth pointing out that this assumption breaks down if the star traces for the A and B positions actually belong to different stars.

By default, no weights are applied to the frames, but the user can set frame_weights to 'None', 'Flux', 'Seeing', or 'Peak'.

4.14 `flame_checkdata`

This module performs a quality check on the reduced data set. If present, the reference star spectrum is analyzed and the effective seeing calculated. The file reference_star.ps is written, showing the spatial profile, the width and position of the stellar track as a function of wavelength, and the xtracted 1D spectrum of the star. Then, a ps file is written for each slit. The first plot shows the stacked sky spectrum, from which the sky lines are identified and fit, and the wavelength residuals and spectral resolution R are shown. Stats on the wavelength solution and the velocity resolution are shown at the bottom. The remaining plots illustrate the performance of the wavelength calibration as a function of wavelength or frame. Looking at these figures it is easy to see if one of more of the frames do not have a good wavelength solution. In such cases the user can decide to run the reduction again excluding those frames, or trying to improve the wavelength solution by tweaking the settings.

4.15 `flame_extract`

Finally, this module identifies the brightest object in each slit and extracts the 1D spectrum. Output files are saved in the directory `extraction/`, which is created inside the output directory. These include, for each slit, a ps file showing the spatial profile with its Gaussian fit, the extracted spectrum, and the SNR for both the boxcar extraction and the optimal extraction. The 1D spectrum extracted from each slit is stored in a FITS file as a structure containing the fields `lambda`, `flux`, and `ivar`, corresponding to wavelength (NB: in angstrom), flux (in electron per second), and inverse variance. The error spectrum can be easily obtained as $\mathrm{err} = 1/\sqrt{\mathrm{ivar}}$.

There are a number of third-party programs that can be used to visualize and analyze the extracted 1D spectra. In particular, the FITS files output by *Flame* are compatible with the interactive programs SpecPro and SpecViz.

SpecPro[5] is written in IDL. It uses both the IDL Astronomy User's Library and the mpfit routines, which are also required by *Flame*. After downloading and installing the source code, simply launch IDL from the `extraction/` directory and run the `specpro` command.

SpecViz[6] is written in Python. After installing SpecViz, copy the configuration file `flame/external/flame.yaml` to the `~/.specviz/` directory. This file contains a description of the data format used by *Flame*. Open SpecViz, click the *Open* button, and in the dropdown menu *Files of type* there should now be an entry called *Flame data reduction pipeline*. Note that it is not necessary to select the file type every time you open a spectrum; SpecViz should be able to automatically recognize the file and correctly read in the *Flame* output.

5 Supported Instruments

All the *Flame* modules described in Section 4 are written in an instrument-independent way, and can be used on nearly any optical or near-infrared data set. The instrument-specific part is relegated to the initialization module. When running *Flame*, the correct initialization module must be called, and those options that are required and/or relevant to the specific instrument must be set. To simplify this procedure, a different driver file is available for each instrument, although in principle a skilled user should be able to easily write the correct driver for any supported instrument. All driver files are split into three parts:

1. The inputs are set, in slightly different ways for different instruments.
2. The specific initialization module is then run and the `fuel` structure is created.
3. The data reduction modules are called. This part is identical for all drivers.

The list of files in the `flame/drivers/` directory shows what instruments are supported. Currently, this directory contains the following files:

`flame_driver_luci.pro`: driver for reducing data from the LUCI1 and LUCI2 instruments at the Large Binocular Telescope.

`flame_driver_lris.pro`: driver for reducing data from the blue and red channel of LRIS at the Keck telescope.

`flame_driver_complete.pro`: generic driver that contains all the possible options for the structure and for the settings in the `fuel` structure; it can be used as a reference by copying the relevant lines and pasting them on the actual driver.

`flame_driver_minimal.pro`: minimalistic driver that can be used to run the first two modules (diagnostics and quick stack) on any data set, including instruments that are not supported.

[5] `http://specpro.caltech.edu/`
[6] `https://github.com/spacetelescope/specviz`

We describe each supported instrument, and give indications on how to implement support for new instruments, in the remainder of this section.

5.1 LUCI at LBT

The reduction of LUCI data is described throughout Section 3 and 4, and no further discussion for this instrument is needed.

5.2 LRIS at Keck

5.3 MOSFIRE at Keck

Work in progress.

5.4 Diagnostics for unsupported instruments

5.5 How to add support for a new instrument