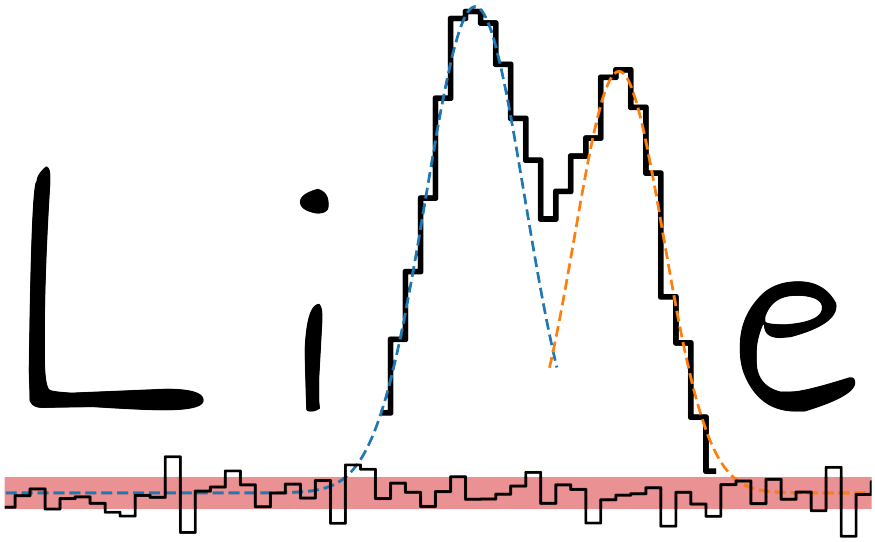
lime

Release 0.9.99.1

Vital-Fernandez

Mar 27, 2024

REFERENCE



This library provides a set of tools to fit lines in astronomical spectra. Its design aims for a user-friendly workflow for both single lines and large data sets. The library provides tools for masking, detecting, profile fitting and storing the results. The output measurements are focused on the gas chemical and kinematic analysis.

These are some of the features currently available:

- Non-profile and Gaussian profile emission and absorption line measurements.
- The user can include the pixel error spectrum in the calculation.
- The Multi-Gaussian profile parameters can be constrained by the user during the fitting.
- Tools to confirm the presence of lines.
- Static and interactive plots for the visual appraisal of inputs and outputs
- Line labels adhere to the PyNeb format.
- The measurements can be saved in several file types, including multi-page *.fits*, *.asdf* and *.xlsx* files

---

Where to find what you need

To install or update the library go to the installation page. Download the sample data folder and try to run the examples.

For a quick start go to the **Tutorials** section. These examples are organized by increasing complexity and they provide a working knowledge of the library algorithms.

To learn more about the library design, please check the **Inputs** section to understand how to adapt LiMe to your workflow. The library functions description can be found at the *API*.

The tabulated and graphical measurements description is available in the **Outputs** section.

CHAPTER

ONE

INSTALLATION

LiMe can be installed from its pip project page by running this command:

```
pip install lime-stable
```

To update the library to its latest version you can run this command:

```
pip install lime-stable --upgrade
```

1.1 Dependencies

The current version of LiMe has these dependencies:

- Numpy
- Pandas
- Matplotlib
- LmFit (fitting library)
- Astropy (loading and saving *.fits* files)
- tomli (read configuration files with toml files for python < 3.11)

The following dependencies are not compulsory but they provide more options for the library inputs/outputs:

- mplcursors (Interactive pop-ups in plots)
- asdf (Saving the logs as *.asdf* files)
- PyLatex (Saving the logs as *.pdf* files)
- openpyxl (Saving the logs as *.xlsx* files)

1.2 Development

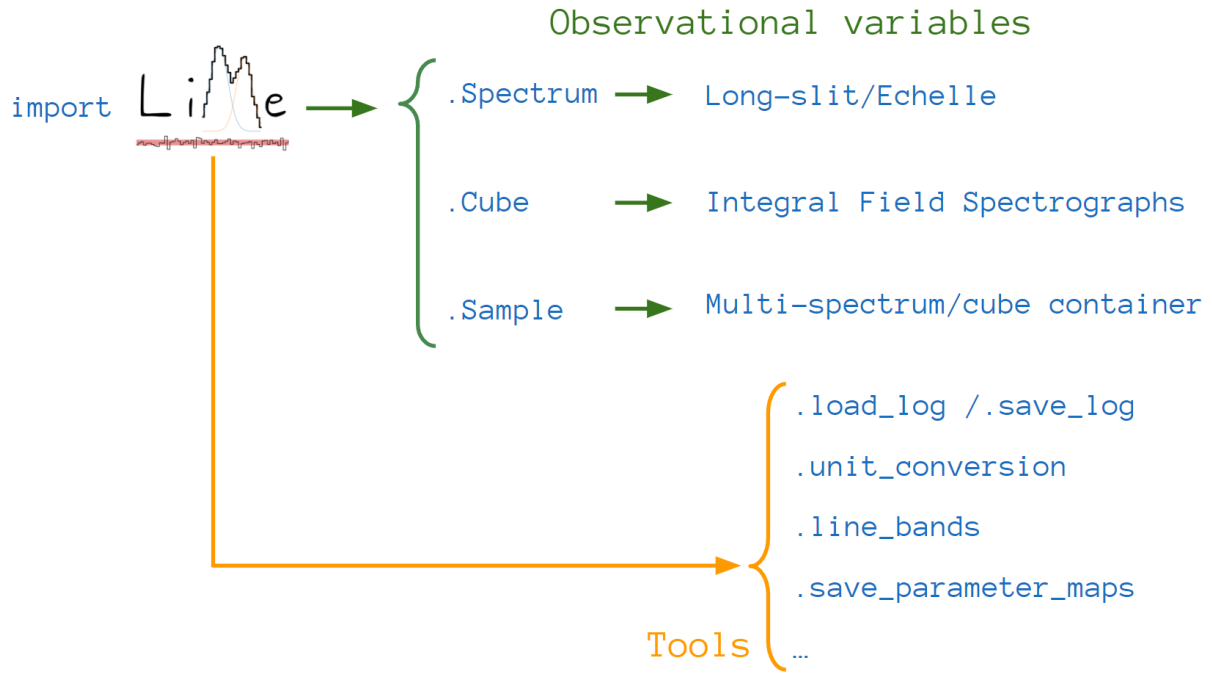
LiMe is currently in a beta release. Please check its github for the latest version news and tutorials. Any comments/problems/request can be added as an issue on the github page.

CHAPTER

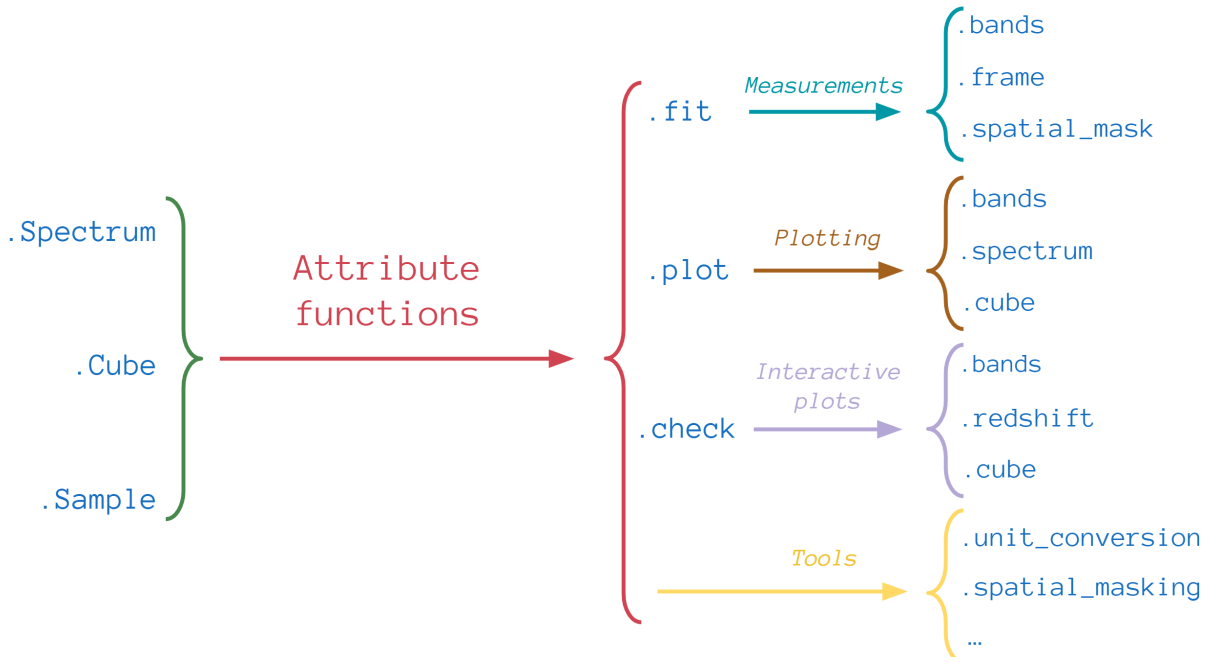
TWO

API

Most of LiMe functions can be organized in two categories. In the first one we have the `Spectrum`, `Cube` and `Sample` classes. These functions recreate astronomical observations:



The second set of functions belong to the astronomical objects mentioned above. For example, the `.fit`, `.plot` or `.check` functions allow us to measure, plot and interact with the data:



Finally, there are additional utility functions. Some of these tools belong to the astronomical objects or they can be imported directly from LiMe or both (as in the case of `.load_log`/`.save_log`)

In the following sections we describe these functions and their attributes.

2.1 Inputs/outputs

`lime.`**`load_cfg`**(*file_address*, *fit_cfg_suffix='_line_fitting'*)

This function reads a configuration file with the toml format. The text file extension must adhere to this format specifications to be successfully read.

If one of the file sections has the suffix specified by the `fit_cfg_suffix` this function will query its items and convert the entries to the format expected by LiMe functions. The default suffix is "_line_fitting".

The function will show a critical warning if it fails to convert an item in a `fit_cfg_suffix` section.

Parameters

- **`file_address`** (*`str, pathlib.Path`*) – Input configuration file address.
- **`fit_cfg_suffix`** (*`str`*) – Suffix for LiMe configuration sections. The default value is "_line_fitting".

Returns

Parsed configuration data

Type

dict

`lime.`**`load_log`**(*file_address*, *page: str = 'LINELOG'*, *levels: list = ['id', 'line']*)

This function reads the input `file_address` as a pandas dataframe.

The expected file types are ".txt", ".pdf", ".fits", ".asdf" and ".xlsx". The dataframes expected format is discussed on the line bands and measurements documentation.

osiris.toml

```
[sample_data]

files_list = ["gp030321_BR.fits", "gp101157_BR.fits", "gp121903_BR.fits"]
object_list = ["gp030321", "gp101157", "gp121903"]

z_array = [0.16465, 0.14334, 0.19531]
zErr_array = [7.389e-5, 0.000129, 0.0001403]

norm_flux = 1e-14

[gp121903_line_fitting]

O2_3726A_m = "O2_3726A+O2_3729A"
H1_3889A_m = "H1_3889A+He1_3889A"
Ar4_4711A_m = "Ar4_4711A+He1_4713A"
O2_7319A_m = "O2_7319A+O2_7330A"
H1_6563A_b = "H1_6563A+N2_6584A+N2_6548A"
O1_6300A_b = "O1_6300A+S3_6312A"

S2_6716A_b = "S2_6716A+S2_6731A"
S2_6731A_kinem = "S2_6716A"

N2_6548A_amp = "expr:N2_6584A_amp/2.94"
N2_6548A_kinem = "N2_6584A"
```

snappify.com

Fig. 1: Example of LiMe configuration file

For ".fits" and ".xlsx" files the user can provide a page name `ext` for the HDU/sheet. The default name is "_LINELOG".

To reconstruct a MultiIndex dataframe the user needs to specify the `sample_levels`.

Parameters

- **file_address** (*str, Path*) – Input log address.
- **page** (*str, optional*) – Name of the HDU/sheet for ".fits"/".xlsx" files. The default value is "_LINELOG".
- **levels** (*list, optional*) – Indexes name list for MultiIndex dataframes. The default value is ['id', 'line'].

Returns
lines log table

Return type
pandas.DataFrame

lime.**save_log**(*dataframe*, *file_address*, *page='LINELOG'*, *parameters='all'*, *header=None*, *column_dtypes=None*, *safe_version=True*, ***kwargs*)

This function saves the input `log_dataframe` at the `file_address` provided by the user.

The accepted extensions are ".txt", ".pdf", ".fits", ".asdf" and ".xlsx".

For ".fits" and ".xlsx" files the user can provide a page name for the HDU/sheet with the `ext` argument. The default name is "LINELOG".

The user can specify the `parameters` to be saved in the output file.

For ".fits" files the user can provide a dictionary to add to the `fits_header`. The user can provide a `column_dtypes` string or dictionary for the output fits file record array. This overwrites LiMe deafult formatting and it must have the same columns as the file names.

Parameters

- **dataframe** (*pandas.DataFrame*) – Lines log dataframe.
- **file_address** (*str, Path*) – Output log address.
- **parameters** (*list*) – Output parameters list. The default value is "all"
- **page** (*str, optional*) – Name of the HDU/sheet for ".fits"/".xlsx" files.
- **header** (*dict, optional*) – Dictionary for ".fits" and ".asdf" file headers.
- **column_dtypes** (*str, dict, optional*) – Conversion variable for the *records array <https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.to_records.html>*. for the output fits file. If a string or type, the data type to store all columns. If a dictionary, a mapping of column names and indices (zero-indexed) to specific data types.
- **safe_version** (*bool, optional*) – Save LiMe version as footnote or page header on the output log. The default value is True.

lime.OpenFits.**sdss**(*fits_address*, *data_ext_list=(1, 2)*, *hdr_ext_list=0*, *pixel_mask=None*)

This method returns the spectrum array data and headers from a SDSS observation.

The function returns numpy arrays with the wavelength, flux and uncertainty flux (if available this is the standard deviation available), a list with the requested headers and a dictionary with the parameters to construct a LiMe Spectrum. These parameters include the observation wavelength/flux units, normalization and wcs from the input fits file.

Parameters

- **fits_address** (*int, str or list of either, optional*) – File location address for the observation .fits file.
- **data_ext_list** – Data extension number or name to extract from the .fits file.
- **hdr_ext_list** (*int, str or list of either, optional*) – header extension number or name to extract from the .fits file.

Returns

wavelength array, flux array, uncertainty array, header list, observation parameter dict

2.2 Tools

lime.**line_bands**(*wave_intvl=None*, *lines_list=None*, *particle_list=None*, *z_intvl=None*, *units_wave='A'*, *decimals=None*, *vacuum=False*, *ref_bands=None*)

This function returns LiMe bands database as a pandas dataframe.

If the user provides a wavelength array (`wave_inter`), a lime.Spectrum or lime.Cube the output dataframe will be limited to the lines within this wavelength interval.

Similarly, the user provides a `lines_list` or a `particle_list` the output bands will be limited to the these lists. These inputs must follow LiMe notation style

If the user provides a redshift interval (`z_intvl`) alongside the wavelength interval (`wave_intvl`) the output bands will be limited to the transitions which can be observed given the two parameters.

The default line labels and bands `units_wave` are angstroms (A), additional options are: um, nm, Hz, cm, mm.

The argument `decimals` determines the number of decimal figures for the line labels.

The user can request the output line labels and bands wavelengths in vacuum setting `vacuum=True`. This conversion is done using the relation from Greisen et al. (2006).

Instead of the default LiMe database, the user can provide a `ref_bands` dataframe (or the dataframe file address) to use as the reference database.

Parameters

- **wave_intvl** (*list, numpy.array, lime.Spectrum, lime.Cube, optional*) – Wavelength interval for output line transitions.
- **lines_list** (*list, numpy.array, optional*) – Line list for output line bands.
- **particle_list** (*list, numpy.array, optional*) – Particle list for output line bands.
- **z_intvl** (*list, numpy.array, optional*) – Redshift interval for output line bands.
- **units_wave** (*str, optional*) – Labels and bands wavelength units. The default value is "A".
- **decimals** (*int, optional*) – Number of decimal figures for the line labels.
- **vacuum** (*bool, optional*) – Set to True for vacuum wavelength values. The default value is False.
- **ref_bands** (*pandas.Dataframe, str, pathlib.Path, optional*) – Reference bands dataframe. The default value is None.

Returns

lime.**label_decomposition**(*lines_list*, *bands=None*, *fit_conf=None*, *params_list=('particle', 'wavelength', 'latex_label')*, *scalar_output=False*)

This function takes a `lines_list` and returns several arrays with the requested parameters.

If the user provides a bands dataframe (`bands` argument) dataframe and a fitting documentation. (`fit_conf` argument) the function will use this information to compute the requested outputs. Otherwise, only the line label will be used to derive the information.

The `params_list` argument establishes the output parameters arrays. The options available are: "particle", "wavelength", "latex_label", "kinem", "profile_comp" and "transition_comp".

If the `lines_list` argument only has one element the user can request an scalar output with `scalar_output=True`.

Parameters

- **lines_list** (*list*) – Array of lines in LiMe notation.
- **bands** (*pandas.Dataframe, str, path.Pathlib, optional*) – Bands dataframe (or file address to the dataframe).
- **fit_conf** (*dict, optional*) – Fitting configuration.
- **params_list** (*tuple, optional*) – List of output parameters. The default value is ('particle', 'wavelength', 'latex_label')
- **scalar_output** (*bool*) – Set to True for a Scalar output.

lime.Spectrum.**line_detection**(*self*, *bands=None*, *cont_fit_degree=(3, 7, 7, 7)*, *cont_int_thres=(5, 3, 2, 2)*, *noise_sigma_factor=3*, *line_type='emission'*, *width_tol=5*, *band_modification=None*, *ml_detection=None*, *plot_cont_calc=False*, *plot_peak_calc=False*)

This function compares the input lines bands in the observation spectrum to confirm the presence of lines.

The input bands can be specified as a pandas dataframe or the path to its file via the `bands_df` argument.

Prior to the line detection, the spectrum continuum is fit in an iterative process. The `cont_fit_degree` array provides the order of the fitting polynomial, while the `cont_int_thres` array provides the threshold intensity factor for the threshold flux above and below the continuum to exclude at each iteration.

After the continuum has been normalized, the `noise_sigma_factor` establishes the standard deviation factor beyond which a positive line detection is assumed.

The additional arguments provide additional tools to adjust the line detection and show the steps/results.

Parameters

- **bands** (*pandas.Dataframe, str, pathlib.Path*) – Input bands dataframe or the address to its file.
- **cont_fit_degree** (*tuple, optional*) – Continuum polynomial fitting degree.
- **cont_int_thres** (*tuple, optional*) – Continuum maximum intensity threshold to include pixels.
- **noise_sigma_factor** (*float, optional*) – Continuum standard deviation factor for line detection. The default value is 3.
- **line_type** (*str, optional*) – Line type. The default value is "emission".
- **width_tol** (*float, optional*) – Minimum number of pixels between peaks/troughs. The default value is 5.

- **band_modification** (*str, optional*) – Method to adjust the line band with. The default value is None.
- **ml_detection** (*str, optional*) – Machine learning algorithm to detect lines. The default value is None.
- **plot_cont_calc** (*bool, optional*) – Plot the continuum fit at each iteration. The default value is False
- **plot_peak_calc** (*bool, optional*) – Plot the detected peaks/troughs. The default value is False

lime.tools.**logs_into_fits**(*log_file_list*, *output_address*, *delete_after_join=False*, *levels=['id', 'line']*)

This functions combines multiple log files into single *.fits* file. The user can request to the delete the individual logs after the individual logs have been combined.

If the case of individual *.fits* the function loop through the individual HDU and add them to the output file. Currently, this is not available to other multi-page files (such as .xlsx or .asdf)

Parameters

- **log_file_list** (*list*) – Input list of log files.
- **output_address** (*bool, optional*) – String or path for the output combined log file.
- **delete_after_join** – Delete individual files after joining them. The default value is False
- **levels** (*list, optional*) – Indexes name list for MultiIndex dataframes. The default value is [‘id’, ‘line’].

Returns

lime.**unit_conversion**(*in_units*, *out_units*, *wave_array=None*, *flux_array=None*, *dispersion_units=None*, *decimals=None*)

This function converts the input array (wavelength or flux) `in_units` into the requested `out_units`.

Attention: Due to the nature of the `flux_array`, the user also needs to include the `wave_array` and its units in the `dispersion_units`. units

The user can also provide the number of `decimals` to round the output array.

Parameters

- **in_units** (*str*) – Input array units
- **out_units** (*str*) – Output array untis
- **wave_array** (*numpy.array*) – Wavelength array
- **flux_array** (*numpy.array*) – Flux array
- **dispersion_units** –
- **decimals** (*int, optional*) – Number of decimals.

2.3 Astronomical objects

class lime.**Spectrum**(*input_wave=None*, *input_flux=None*, *input_err=None*, *redshift=None*, *norm_flux=None*, *crop_waves=None*, *inst_FWHM=nan*, *units_wave='AA'*, *units_flux='FLAM'*, *pixel_mask=None*, *id_label=None*, *review_inputs=True*)

This class creates an astronomical cube variable for an integral field spectrograph observation.

The user needs to provide wavelength and flux arrays. Additionally, the user can include a flux uncertainty array. This uncertainty must be in the same units as the flux. The cube should include its `redshift`.

If the flux units result in very small magnitudes, the user should also provide a normalization to make the flux magnitude well above zero. Otherwise, the profile fittings are likely to fail. This normalization is removed in the output measurements.

The user can provide a `pixel_mask` boolean array with the pixels **to be excluded** from the measurements.

The default `units_wave` are angtroms (Å), additional options are: um, nm, Hz, cm, mm

The default `units_flux` are Flam (erg s^-1 cm^-2 Å^-1), additional options are: Fnu, Jy, mJy, nJy

The user can also specify an instrument FWHM (`inst_FWHM`), so it can be taken into account during the measurements.

The user can provide a `pixel_mask` boolean array with the pixels **to be excluded** from the measurements.

Variables

- **fit** – Fitting function instance from `lime.workflow.SpecTreatment`.
- **plot** – Plotting function instance from `lime.plots.SpectrumFigures`.

Parameters

- **input_wave** (*numpy.array*) – wavelength array.
- **input_flux** (*numpy.array*) – flux array.
- **input_err** (*numpy.array, optional*) – flux sigma uncertainty array.
- **redshift** (*float, optional*) – observation redshift.
- **norm_flux** (*float, optional*) – spectrum flux normalization.
- **crop_waves** (*np.array, tuple, optional*) – spectrum (minimum, maximum) values
- **inst_FWHM** (*float, optional*) – Instrumental FWHM.
- **units_wave** (*str, optional*) – Wavelength array units. The default value is "A".
- **units_flux** (*str, optional*) – Flux array physical units. The default value is "Flam".
- **pixel_mask** (*np.array, optional*) – Boolean array with True values for rejected pixels.
- **id_label** (*str, optional*) – identity label for the spectrum object

lime.Spectrum.**from_file**(*file_address*, *instrument*, *mask_flux_entries=None*, ***kwargs*)

This method creates a lime.Spectrum object from an observational (.fits) file. The user needs to introduce the file address location and the name of the instrument of survey.

Currently, this method supports NIRSPEC, ISIS, OSIRIS and SDSS as input instrument sources. This method will lower case the input instrument or survey name.

The user can include list of pixel values to generate a mask from the input file flux entries. For example, if the user introduces [np.nan, 'negative'] the output spectrum will mask np.nan entries and negative fluxes.

This method provides the instrument observational units and normalization but the user should introduce the additional LiMe.Spectrum arguments (such as the observation redshift).

Parameters

- **file_address** (*Path, string*) – Input file location address.
- **instrument** (*str*) – Input file instrument or survey name
- **mask_flux_entries** (*list*) – List of pixel values to mask from flux array
- **kwargs** – lime.Spectrum arguments.

Returns
lime.Spectrum

lime.Spectrum.**unit_conversion**(*self*, *wave_units_out=None*, *flux_units_out=None*, *norm_flux=None*)

This function converts spectrum wavelength array, the flux array or both arrays units.

The user can also provide a flux normalization for the spectrum flux array.

The wavelength units available are AA (angstroms), um, nm, Hz, cm, mm

The flux units available are Flam (erg s^-1 cm^-2 Å^-1), Fnu (erg s^-1 cm^-2 Hz^-1), Jy, mJy, nJy

Parameters

- **wave_units_out** (*str, optional*) – Wavelength array units
- **flux_units_out** (*str, optional*) – Flux array units
- **norm_flux** (*float, optional*) – Flux normalization

class lime.**Cube**(*input_wave=None*, *input_flux=None*, *input_err=None*, *redshift=None*, *norm_flux=None*, *crop_waves=None*, *inst_FWHM=nan*, *units_wave='AA'*, *units_flux='FLAM'*, *pixel_mask=None*, *id_label=None*, *wcs=None*)

This class creates an astronomical cube for an integral field spectrograph observation.

The user needs to provide 1D wavelength and 3D flux arrays. Additionally, the user can include a 3D flux uncertainty array. This uncertainty must be in the same units as the flux. The cube should include its `redshift`.

If the flux units result in very small magnitudes, the user should also provide a normalization to make the flux magnitude well above zero. Otherwise, the profile fittings are likely to fail. This normalization is removed in the output measurements.

The default `units_wave` are angtroms (Å), additional options are: um, nm, Hz, cm, mm

The default `units_flux` are Flam (erg s^-1 cm^-2 Å^-1), additional options are: Fnu, Jy, mJy, nJy

The user can also specify an instrument FWHM (`inst_FWHM`), so it can be taken into account during the measurements.

The user can provide a `pixel_mask` boolean 3D array with the pixels **to be excluded** from the measurements.

The observation object should include an astropy World Coordinate System (`wcs`) to export the spatial coordinate system to the measurement files.

Parameters

- **input_wave** (*numpy.array*) – wavelength 1D array
- **input_flux** (*numpy.array*) – flux 3D array
- **input_err** (*numpy.array, optional*) – flux sigma uncertainty 3D array.
- **redshift** (*float, optional*) – observation redshift.

- **norm_flux** (*float, optional*) – spectrum flux normalization
- **crop_waves** (*np.array, tuple, optional*) – spectrum (minimum, maximum) values
- **inst_FWHM** (*float, optional*) – Instrumental FWHM.
- **units_wave** (*str, optional*) – Wavelength units. The default value is "A"
- **units_flux** (*str, optional*) – Flux array physical units. The default value is "Flam"
- **pixel_mask** (*np.array, optional*) – Boolean 3D array with True values for rejected pixels.
- **id_label** (*str, optional*) – identity label for the spectrum object
- **wcs** (*astropy WCS, optional*) – Observation world coordinate system.

lime.Cube.**from_file**(*file_address*, *instrument*, *mask_flux_entries=None*, ***kwargs*)

This method creates a lime.Cube object from an observational (.fits) file. The user needs to introduce the file address location and the name of the instrument of survey.

Currently, this method supports MANGA and MUSE input instrument sources. This method will lower case the input instrument or survey name.

The user can include list of pixel values to generate a mask from the input file flux entries. For example, if the user introduces [np.nan, 'negative'] the output spectrum will mask np.nan entries and negative fluxes.

This method procures the instrument observations units, normalization and wcs but the user should introduce the LiMe.Spectrum arguments (such as the observation redshift).

Parameters

- **file_address** (*Path, string*) – Input file location address.
- **instrument** (*str*) – Input file instrument or survey name
- **mask_flux_entries** (*list*) – List of pixel values to mask from flux array
- **kwargs** – lime.Cube arguments.

Returns
lime.Cube

lime.Cube.**unit_conversion**(*self*, *units_wave=None*, *units_flux=None*, *norm_flux=None*)

This function converts cube wavelength array and/or the flux array units.

The user can also provide a flux normalization for the spectrum flux array.

The wavelength units available are A (angstroms), um, nm, Hz, cm, mm

The flux units available are Flam (erg s^-1 cm^-2 Å^-1), Fnu (erg s^-1 cm^-2 Hz^-1), Jy, mJy, nJy

Parameters

- **units_wave** (*str, optional*) – Wavelength array units
- **units_flux** (*str, optional*) – Flux array units
- **norm_flux** (*float, optional*) – Flux normalization

class lime.**Sample**(*sample_log*, *levels=('id', 'file', 'line')*, *load_function=None*, *instrument=None*, *folder_obs=None*, ***kwargs*)

This class creates a dictionary-like variable to store LiMe observations, by the fault it is assumed that these are `Spectrum` objects.

The sample is indexed via the input `log` parameter, a pandas dataframe, whose levels must be declared via the `levels` parameter. By default, three levels are assumed: an "id" column and a "file" column specifying the

object ID and observation file address respectively. The "line" level refers to the label measurements in the corresponding The user can specify more levels via the `levels` parameter. However, it is recommended to keep this structure: "id" and "file" levels first and the "line" column last.

To create the LiMe observation variables (`Spectrum` or `Cube`) the user needs to specify a `load_function`. This is a python method which declares how the observational files are read and parsed and returns a LiMe object. This `load_function` must have 4 parameters: `log_df`, `obs_idx`, `folder_obs` and `**kwargs`.

The first and second variable represent the sample `log` and a single pandas multi-index entry for the requested observation. The `folder_obs` and `**kwargs` are provided at the `Sample` creation:

The `folder_obs` parameter specifies the root file location for the targeted observation file. This root address is combined with the corresponding log level `file` value. If a `folder_obs` is not specified, it is assumed that the `file` log column contains the absolute file address.

The `**kwargs` argument specifies keyword arguments used in the creation of the `Spectrum` or `Cube` objects such as the `redshift or `norm_flux` for example.

The user may also specify the instrument used for the observation. In this case LiMe will use the inbuilt functions to read the supported instruments. This, however, may not contain all the necessary information to create the LiMe variable (such as the redshift). In this case, the user can include a load_function which returns a dictionary with observation parameters not found on the ".fits" file.

Parameters

- **sample_log** (*pd.Dataframe*) – multi-index dataframe with the parameter properties belonging to the `Sample`.
- **levels** (*list*) – levels for the sample log dataframe. By default, these levels are "id", "file", "line".
- **load_function** (*python method*) – python method with the instructions to convert the observation file into a LiMe observation.
- **instrument** (*string, optional.*) – instrument name responsible for the sample observations.
- **folder_obs** (*string, optional.*) – Root address for the observations' location. This address is combined with the "file" log column value.
- **kwargs** – Additional keyword arguments for the creation of the LiMe observation variables.

lime.Sample.**from_file**(*id_list*, *log_list=None*, *file_list=None*, *page_list=None*, *levels=('id', 'file', 'line')*, *load_function=None*, *instrument=None*, *folder_obs=None*, ***kwargs*)

This class creates a dictionary-like variable to store LiMe observations taking a list of observations IDs, line logs and a list of files.

The sample is indexed via the input `log` parameter, a pandas dataframe, whose levels must are declared via the `levels` parameter. By default, three levels are assumed: an "id" column and a "file" column specifying the object ID and observation file address respectively. The "line" level refers to the label measurements in the corresponding The user can specify more levels via the `levels` parameter. However, it is recommended to keep this structure: "id" and "file" levels first and the "line" column last.

The sample log levels are created from the input values for the `id_list`, `log_list` and `file_list` while the individual logs from each observation are combined where the line labels in the "line" level. If the input logs are ".fits" files the user must specify extension name or number via the `page_list` parameter.

To create the LiMe observation variables (`Spectrum` or `Cube`) the user needs to specify a `load_function`. This is a python method which declares how the observational files are read and parsed and returns a LiMe object. This `load_function` must have four parameters: `log_df`, `obs_idx`, `folder_obs` and `**kwargs`.

The first and second variable represent the sample `log` and a single pandas multi-index entry for the requested observation. The `folder_obs` and `**kwargs` are provided at the `Sample` creation:

The `folder_obs` parameter specifies the root file location for the targeted observation file. This root address is combined with the corresponding log level `file` value. If a `folder_obs` is not specified, it is assumed that the `file` log column contains the absolute file address. This is

The `**kwargs` argument specifies keyword arguments used in the creation of the `Spectrum` or `Cube` objects such as the `redshift or `norm_flux for example.

Parameters

- **id_list** (*list*) – List of observation names
- **log_list** (*list*) – List of observation log data frames or files or pandas data frames
- **file_list** (*list*) – List of observation files.
- **page_list** (*list*) – List of extension files or names for the observation ".fits" files
- **levels** (*list*) – levels for the sample log dataframe. By default, these levels are "id", "file", "line".
- **load_function** (*python method*) – python method with the instructions to convert the observation file into a LiMe observation.
- **instrument** (*string, optional.*) – instrument name responsible for the sample observations.
- **folder_obs** (*string, optional.*) – Root address for the observations' location. This address is combined with the "file" log column value.
- **kwargs** – Additional keyword arguments for the creation of the LiMe observation variables.

2.4 Fitting

lime.workflow.SpecTreatment.**bands**(*self*, *label*, *bands=None*, *fit_conf=None*, *min_method='least_squares'*, *profile='g-emi'*, *cont_from_bands=True*, *temp=10000.0*, *id_conf_prefix=None*, *default_conf_prefix='default'*)

This function fits a line on the spectrum object from a given band.

The first input is the line `label`. The user can provide a string with the default LiMe notation. Otherwise, the user can provide the transition wavelength in the same units as the spectrum and the transition will be queried from the `bands` argument.

The second input is the line `bands` this argument can be a six value array with the same units as the spectrum wavelength specifying the line position and continua location. Otherwise, the `bands` can be a pandas dataframe (or the frame address) and the wavelength array will be automatically query from it.

If the `bands` are not provided by the user, the default bands database will be used. You can learn more on the bands documentation.

The third input is a dictionary the fitting configuration `fit_conf` attribute. You can learn more on the profile fitting documentation.

The `min_method` argument provides the minimization algorithm for the LmFit functions.

By default, the profile fitting assumes an emission Gaussian shape, with `profile="g-emi"`. The profile keywords are described on the label documentation

The cont_from_bands=True argument forces the continuum to be measured from the adjacent line bands. If cont_from_bands=False the continuum gradient is calculated from the first and last pixel from the line band (w3-w4)

For the calculation of the thermal broadening on the emission lines the user can include the line electron temperature in Kelvin. The default value temp is 10000 K.

Parameters

- **label** (*str, float, optional*) – Line label or wavelength transition to be queried on the bands dataframe.
- **bands** (*np.array, pandas.dataframe, str, Path, optional*) – Bands six-value array, bands dataframe (or file address to the dataframe).
- **fit_conf** (*dict, optional*) – Fitting configuration.
- **min_method** (*str, optional*) – Minimization algorithm. The default value is 'least_squares'
- **profile** (*str, optional*) – Profile type for the fitting. The default value g-emi (Gaussian-emission).
- **cont_from_bands** (*bool, optional*) – Check for continuum calculation from adjacent bands. The default value is True.
- **temp** (*bool, optional*) – Transition electron temperature for thermal broadening calculation. The default value is 10000K.
- **default_conf_prefix** (*str, optional*) – Label for the default configuration section in the `fit_conf variable.
- **id_conf_prefix** (*str, optional*) – Label for the object configuration section in the `fit_conf variable.

lime.workflow.SpecTreatment.**frame**(*self*, *bands*, *fit_conf=None*, *min_method='least_squares'*, *profile='g-emi'*, *cont_from_bands=True*, *temp=10000.0*, *line_list=None*, *default_conf_prefix='default'*, *id_conf_prefix=None*, *line_detection=False*, *plot_fit=False*, *progress_output='bar'*)

This function measures multiple lines on the spectrum object from a bands dataframe.

The input bands_df can be a pandas.Dataframe or a link to its file.

The argument fit_conf provides the profile-fitting configuration.

The min_method argument provides the minimization algorithm for the LmFit functions.

By default, the profile fitting assumes an emission Gaussian shape, with profile="g-emi". The profile keywords are described on the label documentation

The cont_from_bands=True argument forces the continuum to be measured from the adjacent line bands. If cont_from_bands=False the continuum gradient is calculated from the first and last pixel from the line band (w3-w4).

For the calculation of the thermal broadening on the emission lines the user can include the line electron temperature in Kelvin. The default value temp is 10000 K.

The user can limit the fitting to certain bands with the lines_list argument.

If the input fit_conf has multiple sections, this function will read the parameters from the default_conf_key argument, whose default value is "default". If the input dictionary also has a section title with the id_conf_label _line_fitting the default_conf_key _line_fitting parameters will be **updated** by the object configuration.

If `line_detection=True` the input `bands_df` measurements will be limited to those bands with a line detection. The local configuration for the line detection algorithm can be provided from the fit_conf entries.

If `plot_fit=True` this function will plot profile after each fitting.

The `progress_output` argument determines the progress console message. A "bar" value will show a progress bar, while a "counter" value will print a message with the current line being measured. Finally, a None value will not show any message.

Parameters

- **bands** (*pandas.Dataframe, str, path.Pathlib*) – Bands dataframe (or file address to the dataframe).
- **fit_conf** (*dict, optional*) – Fitting configuration.
- **min_method** (*str, optional*) – Minimization algorithm. The default value is 'least_squares'
- **profile** (*str, optional*) – Profile type for the fitting. The default value `g-emi` (Gaussian-emission).
- **cont_from_bands** (*bool, optional*) – Check for continuum calculation from adjacent bands. The default value is True.
- **temp** (*bool, optional*) – Transition electron temperature for thermal broadening calculation. The default value is 10000K.
- **line_list** (*list, optional*) – Line list to measure from the bands dataframe.
- **default_conf_prefix** (*str, optional*) – Label for the default configuration section in the `` `fit_conf `` variable.
- **id_conf_prefix** (*str, optional*) – Label for the object configuration section in the `` `fit_conf `` variable.
- **line_detection** (*bool, optional*) – Set to True to run the dectection line algorithm prior to line measurements.
- **plot_fit** (*bool, optional*) – Set to True to plot the profile fitting at each iteration.
- **progress_output** (*str, optional*) – Progress message output. The options are "bar" (default), "counter" and "None".

`lime.workflow.SpecTreatment.`**continuum**(*self*, *degree_list*, *threshold_list*, *smooth_length=None*, *plot_steps=False*)

This function fits the spectrum continuum in an iterative process. The user specifies two parameters: the `degree_list` for the fitted polynomial and the `threshold_list`` for the multiplicative standard deviation factor. At each interation points beyond this flux threshold are excluded from the continuum current fittings. Consequently, the user should aim towards more constrictive parameter values at each iteration.

The user can specify a window length over which the spectrum will be smoothed before fitting the continuum using the `smooth_length` parameter.

The user can visually inspect the fitting output graphically setting the parameter `plot_steps=True`.

Parameters

- **degree_list** (*list*) – Integer list with the degree of the continuum polynomial
- **threshold_list** (*list*) – Float list for the multiplicative continuum standard deviation flux factor
- **smooth_length** (*integer, optional*) – Size of the smoothing window to convolve the spectrum. The default value is None.

- **plot_steps** (*bool, optional*) – Set to "True" to plot the fitted continuum at each iteration.

Returns

lime.workflow.CubeTreatment.**spatial_mask**(*self*, *mask_file*, *output_address*, *bands=None*, *fit_conf=None*, *mask_list=None*, *line_list=None*, *log_ext_suffix='_LINELOG'*, *min_method='least_squares'*, *profile='g-emi'*, *cont_from_bands=True*, *temp=10000.0*, *default_conf_prefix='default'*, *line_detection=False*, *progress_output='bar'*, *plot_fit=False*, *header=None*, *join_output_files=True*)

This function measures lines on an IFS cube from an input binary spatial `mask_file`.

The results are stored in a multipage ".fits" file, each page contains a measurements and it is named after the spatial array coordinates and the `log_ext_suffix` (i.e. "idx_j-idx_i_LINELOG")

The input `bands` can be a pandas.Dataframe or an address to the file. The user can specify one bands file per mask page on the `mask_file`. To do this, the `fit_conf` argument must include a section for every mask on the `mask_list` (i.e. "Mask1_line_fitting"). This function will check for a key "bands" and load the corresponding bands.

The fitting configuration in the `fit_conf` argument accepts a three-level configuration. At the lowest level, The `default_conf_key` points towards the default configuration for all the spaxels analyzed on the cube (i.e. "default_line_fitting"). At an intermediate level, the parameters from the section with a name from the `mask_list` (i.e. "Mask1_line_fitting") will be applied to the spaxels in the corresponding mask. Finally, at the highest level, the user can provide a spaxel fitting configuration with the spatial array coordiantes "50-28_LINELOG". In all these cases the higher level configurate **updates** the lower levels (only common entries are replaced)

Attention: In this multi-level configuration design, the higher level entries **update** the lower level entries: only shared entries are overwritten, the final configuration will include all the entries from the default mask and spaxel sections.

If the `line_detection` is set to True the function proceeds to run the line detection algorithm prior to the fitting of the lines. The user provide the configuration parameters for the line_detection function in the `fit_conf` argument. At the default, mask or spaxel configuration the user needs to specify these entries with the "function name" + "." + "function argument" (i.e. "line_detection.line_type='emission'"). The multi-level configuration described above will be applied to this function parameters as well.

Note: The parameters for the `line.detection` can be found on the documentation. The user doesn't need to specify a "lime_detection.bands" parameter. The input bands from the corresponding mask will be used.

Parameters

- **mask_file** (*str, pathlib.Path*) – Address of binary spatial mask file
- **output_address** (*str, pathlib.Path*) – File address for the output measurements log.
- **bands** (*pandas.Dataframe, str, path.Pathlib*) – Bands dataframe (or file address to the dataframe).
- **fit_conf** (*dict, optional*) – Fitting configuration.
- **mask_list** (*list, optional*) – Masks name list to explore on the `masks_file`.
- **line_list** (*list, optional*) – Line list to measure from the bands dataframe.

- **log_ext_suffix** (*str, optional.*) – Suffix for the measurements log pages. The default value is "_LINELOG".
- **min_method** (*str, optional*) – Minimization algorithm. The default value is 'least_squares'
- **profile** (*str, optional*) – Profile type for the fitting. The default value `g-emi` (Gaussian-emission).
- **cont_from_bands** (*bool, optional*) – Check for continuum calculation from adjacent bands. The default value is True.
- **temp** (*bool, optional*) – Transition electron temperature for thermal broadening calculation. The default value is 10000K.
- **default_conf_prefix** (*str, optional*) – Label for the default configuration section in the `fit_conf variable.
- **line_detection** (*bool, optional*) – Set to True to run the dectection line algorithm prior to line measurements.
- **plot_fit** (*bool, optional*) – Set to True to plot the spectrum lines fitting at each iteration.
- **progress_output** (*str, optional*) – Progress message output. The options are "bar" (default), "counter" and "None".
- **header** (*dict, optional*) – Dictionary for parameter ".fits" file headers.
- **join_output_files** (*bool, optional*) – In the case of multiple masks, join the individual output ".fits" files into a single one. If set to False there will be one output file named per mask named after it. The default value is True.

2.5 Plotting

lime.plots.SpectrumFigures.**spectrum**(*self*, *output_address=None*, *label=None*, *line_bands=None*, *rest_frame=False*, *log_scale=False*, *include_fits=True*, *include_cont=False*, *in_fig=None*, *fig_cfg={}*, *ax_cfg={}*, *maximize=False*)

This function plots the spectrum flux versus wavelength.

The user can include the line bands on the plot if added via the `line_bands` attribute.

The user can provide a label for the spectrum legend via the `label` argument.

If the user provides an `output_address` the plot will be stored into an image file instead of being displayed into a window.

If the user has installed the library mplcursors, a left-click on a fitted profile will pop-up properties of the fitting, right-click to delete the annotation. This requires `include_fits=True`.

By default, this function creates a matplotlib figure and axes set to plot the data. However, the user can provide their own `in_fig` to plot the data. This will return the data-plotted figure object.

The default axes and plot titles can be modified via the `ax_cfg`. These dictionary keys are "xlabel", "ylabel" and "title". It is not necessary to include all the keys in this argument.

Parameters

- **output_address** (*str, optional*) – File location to store the plot.

- **label** (*str, optional*) – Label for the spectrum plot legend. The default label is 'Observed spectrum'.
- **line_bands** (*pd.Dataframe, str, path, optional*) – Bands Dataframe (or path to dataframe).
- **rest_frame** (*bool, optional*) – Set to True for a display in rest frame. The default value is False
- **log_scale** (*bool, optional*) – Set to True for a display with a logarithmic scale flux. The default value is False
- **include_fits** (*bool, optional*) – Set to True to display fitted profiles. The default value is False.
- **include_cont** (*bool, optional*) – Set to True to display fitted continuum. The default value is False.
- **fig_cfg** (*dict, optional*) – Matplotlib RcParams parameters for the figure format
- **ax_cfg** (*dict, optional*) – Dictionary with the plot "xlabel", "ylabel" and "title" values.
- **in_fig** (*matplotlib.figure*) – Matplotlib figure object to plot the data.
- **maximize** (*bool, optional*) – Maximise plot window. The default value is False.

lime.plots.SpectrumFigures.**bands**(*self*, *line=None*, *bands=None*, *output_address=None*, *include_fits=True*, *rest_frame=False*, *y_scale='auto'*, *in_fig=None*, *fig_cfg={}*, *ax_cfg={}*, *maximize=False*)

This function plots a spectrum `line`. If a `line` is not provided the function will select the last line from the measurements log.

The user can also introduce a `bands` dataframe (or its file path) to query the input `line`.

If the user provides an `output_address` the plot will be stored into an image file instead of being displayed in a window.

The `y_scale` argument sets the flux scale for the lines grid. The default "auto" value automatically switches between the matplotlib scale keywords, otherwise the user can set a uniform scale for all.

The default axes and plot titles can be modified via the `ax_cfg`. These dictionary keys are "xlabel", "ylabel" and "title". It is not necessary to include all the keys in this argument.

Parameters

- **line** (*str, optional*) – Line label to display.
- **bands** (*np.array, pandas.Dataframe, str, path.pathlib, optional*) – Bands array or dataframe (or its file) to display.
- **output_address** (*str, optional*) – File location to store the plot.
- **include_fits** (*bool, optional*) – Set to True to display fitted profiles. The default value is False.
- **rest_frame** (*bool, optional*) – Set to True for a display in rest frame. The default value is False
- **y_scale** (*str, optional.*) – Matplotlib scale keyword. The default value is "auto".
- **in_fig** (*matplotlib.figure*) – Matplotlib figure object to plot the data.
- **fig_cfg** (*dict, optional*) – Dictionary with the matplotlib rcParams parameters .
- **ax_cfg** (*dict, optional*) – Dictionary with the plot "xlabel", "ylabel" and "title" values.

- **maximize** (*bool, optional*) – Maximise plot window. The default value is False.

Returns

lime.plots.SpectrumFigures.**grid**(*self*, *output_address=None*, *rest_frame=True*, *y_scale='auto'*, *n_cols=6*, *n_rows=None*, *col_row_scale=(2, 1.5)*, *include_fits=True*, *in_fig=None*, *fig_cfg={}*, *ax_cfg={}*, *maximize=False*)

This function plots the lines from the object spectrum log as a grid.

If the user has installed the library mplcursors, a left-click on a fitted profile will pop-up properties of the fitting, right-click to delete the annotation.

If the user provides an `output_address` the plot will be stored into an image file instead of being displayed into a window.

The default axes and plot titles can be modified via the `ax_cfg`. These dictionary keys are "xlabel", "ylabel" and "title". It is not necessary to include all the keys in this argument.

By default, this function creates a matplotlib figure and axes set to plot the data. However, the user can provide their own `in_fig` to plot the data. This will return the data-plotted figure object.

Parameters

- **output_address** (*str, pathlib.Path, optional*) – Image file address for plot.
- **rest_frame** (*bool, optional*) – Set to True to plot the spectrum to rest frame. Optional False.
- **y_scale** (*str, optional.*) – Matplotlib scale keyword. The default value is "auto".
- **n_cols** (*int, optional.*) – Number of columns in plot grid. The default value is 6.
- **n_rows** (*int, optional.*) – Number of rows in plot grid.
- **col_row_scale** (*tuple, optional.*) – Multiplicative factor for the grid plots width and height. The default value is (2, 1.5).
- **include_fits** (*bool, optional*) – Set to True to display fitted profiles. The default value is False.
- **fig_cfg** (*dict, optional*) – Matplotlib RcParams parameters for the figure format
- **ax_cfg** (*dict, optional*) – Dictionary with the plot "xlabel", "ylabel" and "title" values.
- **maximize** (*bool, optional*) – Maximise plot window. The default value is False.

lime.plots.CubeFigures.**cube**(*self*, *line*, *bands=None*, *line_fg=None*, *output_address=None*, *min_pctl_bg=60*, *cont_pctls_fg=(90, 95, 99)*, *bg_cmap='gray'*, *fg_cmap='viridis'*, *bg_norm=None*, *fg_norm=None*, *masks_file=None*, *masks_cmap='viridis_r'*, *masks_alpha=0.2*, *wcs=None*, *fig_cfg=None*, *ax_cfg=None*, *in_fig=None*, *maximise=False*)

This function plots the map of a flux band sum for a cube integral field unit observation.

The `line` argument provides the label for the background image. Its bands are read from the `bands` argument dataframe. If none is provided, the default lines database will be used to query the bands. Similarly, if the user provides a foreground `line_fg` the plot will include intensity contours from its corresponding band.

The user can provide a map baground and foreground contours matplotlib color normalization. Otherwise, a logarithmic normalization will be used.

If the user does not provide a color normalizations at `bg_norm` and `fg_norm`. A logarithmic normalization will be used. In this scenario `min_pctl_bg` establishes the minimum flux percentile flux for the background image. The number and separation of flux foreground contours is calculated from the sequence in the `cont_pctls_fg`.

If the user provides the address to a binary fits file to a mask file, this will be overploted on the map as shaded pixels.

Parameters

- **line** (*str*) – Line label for the spatial map background image.
- **bands** (*pandas.Dataframe, str, path.Pathlib, optional*) – Bands dataframe (or file address to the dataframe).
- **line_fg** (*str, optional*) – Line label for the spatial map background image contours
- **output_address** (*str, optional*) – File location to store the plot.
- **min_pctl_bg** (*float, optional*) – Minimum band flux percentile for spatial map background image. The default value is 60.
- **cont_pctls_fg** (*tuple, optional*) – Band flux percentiles for foreground `line_fg` contours. The default value is (90, 95, 99)
- **bg_cmap** (*str, optional*) – Background image flux color map. The default value is "gray".
- **fg_cmap** (*str, optional*) – Foreground image flux color map. The default value is "viridis".
- **bg_norm** (*Normalization from matplotlib.colors, optional*) – Background image color normalization. The default value is SymLogNorm.
- **fg_norm** (*Normalization from matplotlib.colors, optional*) – Foreground contours color normalization. The default value is LogNorm.
- **masks_file** (*str, optional*) – File address for binary spatial masks
- **masks_cmap** (*str, optional*) – Binary masks color map.
- **masks_alpha** (*float, optional*) – Transparency alpha value. The default value is 0.2 (0 to 1 scale).
- **wcs** (*astropy WCS, optional*) – Observation world coordinate system.
- **fig_cfg** (*dict, optional*) – Matplotlib RcParams parameters for the figure format
- **ax_cfg** (*dict, optional*) – Dictionary with the plot "xlabel", "ylabel" and "title" values.
- **in_fig** (*matplotlib.figure*) – Matplotlib figure object to plot the data.
- **maximize** (*bool, optional*) – Maximise plot window. The default value is False.

2.6 Interactive plotting

lime.plots_interactive.BandsInspection.**bands**(*self*, *bands_file*, *ref_bands=None*, *y_scale='auto'*, *n_cols=6*, *n_rows=None*, *col_row_scale=(2, 1.5)*, *z_log_address=None*, *object_label=None*, *z_column='redshift'*, *fig_cfg={}*, *ax_cfg={}*, *in_fig=None*, *maximize=False*)

This function launches an interactive plot from which to select the line bands on the observed spectrum. If this function is run a second time, the user selections won't be overwritten.

The `bands_file` argument provides to the output database on which the user selections will be saved.

The `ref_bands` argument provides the reference database. The default database will be used if none is provided.

The `y_scale` sets the flux scale for the lines grid. The default "auto" value automatically switches between the matplotlib scale keywords, otherwise the user can set a uniform scale for all.

If the user wants to adjust the observation redshift and save the new values the `z_log_address` sets an output dataframe. The `object_label` and `z_column` provide the row and column indexes to save the new redshift.

The default axes and plot titles can be modified via the `ax_cfg`. These dictionary keys are "xlabel", "ylabel" and "title". It is not necessary to include all the keys in this argument.

The online documentation provides more details on the mechanics of this plot function.

Parameters

- **bands_file** (*str, pathlib.Path*) – Output file address for user bands selection.
- **ref_bands** (*pandas.Dataframe, str, pathlib.Path, optional*) – Reference bands dataframe or its file address. The default database will be used if none is provided.
- **y_scale** (*str, optional.*) – Matplotlib scale keywords. The default value is "auto".
- **n_cols** (*int, optional.*) – Number of columns in plot grid. The default value is 6.
- **n_rows** (*int, optional.*) – Number of rows in plot grid.
- **col_row_scale** (*tuple, optional.*) – Multiplicative factor for the grid plots width and height. The default value is (2, 1.5).
- **z_log_address** (*str, pathlib.Path, optional*) – Output address for redshift dataframe file.
- **object_label** (*str, optional*) – Object label for redshift dataframe row indexing.
- **z_column** (*str, optional*) – Column label for redshift dataframe column indexing. The default value is "redshift".
- **fig_cfg** (*dict, optional*) – Dictionary with the matplotlib rcParams parameters .
- **ax_cfg** (*dict, optional*) – Dictionary with the plot "xlabel", "ylabel" and "title" values.
- **maximize** (*bool, optional*) – Maximise plot window. The default value is False.

lime.plots_interactive.CubeInspection.**cube**(*self*, *line*, *bands=None*, *line_fg=None*, *min_pctl_bg=60*, *cont_pctls_fg=(90, 95, 99)*, *bg_cmap='gray'*, *fg_cmap='viridis'*, *bg_norm=None*, *fg_norm=None*, *masks_file=None*, *masks_cmap='viridis_r'*, *masks_alpha=0.2*, *rest_frame=False*, *log_scale=False*, *fig_cfg={}*, *ax_cfg_image={}*, *ax_cfg_spec={}*, *in_fig=None*, *lines_log_file=None*, *ext_log='_LINELOG'*, *wcs=None*, *maximize=False*)

This function opens an interactive plot to display and individual spaxel spectrum from the selection on the image map.

The left-hand side plot displays an image map with the flux sum of a line band as described on the Cube.plot.cube documentation.

A right-click on a spaxel of the band image map will plot the selected spaxel spectrum on the right-hand side plot. This will also mark the spaxel with a red cross.

If the user provides a `masks_file` the plot window will include a dot mask selector. Activating one mask will overplotted on the image band. A middle button click on the image band will add/remove a spaxel to the current pixel selected masks. If the spaxel was part of another mask it will be removed from the previous mask region.

If the user provides a `lines_log_file` .fits file, the fitted profiles will be included on its corresponding spaxel spectrum plot. The measurements logs on this ".fits" file must be named using the spaxel array coordinate and the suffix on the `ext_log` argument.

If the user has installed the library mplcursors, a left-click on a fitted profile will pop-up properties of the fitting, right-click to delete the annotation.

Parameters

- **line** (*str*) – Line label for the spatial map background image.
- **bands** (*pandas.Dataframe, str, path.Pathlib, optional*) – Bands dataframe (or file address to the dataframe).
- **line_fg** (*str, optional*) – Line label for the spatial map background image contours
- **min_pctl_bg** (*float, optional*) – Minimum band flux percentile for spatial map background image. The default value is 60.
- **cont_pctls_fg** (*tuple, optional*) – Band flux percentiles for foreground `line_fg` contours. The default value is (90, 95, 99)
- **bg_cmap** (*str, optional*) – Background image flux color map. The default value is "gray".
- **fg_cmap** (*str, optional*) – Foreground image flux color map. The default value is "viridis".
- **bg_norm** (*Normalization from matplotlib.colors, optional*) – Background image color normalization. The default value is SymLogNorm.
- **fg_norm** (*Normalization from matplotlib.colors, optional*) – Foreground contours color normalization. The default value is LogNorm.
- **masks_file** (*str, optional*) – File address for binary spatial masks
- **masks_cmap** (*str, optional*) – Binary masks color map.
- **masks_alpha** (*float, optional*) – Transparency alpha value. The default value is 0.2 (0 to 1 scale).
- **rest_frame** (*bool, optional*) – Set to True for a display in rest frame. The default value is False
- **log_scale** (*bool, optional*) – Set to True for a display with a logarithmic scale flux. The default value is False
- **fig_cfg** (*dict, optional*) – Matplotlib RcParams parameters for the figure format
- **ax_cfg_image** (*dict, optional*) – Dictionary with the image band flux plot "xlabel", "ylabel" and "title" values.
- **ax_cfg_spec** (*dict, optional*) – Dictionary with the spaxel spectrum plot "xlabel", "ylabel" and "title" values.
- **lines_log_file** (*str, pathlib.Path, optional*) – Address for the line measurements log ".fits" file.
- **ext_log** (*str, optional*) – Suffix for the line measurements log spaxel page name
- **wcs** (*astropy WCS, optional*) – Observation world coordinate system.
- **maximize** (*bool, optional*) – Maximise plot window. The default value is False.

[1]:
```
%matplotlib notebook
```

CHAPTER

THREE

SPECTRA

At the present, $LiMe$ does not have an inbuilt function to load spectra from a *.fits* files. Consequently, the user must write his/her own function to unpack the scientific data using numpy and astropy.

As a guideline for novel astronomers (or Python programmers), in this tutorial, we are going to visit some examples on how to open spectra from well-known telescope instruments. You can find this page as a notebook in the documentation/inputs folder.

Any user, however, is encouraged to read the FITS handling documentation carefully, to better manage their astronomical data.

Independent of the observational range, there are a few things which could **dramatically** affect the quality of your measurements:

3.1 Flux normalization:

In many cases, astronomical spectra is in C.G.S (Centimetre–gram–second) units. This translates into a flux scale with very small numbers. This will make you fittings fail. If you input spectrum is not normalized it is essential you include a `norm_flux` in your `lime.Spectrum` and `lime.Cube` definition.

3.2 Wavelength units:

As in the case above, wavelength units (such as μm) may translate into pixel resolution steps ($\Delta\lambda$) with very small magnitues. Your are recomended to use the `.convert_units` attribute.

3.3 Redshift:

$LiMe$ measurements are **performed and reported in the observed frame**. However, the default database and line labels are stored in the rest frame. This design was favoured to guarantee a uniform workflow in observational samples with different redshift values. Consequently, you are strongly encouraged to provide a `redshift`.

3.3.1 1) Long-slit spectra

The reduced spectrum of a long-slit or echelle instrument, consists in a container for the photons flux and dispersion. The first array is stored directly on a file page but the latter, depending on the instrument, may need to be reconstructed from the page header keys.

a) OSIRIS instrument at the Gran Telescopio de Canarias

The long-slit spectra from this instruments follows the standard reduction using IRAF. Consequently, a page the wavelength array is computed from the `CRVAL1`, `CD1_1` and `NAXIS1` keys.

Let's start by importing the libraries we are going to use and declare the folder with the scientific data:

[2]:
```
import numpy as np
from astropy.io import fits
from pathlib import Path
from astropy.wcs import WCS
import lime

# Sample data location
spectra_path = Path('../sample_data')
fits_path = spectra_path/'gp121903_osiris.fits'
extension = 0
```

The extension above specifies the page from the *.fits* file, from which we read the data. A quick way to learn about the the extensions data you can use the convenience function `fits.info`:

[3]:
```
# Fits file information:
fits.info(fits_path)
```

```
Filename: ..\sample_data\gp121903_osiris.fits
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU     143   (3199,)   float32
```

In this file, there is only one extension, so we can now open the file:

[4]:
```
# Open the fits file
with fits.open(fits_path) as hdul:
    flux_array, header = hdul[extension].data, hdul[extension].header
```

The code above we are using the with context manager. This structure might be a confusing for the novel user. However, this design makes sure that the file is closed once we extract the data. This is safe practice in the case of large *.fits* files.

Now, we are going to compute the wavelength array:

[5]:
```
# Reconstruct the wavelength array from the header data
w_min, dw, n_pix = header['CRVAL1'],  header['CD1_1'], header['NAXIS1']
w_max = w_min + dw * n_pix
wave_array = np.linspace(w_min, w_max, n_pix, endpoint=False)
print(wave_array)
```

```
[ 3626.97753906  3629.04561139  3631.11368372 ... 10236.5367069
 10238.60477923 10240.67285156]
```

In this file the `CRVAL1` provides the spectrum lower limit wavelength, `CD1_1` states the wavelength resolution (constant for this instrument) and the integer `NAXIS1` is the number of pixels along the dispersion axis.

Please remember: The keys in a **.fits** header are not fully standarised. Make sure to check the keys description on the header data and the instrument documentation to confirm you are loading the files properly.

The units on the spectrum arrays should also be specified on the *.fits* header:

[6]:
```
print(f'Wavelength array: {header["WAT1_001"]}')
print(f'Flux array: {header["BUNIT"]}')
```

```
Wavelength array: wtype=linear label=Wavelength units=Angstroms
Flux array: erg/cm2/s/A
```

Now we have all the data start analyzing this spectrum:

[7]:
```
obj = lime.Spectrum(wave_array, flux_array, redshift=0.19531, norm_flux=1e-17)
obj.plot.spectrum()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

b) SDSS spectrum (data release 18)

The *.fits* files from SDSS survey are indexed following a dictionary-like structure which includes the wavelength array.

[8]:
```
fits_path = spectra_path/'SHOC579_SDSS_dr18.fits'
```

To load the data we can use:

[9]:
```
# Open the fits file
extension = 1
with fits.open(fits_path) as hdul:
    data = hdul[extension].data
    header = hdul[extension].header
```

The `flux` key indexes the flux. This value is normalized by 10^{-17}:

[10]:
```
flux_array = data['flux'] * 1e-17
```

The flux uncertainty is stored as the inversed of the variance. The bad values in this array are set to zero, hence, we are going to mask these values to compute the error spectrum.

[11]:
```
ivar_array = data['ivar']
pixel_mask = ivar_array == 0
```

[12]:
```
err_array = np.sqrt(1/np.ma.masked_array(ivar_array, pixel_mask)) * 1e-17
```

Finally, the wavelength array is stored in logarithmic scale and they are accessed via the `loglam` key:

[13]:
```
wave_vac_array = np.power(10, data['loglam'])
```

This wavelength array is in vacuum. To convert it to air units we can use the Morton (1991, ApJS, 77, 119) law:

[14]:
```
wave_array = wave_vac_array / (1.0 + 2.735182E-4 + 131.4182 / wave_vac_array**2 + 2.
↪76249E8 / wave_vac_array**4)
```

Now we can create the Spectrum object:

[15]:
```
obj = lime.Spectrum(wave_array, flux_array, err_array, pixel_mask=pixel_mask, norm_
↪flux=1e-17, redshift=0.0475)
obj.plot.spectrum()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

3.3.2 2) IFU data sets

Integrated Field provide a cube data array, whose depthness edge corresponds to the dispersion axis. The other two axes provide the spatial coordinates. Depending of the instrument, the size of these files can be very large and 3rd party libraries are necessary to open such files.

a) MANGA survey cubes

As we did in the case of the SDSS survey, we are opening an observation from the SHOC579 galaxy, which we are also analysing in the 4th tutorial.

[16]:
```
fits_path = spectra_path/'manga-8626-12704-LOGCUBE.fits.gz'
```

This file is compressed, however, the `astropy.io.fits.open` can read compressed files. However, loading compressed file is slower than an uncompressed one:

[17]:
```
# Open the MANGA cube fits file
with fits.open(fits_path) as hdul:

    # Wavelength 1D array
    wave = hdul['WAVE'].data

    # Flux 3D array
    flux_cube = hdul['FLUX'].data * 1e-17

    # Convert inverse variance cube to standard error, masking 0-value pixels first
    ivar_cube = hdul['IVAR'].data
    pixel_mask_cube = ivar_cube == 0
    pixel_mask_cube = pixel_mask_cube.reshape(ivar_cube.shape)
    err_cube = np.sqrt(1/np.ma.masked_array(ivar_cube, pixel_mask_cube)) * 1e-17

    # Header
    hdr = hdul['FLUX'].header
```

The header of *.fits* spectra usually contains the astronomical coordinates of the observation. $LiMe$ relies on astropy World Coordidnate System function, to plot the IFU data and export the coordinates to the output *.fits* with your measurements:

You can create the WCS object from your information from the data on the observation header:

[18]:
```
# WCS from the observation
wcs = WCS(hdr)
```

```
WARNING: FITSFixedWarning: PLATEID = 8626 / Current plate
a string value was expected. [astropy.wcs.wcs]
WARNING: FITSFixedWarning: 'datfix' made the change 'Set MJD-OBS to 57277.000000 from␣
↪DATE-OBS'. [astropy.wcs.wcs]
```

Now, we have all the create our Cube object:

[19]:
```
cube = lime.Cube(wave, flux_cube, err_cube, redshift=0.0475, norm_flux=1e-17, pixel_
↪mask=pixel_mask_cube)
```

```
LiMe WARNING: Your wave array does not include a pixel mask this can caused issues on␣
↪the fittings
LiMe WARNING: Your wave_rest array does not include a pixel mask this can caused issues␣
↪on the fittings
```

Now you can extract individual spaxels to analyse the object lines. Each spaxel will have its corresponding error spectrum and pixel mask

[20]:
```
# Extract spaxel
spaxel = cube.get_spectrum(38, 35)

# Plot spectrum
spaxel.plot.spectrum(rest_frame=True)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

Finally, we can perform measurements on this spaxel using the `Spectrum` attributes

[21]:
```
# Fit emission line using the bands from the default database
spaxel.fit.bands('S3_6312A')

# Plot last fitting
spaxel.plot.bands()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

[1]:
```
%matplotlib notebook
```

CHAPTER

FOUR

LINE LABELS

The first input for $LiMe$ measurements is a label for the line being measured. This label follows a notation based on the PyNeb the emission analysis package by V. Luridiana, C. Morisset and R. A. Shaw (2015).

The first element is the particle responsible for the transition while the second element is the wavelength of the transition. This wavelength must include the units.

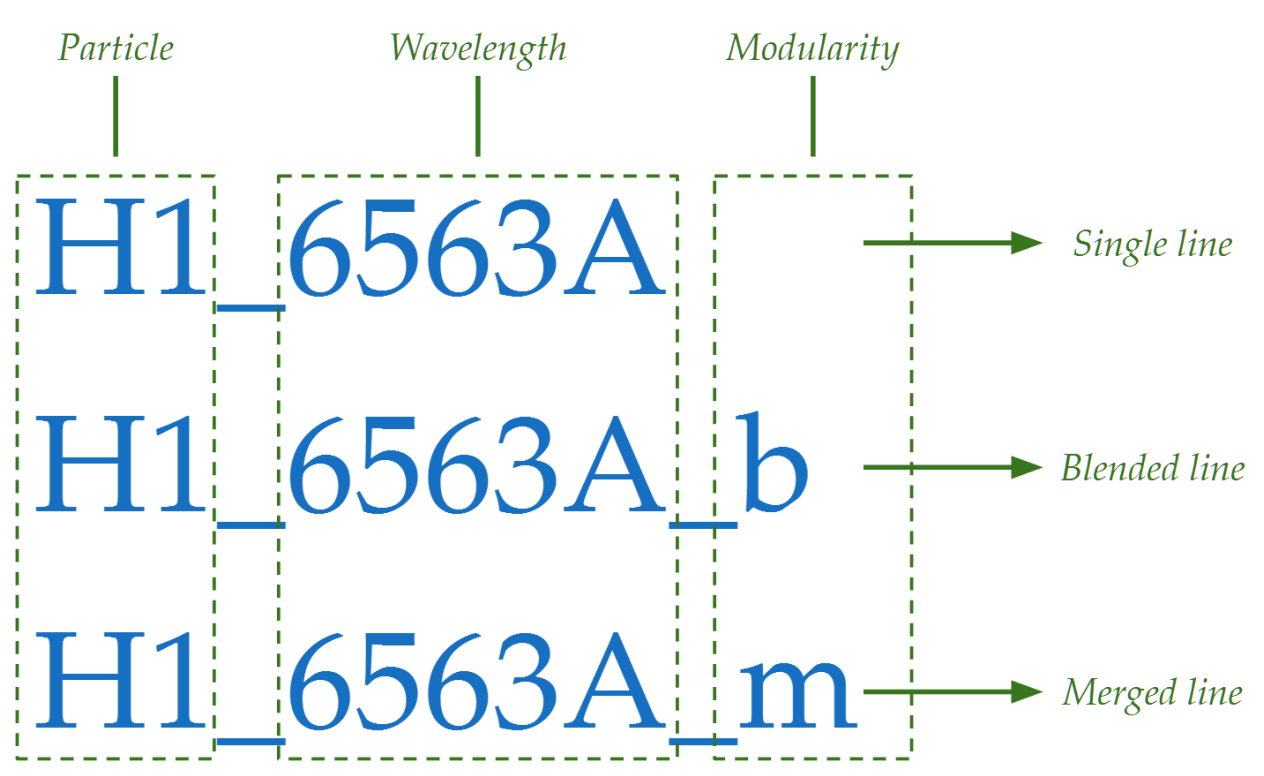
Additional suffixes provide additional information for the fitting or information regarding the transitions.

[2]:
```
# Import libraries for the example
from IPython.display import Image, display
import lime
```

4.1 Core components

These elements of the label are compulsary and their order is fixed.

[3]:
```
display(Image(filename='../images/label_core_components.png', width = 800))
```



4.1.1 Particle component

The first of the line label is the particle responsible for the transition:

- Ionized ions: These particles are specified using the corresponding element chemical symbol followed by the stage of ionization in stage of ionization using arabic numerals.

Please remember: The guidelines above are the recomended format for the particle component in $LiMe$ label notation. You can still type whatever particle name you want and it should not be an issue for your fittings (although you may miss some features).

4.1.2 Wavelength component

The second element of the line label is the transition wavelength. This element is arguable the most important for the fitting. This transition must be on the rest frame. It provides the following information during the fitting:

- Theoretical wavelength value for $LiMe$ line detection functions.
- Initial value for the Gaussian center for the line location.
- Transition wavelength units.

Regarding the latter item, the units that $LiMe$ can recogniced currently are:

[4]:
```
from lime.tools import UNITS_LATEX_DICT

print(f'(LiMe - LaTex)')
```

(continues on next page)

(continued from previous page)

```
for key, value in UNITS_LATEX_DICT.items():
    print(f'{key} =\t {value}')
```

```
(LiMe - LaTex)
A =      \AA
um =     \mu\!m
nm =     nm
Hz =     Hz
cm =     cm
mm =     mm
Flam =   erg\,cm^{-2}s^{-1}\AA^{-1}
Fnu =    erg\,cm^{-2}s^{-1}\Hz^{-1}
Jy =     Jy
mJy =    mJy
nJy =    nJy
```

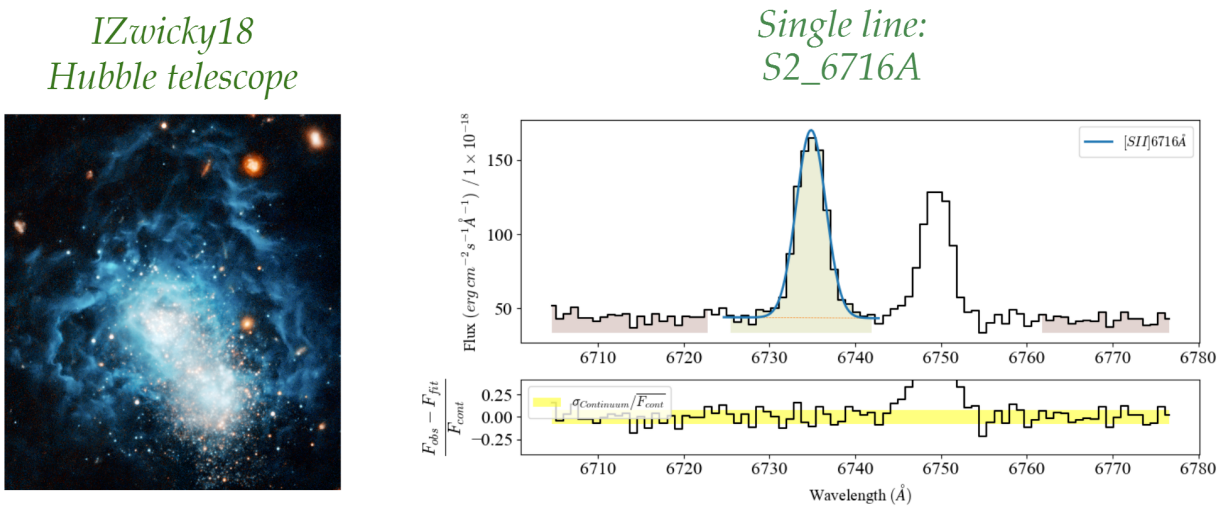
4.1.3 Modularity component

The final core component informs $LiMe$ if the line profile fitting consists in one or multiple Gaussian curves. This item must be at the end of the line label string. The following images provide and example of each with the observation of the $[SII]6716, 6731$ doublet:

Single line

In $LiMe$ a single line referst to an emission or absorption feature, with can be modelled with a single transition and fit with a single Gaussian curve:

[5]:
```
display(Image(filename='../images/SingleLine_diagram.png', width = 800))
```

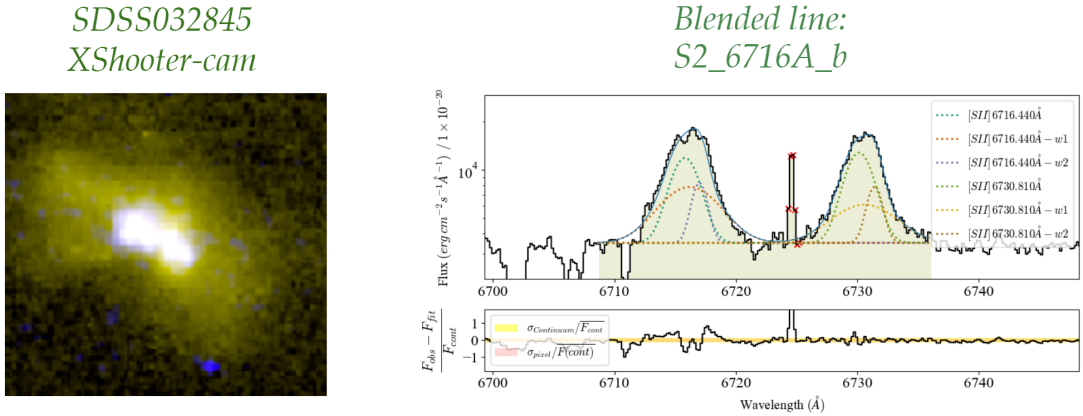


This is the default line format.

Blended line

A blended line is composed of multiple transtions and/or kinematic components. If the user adds the the *"_b"* suffix **and** includes the components in the fitting configuration (joined by "+"). $LiMe$ will proceed to fit one Gaussian profile per component.

[6]:
```
display(Image(filename='../images/BlendedLine_diagram.png', width = 800))
```



Configuration file entry:

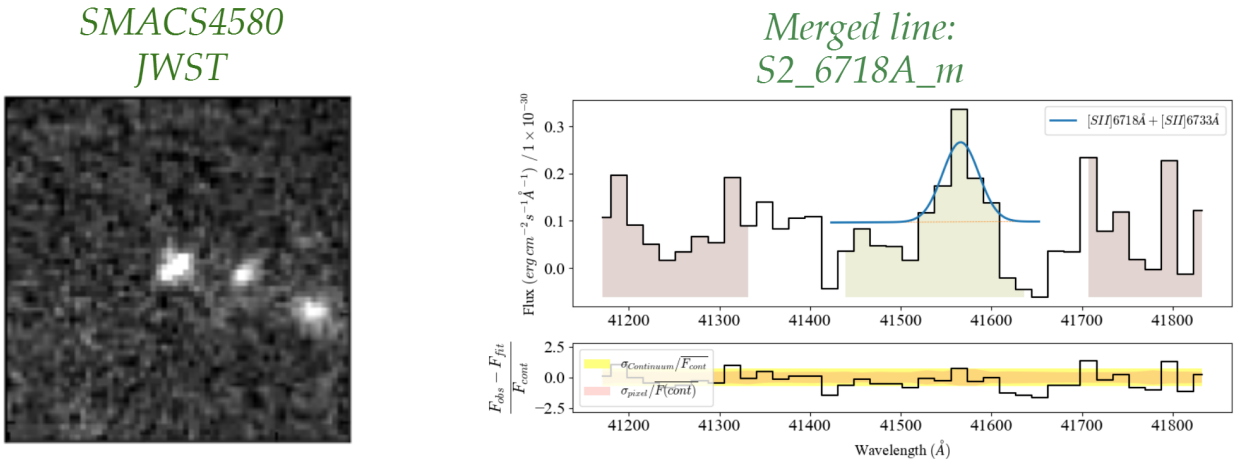
```
S2_6716A_b = S2_6716A+S2_6731A+S2_6716A_k-1+S2_6731A_k-1+S2_6716A_k-2+S2_6731A_k-2
```

In the example, we fit both $[SII]6716, 6731$ transitions, where each line has 2 additional kinematic components. These kinematic components must include the kinematic suffix (_k-1, _k-2, ...).

Merged line

A merged line assumes that there are multiple transition contributing to the observed line. The user adds the *"_m"* suffix **and** includes the components in the fitting configuration (joined by "+"). However, $LiMe$ will only fit one line:

[7]:
```
display(Image(filename='../images/MergedLine_diagram.png', width = 800))
```



Configuration file entry:

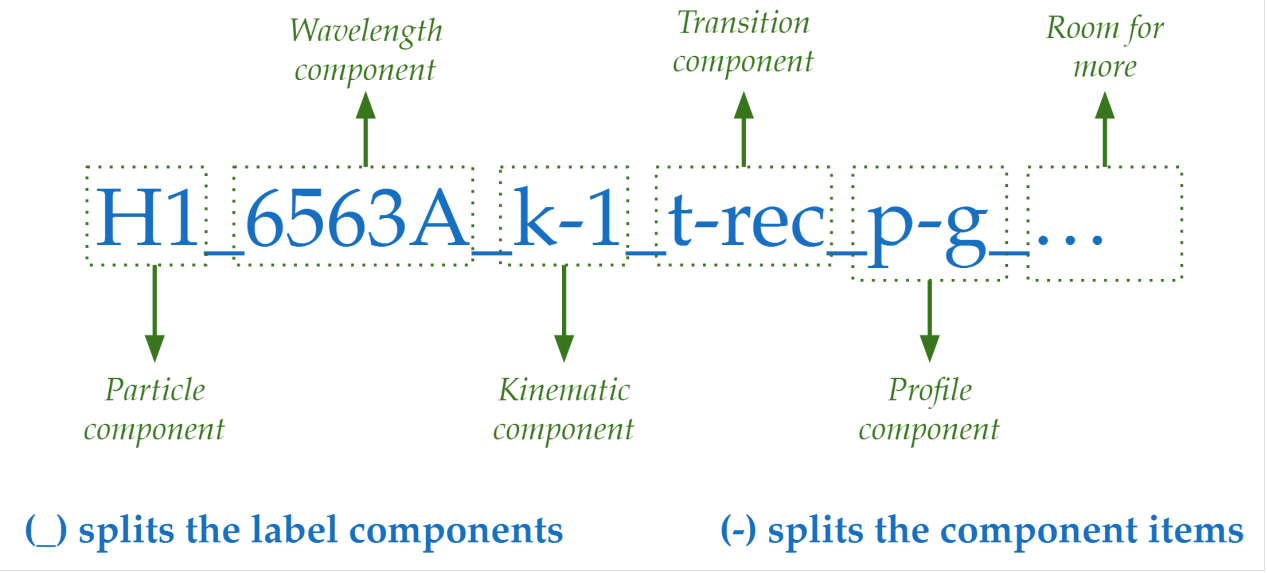
```
S2_6718A_m = S2_6718A+S2_6733A
```

This classification is usefull in those cases where the user wants to keep track of the transitions contributting to a line flux but the observational resolution is not good enought to isolate individual components. In the case above, we can see that the noise has devoured the $[SII]6716$ so we can only fitted the redder component. In the output log, this line will be saved as "S2_6718A_m" with the `profile_label=S2_6718A+S2_6733A` (since this observation was done in vacuum, the wavelength is slightly higher)

4.2 Optional components

Unlike in the previous case, the optional components have default values. This means that the user **can exclude them** from the label and they have a **free order**:

[8]:
```
display(Image(filename='../images/label_all_components.png', width = 800))
```



If the user includes any of these components they **must start** with the key letter followed by a dash **"-"**.

4.2.1 Kinematic component (k)

To specify line components generated from gas with different kinematic conditions, the user needs to add the "_k-1" suffix. The default value for lines with only one kinematic component is "_k-0". However, it is not neccesary to specify the zero value component. For example: In your configuration file, a transition with three kinematic componentsthis would look like this:

```
O3_5007A_b = O3_5007A+O3_5007A_k-1_O3_5007A_k-2

O3_5007A_k-1_sigma = expr:>2.0*O3_5007A_sigma

O3_5007A_k-2_sigma = expr:>2.0*O3_5007A_k-1_sigma
```

where `O3_5007A` would be the narrower component. It is recommended to use higher cardinals for the wider components.

Please remember: If you want to compare the same kinematic component between diferent lines or spectra, it is essential to add more constrains in the fitting configuration to make sure that the output profiles maintain a uniform

labeling.

4.2.2 Profile component (p)

This component specifies the profile for the fitting. The default value is "p-g-emi" for a Gaussian emission profile. Additional values for this entry are:

- p-g-emi: Emission profile (Gaussian)
- p-g-abs: Absorption profile (Gaussian)
- p-g-mix: Emission and absorption mixture (Gaussian)

Please remember: At the present time $LiMe$ can only fit a Gaussian profile.

4.2.3 Transition component (t)

This component provides information regarding the line transition in order to construct its classical notation using latex. The options currently available are:

- t-rec: Recombination line
- t-col: Collisional excited line
- t-sem: Semi-forbidden transition line
- t-mol: Mollecular line

[1]:
```
%matplotlib notebook
```

CHAPTER

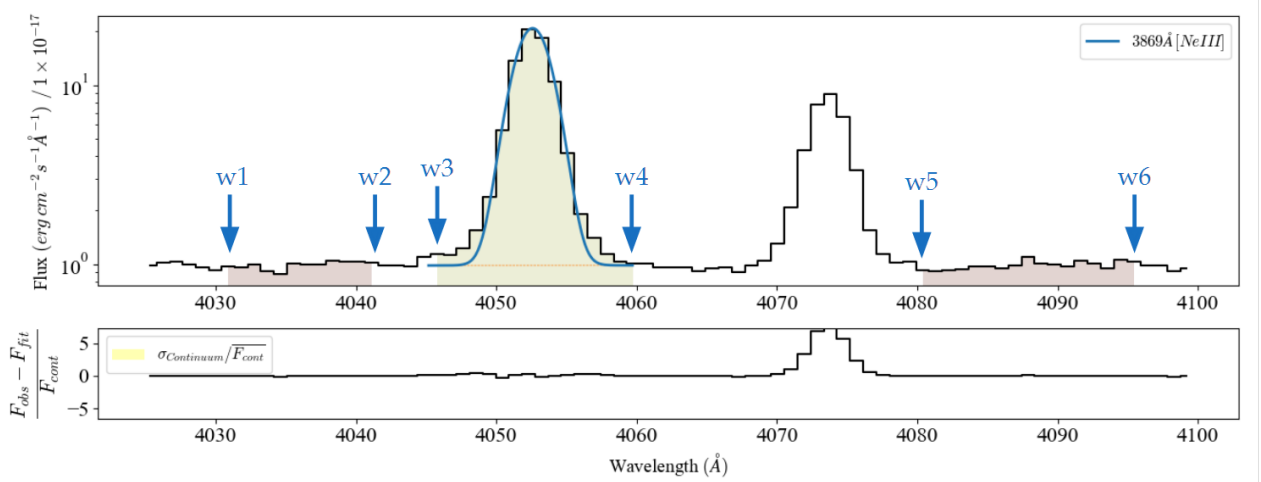
FIVE

LINE BANDS

The second input in $LiMe$ measurements is the band wavelength intervales with the line location and two adjacent and featureless continua. This design was inspired by the the Lick indexes by Worthey et al (1993) and references therein.

At the $LiMe$ github, you may download this page as a notebook or a script.

[2]:
```
import numpy as np
from astropy.io import fits
from IPython.display import Image, display
from pathlib import Path
import lime

display(Image(filename='../images/bands_definition.png'))
```



A band consists in a 6 values array $(w_1, ..w_6)$ with the wavelength boundaries for the line location and two nearby blue and red continua. For $LiMe$ measurements, it is **essential** that:

- The wavelenght array is sorted from lower to higher values.
- The wavelength values are in the rest frame.
- The wavelength units are the same as those declared in the target `lime.Spectrum` or `lime.Cube` observations.

5.1 Default lines database:

LiMe includes a database with common lines which can be observed in astronomical spectra. To access this database you can use the `lime.line_bands`:

[3]:
```
bands_df = lime.line_bands()
```

The default table wavelengths are in angstroms with the observed `wavelength` and band boundaries (`w1`, `w2`,`w3`,`w4`,`w5`,`w6`) wavelength values in air.

However, you can constrain the output bands from the `lime.line_bands` function using its attributes. For example, you can limit the output line bands by a wavelenght interval with `wave_inter`, as well as a `lines_list` and `particle_list`. Regarding the output values, you can specify the `units_wave` and whether to apply a `vacuum_conversion`. Finally, you can state the number of decimals on the line labels using the `sig_fig` attribute:

[4]:
```
lime.line_bands(wave_intvl=(300, 900), particle_list=('He1','O3','S2'), units_wave='nm',␣
↪decimals=None, vacuum=True)
```

[4]:

	wavelength	wave_vac	w1	w2	w3	\
He1_403nm	402.733452	402.73345	401.503486	402.244991	402.270129	
S2_407nm	406.974903	406.97490	405.938106	406.587703	406.630488	
O3_436nm	436.443598	436.44360	429.897570	432.494291	435.817822	
He1_447nm	447.274040	447.27404	445.323193	446.639157	446.851915	
He1_492nm	492.330504	492.33050	490.718146	491.762466	491.908118	
O3_496nm	496.029500	496.02950	493.073376	494.818947	494.976792	
O3_501nm	500.824004	500.82400	497.326052	498.598165	499.681938	
He1_588nm	587.724329	587.72433	584.742570	586.742789	586.886434	
He1_668nm	667.999556	667.99955	666.105098	667.204538	667.267813	
S2_672nm	671.829502	671.82950	668.873329	670.826383	670.941255	
S2_673nm	673.267400	673.26740	668.873329	670.826383	672.741766	
He1_707nm	706.716328	706.71633	703.114524	704.368881	705.727267	

	w4	w5	w6	latex_label	particle	\
He1_403nm	403.335171	403.435647	404.138977	$HeI403nm$	He1	
S2_407nm	407.441240	408.159187	408.819158	$[SII]407nm$	S2	
O3_436nm	437.401787	439.206155	441.750863	$[OIII]436nm$	O3	
He1_447nm	447.876791	448.209876	449.824056	$HeI447nm$	He1	
He1_492nm	492.953284	493.204228	494.116484	$HeI492nm$	He1	
O3_496nm	497.217554	497.428449	498.588373	$[OIII]496nm$	O3	
O3_501nm	502.578172	502.922278	504.528111	$[OIII]501nm$	O3	
He1_588nm	588.729942	589.010016	590.358048	$HeI588nm$	He1	
He1_668nm	669.001580	669.174956	670.012944	$HeI668nm$	He1	
S2_672nm	672.742554	674.640249	676.192505	$[SII]672nm$	S2	
S2_673nm	674.343008	674.640249	676.192505	$[SII]673nm$	S2	
He1_707nm	708.149955	708.955184	710.651666	$HeI707nm$	He1	

	transition	rel_int
He1_403nm	rec	0
S2_407nm	col	0
O3_436nm	col	0
He1_447nm	rec	0
He1_492nm	rec	0
O3_496nm	col	0
O3_501nm	col	0

(continues on next page)

(continued from previous page)

```
He1_588nm        rec        0
He1_668nm        rec        0
S2_672nm         col        0
S2_673nm         col        0
He1_707nm        rec        0
```

5.2 Using a dataframe:

In $LiMe$, a bands table (and the output measurement logs) variable is a pandas Dataframe.

To get the data from a certain column you can use several commands:

[5]:
```
print(bands_df.columns)

labels = bands_df.index.to_numpy()
ions = bands_df['particle'].to_numpy()
wave_array = bands_df.wavelength.to_numpy()

labels, ions, wave_array
```

```
Index(['wavelength', 'wave_vac', 'w1', 'w2', 'w3', 'w4', 'w5', 'w6',
       'latex_label', 'particle', 'transition', 'rel_int'],
      dtype='object')
```

[5]:
```
(array(['H1_1215A', 'C4_1548A', 'He2_1640A', 'O3_1666A', 'C3_1908A',
        'Mg2_2803A', 'Ne5_3426A', 'H1_3704A', 'O2_3726A', 'O2_3729A',
        'H1_3750A', 'H1_3771A', 'H1_3798A', 'H1_3835A', 'Ne3_3869A',
        'H1_3889A', 'H1_3970A', 'He1_4026A', 'S2_4069A', 'H1_4102A',
        'H1_4340A', 'O3_4363A', 'He1_4471A', 'Fe3_4658A', 'He2_4685A',
        'Ar4_4711A', 'Ar4_4740A', 'H1_4861A', 'He1_4922A', 'O3_4959A',
        'O3_5007A', 'N2_5755A', 'He1_5876A', 'O1_6300A', 'S3_6312A',
        'N2_6548A', 'H1_6563A', 'N2_6583A', 'He1_6678A', 'S2_6716A',
        'S2_6731A', 'He1_7065A', 'Ar3_7136A', 'O2_7319A', 'O2_7330A',
        'Ar3_7751A', 'H1_8392A', 'H1_8413A', 'H1_8438A', 'H1_8467A',
        'H1_8502A', 'H1_8545A', 'H1_8598A', 'H1_8665A', 'H1_8750A',
        'H1_8863A', 'H1_9015A', 'S3_9068A', 'H1_9229A', 'S3_9530A',
        'H1_9546A', 'H1_10049A', 'He1_10830A', 'H1_10938A', 'O1_11287A',
        'Fe2_12566A', 'H1_12818A', 'Fe2_16443A', 'H1_18751A', 'H1_19445A',
        'H2_19570A', 'Si6_19640A', 'He1_20581A', 'H2_21212A', 'H1_21655A',
        'H2_24059A', 'Si6_24823A', 'H1_26251A', 'H1_30383A', 'H1_32960A',
        'PAH1_32991A', 'H1_37395A', 'H1_40511A', 'H1_46524A', 'S4_105075A',
        'S3_187076A'], dtype=object),
 array(['H1', 'C4', 'He2', 'O3', 'C3', 'Mg2', 'Ne5', 'H1', 'O2', 'O2',
        'H1', 'H1', 'H1', 'H1', 'Ne3', 'H1', 'H1', 'He1', 'S2', 'H1', 'H1',
        'O3', 'He1', 'Fe3', 'He2', 'Ar4', 'Ar4', 'H1', 'He1', 'O3', 'O3',
        'N2', 'He1', 'O1', 'S3', 'N2', 'H1', 'N2', 'He1', 'S2', 'S2',
        'He1', 'Ar3', 'O2', 'O2', 'Ar3', 'H1', 'H1', 'H1', 'H1', 'H1',
        'H1', 'H1', 'H1', 'H1', 'H1', 'H1', 'S3', 'H1', 'S3', 'H1', 'H1',
        'He1', 'H1', 'O1', 'Fe2', 'H1', 'Fe2', 'H1', 'H1', 'H2', 'Si6',
        'He1', 'H2', 'H1', 'H2', 'Si6', 'H1', 'H1', 'H1', 'PAH1', 'H1',
        'H1', 'H1', 'S4', 'S3'], dtype=object),
```

(continues on next page)

(continued from previous page)

```
array([  1215.1108,   1547.6001,   1639.7896,   1665.5438,   1908.0803,
         2802.6607,   3426.    ,   3703.8013,   3726.03  ,   3728.82  ,
         3750.0998,   3770.5779,   3797.845 ,   3835.3309,   3868.7029,
         3888.995 ,   3970.017 ,   4026.134 ,   4068.5368,   4101.6777,
         4340.4035,   4363.1421,   4471.4164,   4658.024 ,   4685.4956,
         4711.1891,   4740.0511,   4861.2582,   4921.8552,   4958.8348,
         5006.7664,   5754.64  ,   5875.525 ,   6300.2076,   6311.9682,
         6547.9514,   6562.7192,   6583.3513,   6678.05  ,   6716.3386,
         6730.7135,   7065.1078,   7135.6845,   7318.8124,   7329.5494,
         7750.9894,   8392.2696,   8413.1907,   8437.8276,   8467.1263,
         8502.3542,   8545.254 ,   8598.2619,   8664.8868,   8750.3405,
         8862.6484,   9014.773 ,   9068.4736,   9228.8738,   9530.4417,
         9545.8253,  10049.2164,  10830.1763,  10937.9211,  11287.    ,
        12566.    ,  12817.8766,  16443.2162,  18750.6943,  19445.2888,
        19570.3629,  19640.3428,  20580.9773,  21212.2213,  21654.9742,
        24058.9915,  24822.854 ,  26251.1236,  30383.2759,  32960.4251,
        32990.5065,  37394.7903,  40511.0344,  46524.3554, 105074.7773,
       187076.1931]))
```

Similarly, you can use these comands to get the data from one of the rows:

[6]:
```
# List of lines on this dataframe
print(bands_df.index.to_numpy())
```

```
['H1_1215A' 'C4_1548A' 'He2_1640A' 'O3_1666A' 'C3_1908A' 'Mg2_2803A'
 'Ne5_3426A' 'H1_3704A' 'O2_3726A' 'O2_3729A' 'H1_3750A' 'H1_3771A'
 'H1_3798A' 'H1_3835A' 'Ne3_3869A' 'H1_3889A' 'H1_3970A' 'He1_4026A'
 'S2_4069A' 'H1_4102A' 'H1_4340A' 'O3_4363A' 'He1_4471A' 'Fe3_4658A'
 'He2_4685A' 'Ar4_4711A' 'Ar4_4740A' 'H1_4861A' 'He1_4922A' 'O3_4959A'
 'O3_5007A' 'N2_5755A' 'He1_5876A' 'O1_6300A' 'S3_6312A' 'N2_6548A'
 'H1_6563A' 'N2_6583A' 'He1_6678A' 'S2_6716A' 'S2_6731A' 'He1_7065A'
 'Ar3_7136A' 'O2_7319A' 'O2_7330A' 'Ar3_7751A' 'H1_8392A' 'H1_8413A'
 'H1_8438A' 'H1_8467A' 'H1_8502A' 'H1_8545A' 'H1_8598A' 'H1_8665A'
 'H1_8750A' 'H1_8863A' 'H1_9015A' 'S3_9068A' 'H1_9229A' 'S3_9530A'
 'H1_9546A' 'H1_10049A' 'He1_10830A' 'H1_10938A' 'O1_11287A' 'Fe2_12566A'
 'H1_12818A' 'Fe2_16443A' 'H1_18751A' 'H1_19445A' 'H2_19570A' 'Si6_19640A'
 'He1_20581A' 'H2_21212A' 'H1_21655A' 'H2_24059A' 'Si6_24823A' 'H1_26251A'
 'H1_30383A' 'H1_32960A' 'PAH1_32991A' 'H1_37395A' 'H1_40511A' 'H1_46524A'
 'S4_105075A' 'S3_187076A']
```

[7]:
```
H1_1215A_params = bands_df.iloc[0].to_numpy()
H1_4861A_params = bands_df.loc['H1_4861A'].to_numpy()
H1_1215A_params, H1_4861A_params
```

[7]:
```
(array([1215.1108, 1215.6699, 1100.0, 1150.0, 1195.0, 1230.0, 1250.0,
        1300.0, '$HI1215\\AA$', 'H1', 'rec', 0], dtype=object),
 array([4861.2582, 4862.691, 4809.8, 4836.1, 4848.715437, 4876.181741,
        4883.13, 4908.4, '$HI4861\\AA$', 'H1', 'rec', 0], dtype=object))
```

Finally, you can combine these commands to access the data from certain cells:

[8]:
```
bands_df.at['H1_1215A', 'wavelength']
```

```
[8]: 1215.1108
```

```
[9]: bands_df.loc['H1_1215A', 'wavelength'], bands_df.loc['H1_1215A'].wavelength
[9]: (1215.1108, 1215.1108)
```

```
[10]: bands_df.loc[['H1_1215A', 'H1_4861A'], 'wavelength'].to_numpy(),
[10]: (array([1215.1108, 4861.2582]),)
```

```
[11]: bands_df.loc['H1_1215A':'He2_1640A', 'wavelength'].to_numpy()
[11]: array([1215.1108, 1547.6001, 1639.7896])
```

```
[12]: bands_df.loc['H1_1215A', 'w1':'w6'].to_numpy()
[12]: array([1100.0, 1150.0, 1195.0, 1230.0, 1250.0, 1300.0], dtype=object)
```

5.3 Load/save a bands dataframe/measurements log:

To save these dataframes you can use the `save_log`:

```
[13]: # Save to the current folder in several formats:
lime.save_log(bands_df, 'bands_frame.txt')
lime.save_log(bands_df, 'bands_frame.pdf', parameters=['wavelength', 'latex_label'])
```

In files with a multi-page structure you can specify the extension:

```
[15]: lime.save_log(bands_df, 'bands_frame.xlsx', page='BANDS')
lime.save_log(bands_df, 'bands_frame.fits', page='BANDS')
```

```
D:\Pycharm projects\lime\src\lime\io.py:415: FutureWarning: Setting the `book` attribute␣
↪is not part of the public API, usage can give unexpected or corrupted results and will␣
↪be removed in a future version
  writer.book = book
```

Similarly, to read these files as a pandas dataframe you can use:

```
[16]: bands_df = lime.load_log('bands_frame.txt')
bands_df = lime.load_log('bands_frame.xlsx', page='BANDS')
bands_df = lime.load_log('bands_frame.fits', page='BANDS')
```

5.4 Updating the bands database:

The user is adviced to keep his/her own bands database with the bands and latex labels he/she prefers. Afterwards, you can use it in $LiMe$ functions (including `lime.line_bands`) to run your scripts. However, in case you need to update the default database you can recover its location with this command:

```
[17]: lime.io._parent_bands_file
```

[17]:
```
WindowsPath('D:/Pycharm projects/lime/src/lime/resources/parent_bands.txt')
```

Please remember: If you need to update the default $LiMe$ lines database make sure to fill all the columns. If there is any issue you can download the default database from the github (or reinstall $LiMe$)

[1]:
```
%matplotlib notebook
```

CHAPTER

SIX

FITTING CONFIGURATION

For each line fitting, $LiMe$ provides two kind of measurements. Those based on the integration of the flux within the line band and those which assume a shape for the line profile. By default, $LiMe$ assumes a single Gaussian profile, which is parametrised by the following formula:

$F_\lambda = \sum_i A_i e^{-\left(\frac{\lambda-\mu_i}{2\sigma_i}\right)^2} + c_\lambda$

where, one the left-hand side F_λ is the line flux. On the right-hand side, we have two components: The first one is the Gaussian profile flux, where A_i is the height of a Gaussian profile above the continuum level, μ_i is the center of the of Gaussian profile and σ_i is the standard deviation of the profile. The second component is the continuum level of the line c_λ, $LiMe$ computes this continuum from the adjacent bands assuming a linear relation and by default it is kept constanst during the fitting.

Please remember: By default $LiMe$ assumes a single Gaussian profile in the line measurements. To fit anything more complex, the line label must have the blended suffix **(_b)** and the user needs to specify the **line components** in the profile configuration

---

To show some examples regarding the fitting configuration notation, we are going to use some data from the 3rd tutorial. You can download this documentation page as a notebook from the library github.

6.1 Configuration files

Let's start by loading the libraries to run the samples:

[2]:
```
import numpy as np
from astropy.io import fits
from IPython.display import Image, display
from pathlib import Path
import lime
```

Now we proceed to load the scientific data:

[3]:
```
def import_osiris_fits(file_address, ext=0):

    # Open the fits file
    with fits.open(file_address) as hdul:
        data, header = hdul[ext].data, hdul[ext].header

    # Reconstruct the wavelength array from the header data
```

(continues on next page)

(continued from previous page)

```
    w_min, dw, n_pix = header['CRVAL1'],  header['CD1_1'], header['NAXIS1']
    w_max = w_min + dw * n_pix
    wavelength = np.linspace(w_min, w_max, n_pix, endpoint=False)

    return wavelength, data, header
```

[4]:
```
# State the data files
obsFitsFile = '../sample_data/gp121903_osiris.fits'
lineBandsFile = '../sample_data/osiris_bands.txt'
configFile = '../sample_data/osiris.toml'

# Load spectrum
wave, flux, header = import_osiris_fits(obsFitsFile)

# Load line bands
bands_df = lime.load_log(lineBandsFile)
```

Now, we need to read the configuration file. This is a text file which uses the toml configuration format. In the case of the `osiris.toml` file:

[5]:
```
display(Image(filename='../images/conf_file_osiris.png'))
```

osiris.toml

```
[sample_data]

files_list = ["gp030321_BR.fits", "gp101157_BR.fits", "gp121903_BR.fits"]
object_list = ["gp030321", "gp101157", "gp121903"]

z_array = [0.16465, 0.14334, 0.19531]
zErr_array = [7.389e-5, 0.000129, 0.0001403]

norm_flux = 1e-14

[gp121903_line_fitting]

O2_3726A_m = "O2_3726A+O2_3729A"
H1_3889A_m = "H1_3889A+He1_3889A"
Ar4_4711A_m = "Ar4_4711A+He1_4713A"
O2_7319A_m = "O2_7319A+O2_7330A"
H1_6563A_b = "H1_6563A+N2_6584A+N2_6548A"
O1_6300A_b = "O1_6300A+S3_6312A"

S2_6716A_b = "S2_6716A+S2_6731A"
S2_6731A_kinem = "S2_6716A"

N2_6548A_amp = "expr:N2_6584A_amp/2.94"
N2_6548A_kinem = "N2_6584A"
```


These configuration files consit in one or more sections, enclosed by squared brackets (**[section_name]**) with several properties/keys equal to their assigned value (**property = value**). The user is encouraged to read the toml examples to take full advantage on how to store variables on a text file. However, it is important to keep in mind the following constrains:

- String properties must be surrounded by single ('my string') or double ("my string") quotation marks.
- By default Section and property names can only have ASCII letters, ASCII digits, underscores, and dashes. If you want to use additional character you need to declare them as strings (using single or double quotation marks). The latter option is not recomended though.
- Section and property names should not use dots. In configuration files dots are used to declare key-value sub-structures. If you section or property label needs them you can use using single or double quotation marks.

You can use the `lime.load_cfg` function to read these files as a dictionary of dictionaries:

[6]:
```
obs_cfg = lime.load_cfg(configFile)
obs_cfg
```

[6]:
```
{'sample_data': {'files_list': ['gp030321_BR.fits',
   'gp101157_BR.fits',
```

(continues on next page)

(continued from previous page)

```
   'gp121903_BR.fits'],
  'object_list': ['gp030321', 'gp101157', 'gp121903'],
  'z_array': [0.16465, 0.14334, 0.19531],
  'zErr_array': [7.389e-05, 0.000129, 0.0001403],
  'norm_flux': 1e-17},
 'default_line_fitting': {'O2_3726A_m': 'O2_3726A+O2_3729A',
  'H1_3889A_m': 'H1_3889A+He1_3889A',
  'Ar4_4711A_m': 'Ar4_4711A+He1_4713A',
  'O2_7319A_m': 'O2_7319A+O2_7330A'},
 'gp121903_line_fitting': {'O1_6300A_b': 'O1_6300A+S3_6312A',
  'O3_5007A_b': 'O3_5007A+O3_5007A_k-1',
  'O3_5007A_k-1_amp': {'expr': '<100.0*O3_5007A_amp', 'min': 0.0},
  'O3_5007A_k-1_sigma': {'expr': '>2.0*O3_5007A_sigma'},
  'H1_6563A_b': 'H1_6563A+N2_6584A+N2_6548A',
  'N2_6548A_amp': {'expr': 'N2_6584A_amp/2.94'},
  'N2_6548A_kinem': 'N2_6584A',
  'S2_6716A_b': 'S2_6716A+S2_6731A',
  'S2_6731A_kinem': 'S2_6716A'}}
```

Please remember: If one of the sections in your configuration files has the **$line fitting$ * *$suffix$, $LiMe$ $transforms$ $some$ of the key $values$ so $they$ can be $used$ $directly$ in the $functions$. For $example$ the $property$ $value$:

N2_6548A_amp = "expr:N2_6584A_amp/2.94"

is converted into a dictionary:

N2_6548A_amp : {'expr': 'N2_6584A_amp/2.94'}

The user is encourge to use **_line_fitting suffix in sections with data for $LiMe$ fittings. This will make sure the inputs are formatted to the expected format.**

Beyond $LiMe$ fitting parameters you can store any kind of information on this file:

[7]:
```
z_obj = obs_cfg['sample_data']['z_array'][2]
norm_flux = obs_cfg['sample_data']['norm_flux']
```

[8]:
```
# Declare LiMe spectrum
gp_spec = lime.Spectrum(wave, flux, redshift=z_obj, norm_flux=norm_flux)
```

6.2 Blended and merged line components:

We can see that in the [gp121903_line_fitting] section there are several blended and merged lines:

[12]:
```
# Section with the fitting information:
fit_cfg = obs_cfg['gp121903_line_fitting']
```

[13]:
```
# Some lines with the multiple components:
obs_cfg['default_line_fitting']['H1_3889A_m'], obs_cfg['gp121903_line_fitting']['S2_
↪6716A_b']
```

[13]:
```
('H1_3889A+He1_3889A', 'S2_6716A+S2_6731A')
```

If we provide these components in the fitting (and the **_b** suffix is included in the line label) multiple components will be fitted:

[14]:
```
# Fit and plot of a blended line with two components
gp_spec.fit.bands('S2_6716A_b', bands_df, fit_cfg)
gp_spec.plot.bands('S2_6716A_b')
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

Please remember: The components for a blended or merged line must follow the line label notation joined with the plus (**+**) symbol.

6.3 Model parameters:

$LiMe$ solves the mathematical model using LmFIT. For each line we define 3 parameters per component in the equation above plus 2 more for the local continuum. These are labeled using the line name and the following suffixes:

- **_amp**: the height of the Gaussian from the continuum in the spectrum flux (normalised) units.
- **_center**: the wavelength of the Gaussian peak in the spectrum wavelength units.
- **_sigma**: the width of the Gaussian curve in the spectrum wavelength units.
- **_cont_slope**: the local continuum gradient. In blended lines, there is only one continuum labeled after the first component.
- **_cont_intercept**: the linear flux at zero wavelength for the local continuum. In blended lines there is still only one continuum intercept labeled after the first component.

Please remember: It should be explained that we are not using the default Gaussian model in LmFit: Their in-built model defines the amplitude A_i as the area under the Gaussian profile. While this notation provides a cleaner mathematical description, in the fitting of astronomical spectra it is seldom easy to constrain the flux in advance. Instead, it is easier to impose limits on the line height, which is our definition.

In your configuration file, you can adjust the fitting for each of these parameters with the by adding line label followed by the corresponding suffix. For example, in the $[NII]6548$ we constrained its amplitude:

```
H1_6563A_b = "H1_6563A+N2_6584A+N2_6548A"
N2_6548A_amp = "expr:N2_6584A_amp/2.94"
N2_6548A_kinem = "N2_6584A"
```

There are five attributes you can use to adjust the fitting on the parameters:

- `value`: Initial value for the parameter. $LiMe$ provides an initial guess for the parameters from the integrated measurements.
- `vary`: Whether the parameter is free during the fitting (default is True). If set to *False* the initial `value` will remain unchanged.
- `min`: Lower bound for the parameter value. The default is value is -numpy.inf (no lower bound).

- `max`: Upper bound for the parameter value. The default is value is numpy.inf (no upper bound).
- `expr`: Mathematical expression to constrain the value during the fit. The default value is None.

Please remember: In the configuration file, the line parameters attributes are comma **(,)** separated and the key-value entries are separated by a colon **(:)**.

The `expr` parameter can be used to link the fitting to other components (if those components belong to the current fitting). For example, in the case above we are tiding the $[NII]6548$ amplitude to the one of $[NII]6584$ given theoretical emissivity relation between these transitions (2.94, which is constant independently of the physical conditions):

[15]:
```
# Fit and plot of a line with a parameter using the "expr" :
gp_spec.fit.bands('H1_6563A_b', bands_df, fit_cfg)
gp_spec.plot.bands()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

This is a convenient way to tie the flux between lines in the same blended groups. As you can see from the plot above the Gaussin flux ratio between both transitions is very close to the theoretical ratio. This can also be concluded from the measurements log:

[16]:
```
N2_flux_ratio = gp_spec.log.loc["N2_6584A", "gauss_flux"]/gp_spec.log.loc["N2_6548A",
↪"gauss_flux"]
print(f'[NII] doublet gaussian flux ratio: {N2_flux_ratio}')
```

```
[NII] doublet gaussian flux ratio: 2.956163711944168
```

Similarly, if we want to fit the weak wide component of $[OIII]5007$ we can use the command:

```
O3_5007A_b = 'O3_5007A+O3_5007A_k-1'
O3_5007A_sigma = "value:2,max:4"
O3_5007A_k-1_sigma = "value:5,min:4.0"
```

[17]:
```
gp_spec.fit.bands('O3_5007A_b', bands_df.loc['O3_5007A', 'w1':'w6'], fit_cfg)
gp_spec.plot.bands('O3_5007A_b', rest_frame=True)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

6.4 Inequality boundary conditions:

$LiMe$ includes the posibility to add inequalities in LmFIT `expr` parameter. For example, the $[OIII]5007$ wide component could be fit using two inequalities:

[18]:
```
# Same line fitting but using inequalities
O3_ineq_cfg = {'O3_5007A_b'         : 'O3_5007A+O3_5007A_k-1',
               'O3_5007A_k-1_amp'   : {'expr': '<100.0*O3_5007A_amp', 'min': 0.0},
               'O3_5007A_k-1_sigma' : {'expr': '>2.0*O3_5007A_sigma'}}
gp_spec.fit.bands('O3_5007A_b', bands_df.loc['O3_5007A', 'w1':'w6'], O3_ineq_cfg)
gp_spec.plot.bands("O3_5007A_b", rest_frame=True)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

In this case we have provided two inequalities for the amplitude and sigma of the Gaussian profile of the `O3_5007A_k-1`.

It is important for the user to realize that it is essential to include these contrains for the fitting of multiple kinematic components even if the instrument resolution is high enough. This is because the algorithm may interchange the fitting label (k-0, k-1,...) from one line to another if the user does not provide any indication.

Please remember: In the current release only multiplication can be used along the inequality term. Please contact the author if you have any issue/request with this functionality.

6.5 Importing external line kinematics:

While the amplitude of an emission profile is dominated by the emissivity of the corresponding transition and the gas physical conditions; its width is mostly dependant on the gas kinematics. Moreover, discrepancies with the theoretical wavelength are also due to the gas kinematics.

Consequently, in some cases it is useful to constrain the line velocity dispersion (σ) and radial velocity in the line of sight (v_r) from the measurements of another line. For example:

$\sigma_A = \sigma_B$

$v_{r,\,A} = v_{r,\,B}$

where *A* and *B* are two line labels and both v_r and σ are in velocity units (for example km/s). To convert the equations above to the spectrum wavelength units, we use:

$\sigma_A = \sigma_B \cdot \frac{\lambda_A}{\lambda_B}\,(\text{Å})$

$\mu_A = \mu_B \cdot \frac{\lambda_A}{\lambda_B}\,(\text{Å})$

where μ and λ are observed and theoretical transition wavelength respectively. The second term takes into consideration the object redshift.

In the configuration file, these two constrains can be set simultaneously with the **_kinem** suffix on the line importing the kinematics. For example, in the fitting of $H\alpha$ and the $[NII]6548, 6584$ doublet we had:

```
H1_6563A_b = "H1_6563A+N2_6584A+N2_6548A"
N2_6548A_amp = "expr:N2_6584A_amp/2.94"
N2_6548A_kinem = "N2_6584A"
```

Resulting in:

[19]:
```
gp_spec.plot.bands('H1_6563A')
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

In this case, the kinematics of $[NII]6548, 6584$ lines are tied together together during the fitting. However, we can also use this line to import the kinematics from a line measured previously. For example, we can measure $H\beta$:

[20]:
```
gp_spec.fit.bands('H1_4861A', bands_df, fit_cfg)
gp_spec.plot.bands()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

And repeat the fitting imposing this line kinematics on the three transition:

[21]:
```
# Fit of a line importing the kinematics from an external line
Halpha_cfg = {'H1_6563A_b'      : 'H1_6563A+N2_6584A+N2_6548A',
              'H1_6563A_kinem'  : "H1_4861A",
              'N2_6584A_kinem'  : "H1_4861A",
              'N2_6548A_kinem'  : "H1_4861A",
              'N2_6548A_amp'    : {'expr': 'N2_6584A_amp/2.94'},
             }
gp_spec.fit.bands('H1_6563A_b', bands_df, Halpha_cfg)
gp_spec.plot.bands('H1_6563A_b')
```

```
<IPython.core.display.Javascript object>
```

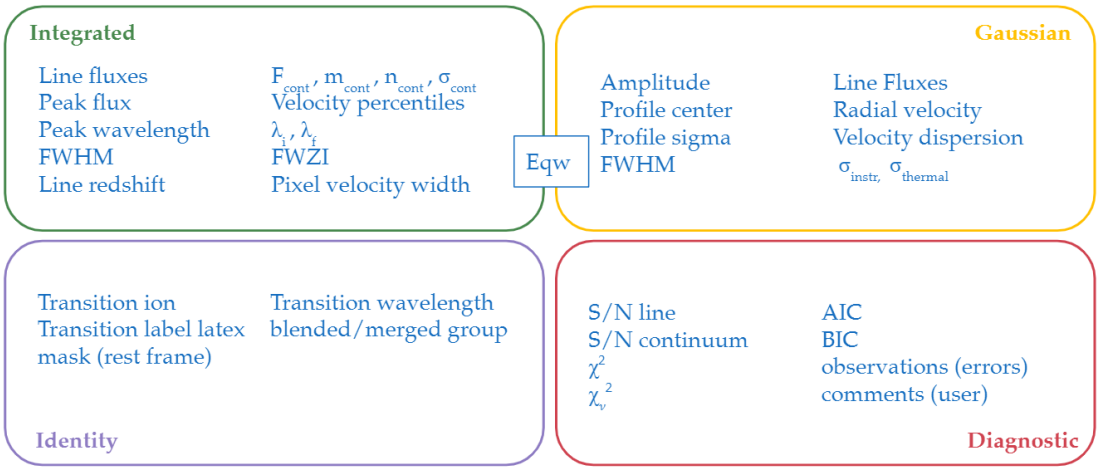
```
<IPython.core.display.HTML object>
```

From these results, you can see that the kinematics of $H\beta$ fit reasonable well in the $H\alpha$ aline. However, there are issues for the $[NII]$ doublet. This is due to the wide component of $H\alpha$.

CHAPTER

SEVEN

MEASUREMENTS DESCRIPTION



This section describes the parameters measured by LiMe. Unless otherwise noted, these parameters have the same notation in the output measurements log as the attributes in the programing objects generated with the `lime.Spectrum` class. These parameter references are also the column names of the `pandas.DataFrame` lines log (`lime.Spectrum.log`).

7.1 Inputs

This section includes 3 parameters which are actually provided by the user inputs. However, they are also included in the output log for consistency.

- **line** (`.line`, `str`): This attribute is name of the line the measurements belong to. It has the LiMe line notation: format.
- **band** (`.band`, `np.array()`): This attribute consists in a six-value vector with the line bands: In the `lime.Spectrum` object, the mask is stored as a vector under the `lime.Spectrum.mask` attribute. In the `.log` the wavelengths are stored in individual columns with the headers: `w1`, `w2`, `w3`, `w4`, `w5` and `w6`.
- **profile_label** (`.profile_label`, `str`): This attribute consists in a string with the line components separated by dashes (-). The individual components labels have the LiMe line notation and they may also have a suffix for the kinematic component. In single lines, the default value for this attribute is `None` (string variable). As an example, two profile labels are included below:

```
H1_6563A_b = H1_6563A-H1_6563A_b1-N2_6584A-N2_6548A
O2_3727A_m = O2_3727A-O2_3729A
```

7.2 Identification

These parameters are not attributes of the `lime.Spectrum` class. Nonetheless, they are stored in the `lime.Spectrum.log` `pandas.DataFrame` and the output measuring logs for their convenience in posterior treatments.

- **wave**: This parameter contains the theoretical, rest-frame, wavelength for the emission line. This value is derived from the line label provided by the user.
- **ion**: This parameter contains the ion responsible for the emission line photons. This value is derived from the line label provided by the user.
- **latex_label**: This parameter contains the transition classical notation in latex format. This string includes the profile components if they were provided during the fitting.

7.3 Integrated properties

These attributes are calculated by the `lime.Spectrum.line_properties` function. In these calculations, there is no assumption on the emission line profile shape.

Attention: In the output measurements log and the `lime.Spectrum.log`, these parameters have the same flux units as the input spectrum. However, the attributes of the `lime.Spectrum` are normalized by the `.norm_flux` constant provided by the user at the `lime.Spectrum` definition.

- **peak_wave** (`.peak_wave`, `float`): This variable is the wavelength of the highest pixel value in the line region.
- **peak_flux** (`.peak_flux`, `float`): This variable is the flux of the highest pixel value in the line region.
- **m_cont** (`.m_cont`, `float`): Using the line adjacent continua regions LiMe fits a linear continuum. This variable represents is the gradient. `y = m*x + n`
- **n_cont** (`.n_cont`, `float`): Using the line adjacent continua regions LiMe fits a linear continuum. This variable represents is the interception. `y = m*x + n`
- **cont** (`.cont`, `float`): This variable is the flux of the linear continuum at the `.peak_wave`.
- **cont_err** (`.cont_err`, `float`): This variable is standard deviation of the adjacent continua flux. It is calculated from the observed continuum minus the linear model for both continua masks.
- **intg_flux** (`.intg_flux`, `float`): This variable contains measurement of the integrated flux. This value is calculated via a Monte Carlo algorithm:
  - If the pixel error spectrum is not provided by the user, the algorithm uses the line `.cont_err` as a uniform uncertainty for all the line pixels.
  - The pixel error is added stochastically to each pixel in the line region mask.
  - The flux in the line region is summed up taking into consideration the line region averaged pixel width and removing the contribution of the linear continuum.
  - The previous two steps are repeated in a 1000 loop. The mean flux value from the resulting array is taken as the integrated flux value.
- **intg_err** (`.intg_err`, `float`): This attribute contains the integrated flux uncertainty. This value is derived from the standard deviation of the Monte Carlo flux calculation described above.

Attention: Blended components have the same `.intg_flux` and `.intg_err` values.

- **eqw** (`.eqw`, `float` or `np.array()`): This parameter is the equivalent of the emission line. It is calculated using the expression below:

$$Eqw = \int_{\lambda_1}^{\lambda_2} \frac{F_c - F_\lambda}{F_c} d\lambda = \int_{\lambda_1}^{\lambda_2} \frac{F_{line}}{F_c} d\lambda$$

  where F_c is the integrated flux of the linear continuum in the line region (`.cont`) and F_λ is the spectrum flux. In single lines, F_{line} is the integrated flux (`.intg_flux`) while in blended lines, the corresponding gaussian flux (`.gauss_flux`) is used. The integration limits for the line region are `w3` and `w4` from the input user mask.

- **eqw_err** (`.eqw`, `float` or `np.array()`): This parameter is the uncertainty in the equivalent width. It is calculated from a Monte Carlo propagation of the `.cont` and its `.cont_err` and the uncertainty of the line flux.

- **z_line** (`.z_line`, `float`): This variable is the emission line redshift:

$$z_\lambda = \frac{\lambda_{obs}}{\lambda_{theo}} - 1$$

  where λ_{obs} is the `.peak_wave`. In blended lines, this variable is computed using the same `.peak_wave` for all transitions (this is the most intense pixel in the line band).

- **FWHM_int** (`.FWHM_int`, `float`): This variable is the Full Width Half-Measure in km/s computed from the integrated profile: The algorithm finds the pixel coordinates which are above half the line peak flux. The blue and and red edge km/s are subtracted (blue is negative).

Attention: This operation is only available for lines whose width is above 15 pixels.

- **snr_line** (`.FWHM_int`, `float`): This variable is the signal to noise ratio of the emission line using the definition by Rola et al. 1994:

$$\frac{S}{N}_{line} \approx \frac{\sqrt{2\pi}}{6} \frac{A_{line}}{\sigma_{cont}} \sqrt{N} \approx \frac{F_{line}}{\sigma_{cont} \cdot \sqrt{N}}$$

  where A_{line} is the amplitude of the line, F_{line} is the integrated flux of the line (`.intg_flux`) σ_{cont} is the continuum flux standard deviation (`.cont_err`) and N is the number of pixels in the input line band. The later parameter approximates to $N = 6\sigma$ in single lines, where σ is the gaussian profile standard deviation.

- **snr_cont** (`.snr_cont`, `float`): This variable is the signal to noise ratio of the emission line region using the formula:

$$\frac{S}{N}_{cont} = \frac{F_{cont}}{\sigma_{cont}}$$

  where σ_{cont} is the continuum flux at the peak wavelength and σ_{cont} is the continuum flux standard deviation.

- **v_med** (`.v_med`, `float`): This variable is the median velocity of the emission line. The emission line wavelength is converted to velocity units using the formula:

$$V(Km/s) = c \cdot \frac{\lambda_{obs}}{\lambda_{peak}} - 1$$

  where $c = 299792.458 km/s$ is the speed of light, λ_{obs} is the wavelength mask array selection between $w3$ and $w4$ points and λ_{peak} is the `.peak_wave` of the emission line.

- **v_50** (`.v_50`, `float`): This variable is velocity corresponding to the 50th percentile of the emission line spectrum where the wavelength array is in km/s. A cumulative sum is performed in the line flux array. Afterwards, this array is multiplied by the `.pixelWidth` and divided by the `.intg_flux`. The resulting vector quantifies the flux percentage corresponding to each pixel in the $w3$ and $w4$ mask selection. Afterwards, this vector is interpolated with respect to the velocity array (whose calculation can be found above).

  Attention: This operation is only available for lines whose width is above 15 pixels.

- **v_5** (`.v_5`, `float`): This variable is the velocity corresponding to the 5th percentile of the emission line flux. The calculation procedure is described at the `.v_50` entry.
- **v_10** (`.v_10`, `float`): This variable is the velocity corresponding to the 10th percentile of the emission line flux. The calculation procedure is described at the `.v_50` entry.
- **v_90** (`.v_90`, `float`): This variable is the velocity corresponding to the 90th percentile of the emission line flux. The calculation procedure is described at the `.v_50` entry.
- **v_95** (`.v_95`, `float`): This variable is the velocity corresponding to the 95th percentile of the emission line flux. The calculation procedure is described at the `.v_50` entry.

7.4 Gaussian properties

These attributes are calculated by the `lime.Spectrum.gauss_lmfit` function. These calculations assume a Gaussian or multi-Gaussian profile:

$$F_\lambda = \sum_i A_i e^{-\left(\frac{\lambda-\mu_i}{2\sigma_i}\right)^2}$$

where F_λ is the combined flux profile of the emission line for the line wavelength range λ. A_i is the height of a gaussian component with respect to the line continuum (`.cont`), μ_i is the center of the of gaussian component and σ_i is the standard deviation. The first parameters has the input flux units (`lime.Spectrum.flux`), while the latter two have the input wavelength units (`lime.Spectrum.wave`).

The output uncertainty in these parameters corresponds to the 1 error: This is the standard error which increases the magnitude of the χ^2 calculated by the least squares algorithm.

Note: The Gaussian built-in model in LmFit defines the amplitude (A_i) as the flux under the gaussian profile. LiMe defines its own model where the amplitude is defined as the height of the line with respect to the adjacent continuum.

- **amp** (`.amp`, `np.array()`): This array contains the amplitude of the Gaussian profiles. The parameter units are those of the input spectrum flux (`lime.Spectrum.flux`).
- **amp_err** (`.amp_err`, `np.array()`): This array contains the uncertainty on the Gaussian profiles amplitude. The parameter units are those of the input flux (`lime.Spectrum.flux`).
- **center** (`.center`, `np.array()`): This array contains the Gaussian components central wavelength. The parameter units are those of the input spectrum wavelength (`lime.Spectrum.wave`).
- **center_err** (`.center_err`, `np.array()`): This array contains the uncertainty on the Gaussian profiles central wavelength.
- **sigma** (`.sigma`, `np.array()`): This array contains the Gaussian components standard deviation. The parameter units are those of the input spectrum wavelength.

- **sigma_err** (`.sigma_err`, `np.array()`): This array contains the uncertainty on the Gaussian profiles standard deviation.
- **v_r** (`.v_r`, `np.array()`): This array contains the Gaussian components radial velocity in km/s. This parameter is calculated using the expression:

$$v_r = c \cdot \frac{\lambda_{center}}{\lambda_{ref}} - 1$$

  where $c = 299792.458 km/s$ is the speed of light, λ_{center} is the Gaussian profile central wavelength (`.center`) and λ_{ref} is the reference wavelength. In non-blended lines λ_{ref} is the observed peak wavelength (`.peak_wave`). In blended lines, λ_{ref} is the component transition wavelength (`.wave`) shifted to the observed frame using the redshif provided by the user at the `lime.Spectrum`.
- **v_r_err** (`.v_r_err`, `np.array()`): This array contains the uncertainty of the Gaussian components radial velocity in km/s.
- **sigma_vel** (`.sigma_vel`, `np.array()`): This array contains the Gaussian components standard deviation in km/s. This parameter is calculated using the expression:

$$\sigma_v(km/s) = c \cdot \frac{\sigma}{\lambda_{ref}}$$

  where c $c = 299792.458 km/s$ is the speed of light, σ is the Gaussian profile standard deviation (`.sigma`) and λ_{ref} is the reference wavelength. In non-blended lines λ_{ref} is the observed peak wavelength (`.peak_wave`). In blended lines, λ_{ref} is the component transition wavelength (`.wave`) shifted to the observed frame using the redshif provided by the user at the `lime.Spectrum`.
- **sigma_vel_err** (`sigma_vel_err`, `float` or `np.array()`) This array contains the uncertainty of the Gaussian components standard deviation in km/s.
- **FWHM_g** (`.FWHM_g`, `np.array()`): This array contains the Full Width Half Maximum of the Gaussian components in in km/s. This parameter is calculated as:

$$FWHM_g = 2\sqrt{2\,ln2}\sigma_v$$

  where σ is the velocity dispersion of the Gaussian components (`.sigma_vel`).
- **gauss_flux** (`.gauss_flux`, `np.array()`): This array contains the flux of the Gaussian components. It is calculated using the expression:

$$F_{i,g} = A_i \cdot \sqrt{2\pi} \cdot \sigma_i$$

  where A_i is Gaussian component amplitude (`.amp`) and σ_i gaussian component standard deviation (`.sigma`)
- **gauss_flux_err** (`.gauss_flux_err`, `np.array()`): This array contains the uncertainty of the Gaussian components flux.

7.5 Diagnostics

These section contains the parameters which provide a qualitative or quantitative diagnostic on the line measurement.

- **chisqr** (`.chisqr`, `float`): This variable contains the χ^2 diagnostic calculated by LmFit
- **redchi** (`.redchi`, `float`): This variable contains the reduced χ^2 diagnostic calculated by LmFit:

$$\chi^2_\nu = \frac{\chi^2}{N - N_{varys}}$$

  where the χ^2 diagnostic is divided by the number of data points, N, minus the number of dimensions N_{varys}

- **aic** (`.aic`, `float`): This variable contains the Akaike information criteria calculated by LmFit
- **bic** (`.bic`, `float`): This variable contains the Bayesian information criteria calculated by LmFit
- **observation** (`.observation`, `str`): This variable contains errors or warnings generated during the fitting of the line (not implemented).
- **comments** (`.comments`, `str`): This variable is left empty for the user to store comments.

[1]:
```
%matplotlib notebook
```

CHAPTER

EIGHT

PLOTS AND INTERFACES

In this section, we review the visual aids provided by $LiMe$ measurements, as well as, some tips on how to adjust them to your workflow. If you have any issues with visualizing your data with $LiMe$, please contact the author or open an issue.

8.1 Jupyter notebooks and interactive shells

$LiMe$ functions should work in any IDE (Integrated Developement Environment) including Jupyter Notebooks.

However, many of $LiMe$ plotting tools make use of matplotlib widgets to select data or review the data. To take advantange of these features (as well as the default matplotlib plot tools) in a notebook, a few extra lines of code may be necessary.

This page has been compiled from a jupyter notebook. You can download it from this link at the library github, alongside the tutorials data to test the best configuration for your work.

Let's start by getting the data from the third tutorial:

[2]:
```
# %matplotlib qt # Commented for the tutorial workflow
import numpy as np
from astropy.io import fits
import lime

def import_osiris_fits(file_address, ext=0):

    # Open fits file
    with fits.open(file_address) as hdul:
        data, hdr = hdul[ext].data, hdul[ext].header

    w_min, dw, n_pix = hdr['CRVAL1'],  hdr['CD1_1'], hdr['NAXIS1']
    w_max = w_min + dw * n_pix
    wavelength = np.linspace(w_min, w_max, n_pix, endpoint=False)

    return wavelength, data, hdr

# State the data location
obsFitsFile = '../sample_data/gp121903_osiris.fits'
bands_df_file = '../sample_data/osiris_bands.txt'
cfgFile = '../sample_data/osiris.toml'

# Load the scientific data
```

(continues on next page)

(continued from previous page)

```
wave, flux, header = import_osiris_fits(obsFitsFile)
mask = lime.load_log(bands_df_file)
obs_cfg = lime.load_cfg(cfgFile)

# Declare spectrum properties
z_obj = obs_cfg['sample_data']['z_array'][2]
norm_flux = obs_cfg['sample_data']['norm_flux']
```

Now, we are going to define a `Spectrum` object and plot it:

[3]:
```
gp_spec = lime.Spectrum(wave, flux, redshift=z_obj, norm_flux=norm_flux)
gp_spec.plot.spectrum(label=f'GP121903 spectrum', rest_frame=True)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

In the expected behaviour, the notebook has displayed a fixed image of the GP121903 spectrum in the cell above. If your cell is empty, you might need to add the inline magic command at the beginnig of your cell (and/or the notebook):

[4]:
```
%matplotlib inline
```

and rerun the previous cell above. The `inline` option is fine in most scenarios, where we want to save the plots or a quick look.

However, when you need a closer inspection you can use the `%matplotlib notebook`, `%matplotlib widget` or `%matplotlib ipympl` commands. Each one selects a different backend depending on the user software.

Running the previous code with the `%matplotlib notebook` command, we should get an interactive plot:

[5]:
```
%matplotlib notebook
gp_spec.plot.spectrum(label=f'GP121903 spectrum', rest_frame=True)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

Depending on your set-up, alternating between the magic commands may not actully change the display type. In this case, you need to restart the jupyter notebook or interactve shell. **Indeed, it is recommended to use only one display style per notebook, where the magic command is the first line (before any imports).** In the unfair scenario, where you must switch between plot types, you can try to use the matplotlib.pyplot.switch_backend() command.

The previous magic commands should allow you to use the $LiMe$ interactive plots. However, due to the limited display window of the browser, this can be a challenge.

For example, lets try to use the `lime.MaskInspector` class to adjust the lines spectral bands:

[6]:
```
%matplotlib notebook
gp_spec.check.bands(bands_df_file)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

You can appreciate that the selection interface is not easy to use nor responsive.

As an alterntive, *the author recommends using the* `%matplotlib qt` backend for the LiMe interactive plots. This opens a new plotting window in your desktop (check whether it is minimized). This should provide the best user experience:

[7]:
```
%matplotlib qt
# Restart the kernel if the backend cannot be changed at this point
gp_spec.check.bands(bands_df_file)
```

```
Warning: Cannot change to a different GUI toolkit: qt. Using notebook instead.
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

This backend makes use of PyQt or Qt for Python. Either should be already available in your python installation.

[1]:
```
%matplotlib notebook
```

CHAPTER

NINE

1) SINGLE LINE FITTING

In this example, we perform single line fitting on the spectrum of the Green Pea galaxy GP121903 which was observed with the GTC (Gran Telescopio de Canarias). You can download this spectrum from the examples/sample_data. You can read more about these observations in Fernandez et al (2021).

This tutorial can found as a script and a notebook on the examples folder.

9.1 Loading the spectrum data

We start by importing the programing packages necessary to run the script, including $LiMe$.

[2]:
```
import numpy as np
from astropy.io import fits
import lime
```

The following functions reads a *.fits* from the *ISIS* instrument and returns the wavelength and flux arrays along with the data extension header.

[3]:
```
def import_osiris_fits(file_address, ext=0):

    # Open the fits file
    with fits.open(file_address) as hdul:
        data, header = hdul[ext].data, hdul[ext].header

    # Reconstruct the wavelength array from the header data
    w_min, dw, n_pix = header['CRVAL1'],  header['CD1_1'], header['NAXIS1']
    w_max = w_min + dw * n_pix
    wavelength = np.linspace(w_min, w_max, n_pix, endpoint=False)

    return wavelength, data, header
```

Now we can declare the data location and load it:

[4]:
```
# Address of the Green Pea galaxy spectrum
fits_file = '../sample_data/spectra/gp121903_osiris.fits'

# Load spectrum
wave, flux, hdr = import_osiris_fits(fits_file)
```

We provide the galaxy redshift and a normalization constant for the spectrum:

[5]:
```
# Galaxy redshift and the flux normalization
z_obj = 0.19531
normFlux = 1e-18
```

We now have all the data to define a $LiMe$ spectrum:

[6]:
```
# Define a spectrum object
gp_spec = lime.Spectrum(wave, flux, redshift=z_obj, norm_flux=normFlux)
```

We can plot the spectrum with the following command

[7]:
```
gp_spec.plot.spectrum(label='GP121903')
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

9.2 Perform the fittings

To measure lines you use the functions from the `fit` attribute. For example, in the case of a single line you would use the `fit.band`. For example, to measure of $H\alpha$ you can use:

[8]:
```
gp_spec.fit.bands(6563)
```

To plot the lattest fitting you can run:

[9]:
```
gp_spec.plot.bands()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

It is recommended, however, that users employ $LiMe$ label notation and provide their own line bands which take into account the spectrum resolution and emission features width. For example:

[10]:
```
# Line name and its location mask in the rest _frame
line = 'H1_6563A'
band_edges = np.array([6438.03, 6508.66, 6535.10, 6600.95, 6627.70, 6661.82])
```

We repeat now the fitting:

[11]:
```
# Run the fitting and plot it
gp_spec.fit.bands(line, band_edges)
gp_spec.plot.bands()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

You can see that the result was not very good. Let's increase the complexity by including the $[NII]$ doublet:

[12]:
```
# Fit configuration
line = 'H1_6563A_b'
fit_conf = {'H1_6563A_b': 'H1_6563A+N2_6584A+N2_6548A',
            'N2_6548A_amp': {'expr': 'N2_6584A_amp/2.94'},
            'N2_6548A_kinem': 'N2_6584A'}
```

The dictionary above we have three elements: - The line labelled as *H1_6563A_b* consists in three components: *H1_6563A*, *N2_6584A* and *N2_6548A*. - The line labelled as *N2_6548A* has an amplitude fixed by the amplitude of *N2_6584A*. - The line labelled as *N2_6548A* has its kinematics (both radial and dispersion velocity) tied to those of *N2_6584A*.

We repeat the fitting including this configuration:

[13]:
```
# New attempt including the fit configuration
gp_spec.fit.bands(line, band_edges, fit_conf=fit_conf)
gp_spec.plot.bands(line)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

Please remember: To declare a multi-Gaussian fitting, two conditions are necessary: The line label must have the **_b** suffix *(H1_6563A_b)* and the line components must be specified in the **fit_conf** dictionary *(H1_6563A-N2_6584A-N2_6548A)*.

9.3 Save the results

You can store a plot into an image file by adding an output address:

[14]:
```
# You can also save the fitting plot to a file
gp_spec.plot.bands(output_address=f'../sample_data/{line}.png')
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

The measurements are stored into the spectrum `log`:

[15]:
```
# Each fit is stored in the lines dataframe (log) attribute
gp_spec.log
```

[15]:
```
          wavelength     intg_flux  intg_flux_err    gauss_flux  \
H1_6563A      6563.0  3.128580e-14   8.603311e-18  2.918097e-14
N2_6584A      6584.0  3.128580e-14   8.603311e-18  1.476068e-15
N2_6548A      6548.0  3.128580e-14   8.603311e-18  4.993188e-16

          gauss_flux_err          eqw   eqw_err particle     latex_label  \
H1_6563A    1.108646e-16  1464.890899  5.565424       H1     $HI6563\AA$
N2_6584A    1.198697e-16    74.098922  6.017485       N2  $[NII]6584\AA$
N2_6548A    6.164800e-17    25.065906  3.201267       N2  $[NII]6548\AA$

                         group_label  ...        v_10        v_90        v_95  \
H1_6563A  H1_6563A+N2_6584A+N2_6548A  ... -304.862908  254.126399  427.536692
N2_6584A  H1_6563A+N2_6584A+N2_6548A  ... -304.862908  254.126399  427.536692
N2_6548A  H1_6563A+N2_6584A+N2_6548A  ... -304.862908  254.126399  427.536692

                 v_99        chisqr     redchi         aic         bic  \
H1_6563A  1043.457948  42224.143691  439.83483  626.629014  642.378851
N2_6584A  1043.457948  42224.143691  439.83483  626.629014  642.378851
```

(continues on next page)

(continued from previous page)

```
N2_6548A  1043.457948  42224.143691  439.83483  626.629014  642.378851

          observations  comments
H1_6563A            no        no
N2_6584A            no        no
N2_6548A            no        no

[3 rows x 59 columns]
```

This log can be saved into a file using the `save_log` attribute. The extention specifies the file type. You can also constrain the output measurements in addition to file sheet (only for multi-page file types)

[16]:
```
# It can be saved into different types of document using the function
gp_spec.save_log('../sample_data/example1_linelog.txt')
gp_spec.save_log('../sample_data/example1_linelog.pdf', param_list=['eqw', 'gauss_flux',
↪'gauss_flux_err'])
gp_spec.save_log('../sample_data/example1_linelog.fits', page='GP121903')
gp_spec.save_log('../sample_data/example1_linelog.xlsx', page='GP121903')
gp_spec.save_log('../sample_data/example1_linelog.asdf', page='GP121903')
```

A lines log can also be saved/loaded using the $LiMe$ functions:

[17]:
```
log_address = '../sample_data/example1_linelog.fits'
lime.save_log(gp_spec.log, log_address, page='GP121903')
log = lime.load_log(log_address, page='GP121903')
```

[1]:
```
%matplotlib notebook
```

CHAPTER

TEN

2) LINE BANDS INTERATIVE INSPECTION

In the previous tutorial, we measured one line from the OSIRIS spectrum of the galaxy GP121903. Before proceding to analyze the full spectrum, however, it is recommended to confirm the presence of lines.

In this tutorial, we are going to compare $LiMe$ lines database with the emission features observed in this spectrum. To do that we are going to use the `Spectrum.check.bands` function to iteratively adjust the observed bands to our instrument and observation.

If you are using a jupyter notebook, it is easier to use the `%matplotlib ql` backend at the top of your notebook, for a better iteration with your plots.

This tutorial is available as a script and a notebook in the in the github examples folder.. The galaxy spectrum is also available in the github sample data.

10.1 Loading the data

Let's start by importing the script packages and defining a function to read the ISIS spectrograph *.fits* files:

[2]:
```
import numpy as np
from astropy.io import fits
from pathlib import Path
from IPython.display import Image, display
import lime
```

[3]:
```
def import_osiris_fits(file_address, ext=0):

    # Open the fits file
    with fits.open(file_address) as hdul:
        data, header = hdul[ext].data, hdul[ext].header

    # Reconstruct the wavelength array from the header data
    w_min, dw, n_pix = header['CRVAL1'],  header['CD1_1'], header['NAXIS1']
    w_max = w_min + dw * n_pix
    wavelength = np.linspace(w_min, w_max, n_pix, endpoint=False)

    return wavelength, data, header
```

[5]:
```
# State the scientific data
obsFitsFile = '../sample_data/spectra/gp121903_osiris.fits'
z_obj = 0.19531
norm_flux = 1e-18
```

(continues on next page)

(continued from previous page)

```
# Load spectrum
wave, flux, header = import_osiris_fits(obsFitsFile)
```

Now we can define our `lime.Spectrum` object:

[6]:
```
gp_spec = lime.Spectrum(wave, flux, redshift=z_obj, norm_flux=norm_flux, units_wave='A',␣
↪units_flux='Flam')
gp_spec.plot.spectrum(rest_frame=True)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

The `lime.spectral_bands` allow us to import a dataframe with the of line bands:

[7]:
```
bands_df = lime.line_bands()
```

[8]:
```
bands_df
```

[8]:
```
             wavelength     wave_vac        w1        w2            w3  \
H1_1215A       1215.1108    1215.6699    1100.0    1150.0    1195.00000
C4_1548A       1547.6001    1548.1870    1400.0    1450.0    1530.00000
He2_1640A      1639.7896    1640.3913    1600.0    1630.0    1635.00000
O3_1666A       1665.5438    1666.1500    1600.0    1630.0    1660.00000
C3_1908A       1908.0803    1908.7340    1870.0    1895.0    1898.18782
...                  ...          ...       ...       ...          ...
H1_37395A    37394.7903   37405.5500   37365.0   37385.0   37395.00000
H1_40511A    40511.0344   40522.6900   40490.0   40500.0   40510.00000
H1_46524A    46524.3554   46537.7400   46510.0   46515.0   46534.00000
S4_105075A  105074.7773  105105.0000  104700.0  104900.0  105000.00000
S3_187076A  187076.1931  187130.0000  186700.0  186900.0  187100.00000

                     w4        w5        w6     latex_label particle  \
H1_1215A      1230.000000    1250.0    1300.0    H1-$1215\AA$       H1
C4_1548A      1565.000000    1600.0    1650.0    C4-$1548\AA$       C4
He2_1640A     1645.000000    1700.0    1750.0   He2-$1640\AA$      He2
O3_1666A      1680.000000    1700.0    1750.0    O3-$1666\AA$       O3
C3_1908A      1912.243544    1930.0    1950.0    C3-$1908\AA$       C3
...                   ...       ...       ...             ...      ...
H1_37395A    37415.000000   37425.0   37440.0   H1-$37395\AA$       H1
H1_40511A    40535.000000   40550.0   40570.0   H1-$40511\AA$       H1
H1_46524A    46546.000000   46555.0   46575.0   H1-$46524\AA$       H1
S4_105075A  105200.000000  105300.0  105500.0  S4-$105075\AA$       S4
S3_187076A  187160.000000  187200.0  187300.0  S3-$187076\AA$       S3

           transition  rel_int
H1_1215A          rec        0
C4_1548A          rec        0
He2_1640A         rec        0
O3_1666A          col        0
C3_1908A          sem        0
...               ...      ...
H1_37395A         rec        0
```

(continues on next page)

(continued from previous page)

```
H1_40511A         rec        0
H1_46524A         rec        0
S4_105075A        col        0
S3_187076A        col        0

[86 rows x 12 columns]
```

The default database is too large for an indivual spectrum. You can provide a wavelength range (in the rest frame) to constrain the output lines. Incidentally, you can also provide a `lime.Spectrum` to state the wavelength limits:

[9]:
```
bands_df = lime.line_bands(wave_intvl=gp_spec)
```

[10]:
```
bands_df
```

[10]:

	wavelength	wave_vac	w1	w2	w3	\
Ne5_3426A	3426.0000	3426.0000	3390.000000	3410.000000	3420.000000	
H1_3704A	3703.8013	3704.9132	3671.309441	3681.364925	3700.450000	
O2_3726A	3726.0300	3727.1000	3665.750000	3694.260000	3716.020000	
O2_3729A	3728.8200	3729.8600	3665.750000	3694.260000	3716.020000	
H1_3750A	3750.0998	3751.2244	3664.503848	3675.720417	3746.433569	
H1_3771A	3770.5779	3771.7081	3759.191222	3767.280375	3767.467189	
H1_3798A	3797.8450	3798.9827	3780.949179	3792.078244	3793.707971	
H1_3835A	3835.3309	3836.4789	3823.148476	3829.538777	3831.331855	
Ne3_3869A	3868.7029	3869.8600	3848.429950	3858.099497	3862.724063	
H1_3889A	3888.9950	3890.1577	3842.087829	3861.282614	3880.390000	
H1_3970A	3970.0170	3971.2020	3945.566344	3957.862489	3962.153343	
He1_4026A	4026.1340	4027.3345	4013.837737	4021.250734	4021.502052	
S2_4069A	4068.5368	4069.7490	4058.171690	4064.665875	4065.093604	
H1_4102A	4101.6777	4102.8991	4084.633589	4093.799103	4095.935564	
H1_4340A	4340.4035	4341.6910	4322.549217	4332.509864	4333.660230	
O3_4363A	4363.1421	4364.4360	4297.700000	4323.660000	4356.886082	
He1_4471A	4471.4164	4472.7404	4451.913360	4465.069336	4467.196330	
Fe3_4658A	4658.0240	4659.4000	4637.507567	4650.238986	4653.613936	
He2_4685A	4685.4956	4686.8793	4663.804252	4678.301808	4681.274066	
Ar4_4711A	4711.1891	4712.5800	4664.639652	4680.894598	4707.992152	
Ar4_4740A	4740.0511	4741.4500	4605.870000	4635.300000	4731.467840	
H1_4861A	4861.2582	4862.6910	4809.800000	4836.100000	4848.715437	
He1_4922A	4921.8552	4923.3050	4905.736141	4916.176415	4917.632529	
O3_4959A	4958.8348	4960.2950	4929.281844	4946.732665	4948.310671	
O3_5007A	5006.7664	5008.2400	4971.796688	4984.514249	4995.348943	
N2_5755A	5754.6400	5755.0000	5700.000000	5720.000000	5745.000000	
He1_5876A	5875.5250	5877.2433	5845.715833	5865.712378	5867.148419	
O1_6300A	6300.2076	6302.0460	6282.963147	6293.637954	6294.754436	
S3_6312A	6311.9682	6313.8100	6280.559495	6295.351299	6307.225753	
N2_6548A	6547.9514	6549.8600	6480.030000	6520.660000	6540.100000	
H1_6563A	6562.7192	6564.6320	6480.030000	6520.660000	6545.100000	
N2_6583A	6583.3513	6585.2700	6480.030000	6520.660000	6575.100000	
He1_6678A	6678.0500	6679.9955	6659.110795	6670.102081	6670.734647	
S2_6716A	6716.3386	6718.2950	6686.785257	6706.310255	6707.458647	
S2_6731A	6730.7135	6732.6740	6686.785257	6706.310255	6725.458647	
He1_7065A	7065.1078	7067.1633	7029.100000	7041.640000	7055.220000	
Ar3_7136A	7135.6845	7137.7600	7099.804897	7122.016894	7129.546717	

(continues on next page)

(continued from previous page)

```
O2_7319A    7318.8124  7320.9400  7294.873033  7310.151519  7311.373798
O2_7330A    7329.5494  7331.6800  7294.873033  7310.151519  7311.373798
Ar3_7751A   7750.9894  7753.2400  7731.458685  7743.423077  7744.234222
H1_8392A    8392.2696  8394.7030  8378.449304  8387.961949  8388.394342
H1_8413A    8413.1907  8415.6300  8399.595403  8407.110952  8410.140000
H1_8438A    8437.8276  8440.2740  8396.910000  8408.320000  8432.872822
H1_8467A    8467.1263  8469.5810  8450.080000  8459.460000  8461.707801
H1_8502A    8502.3542  8504.8190  8474.500000  8491.730000  8499.235520
H1_8545A    8545.2540  8547.7310  8515.180000  8536.310000  8539.268883

                  w4           w5           w6    latex_label particle  \
Ne5_3426A  3430.000000  3445.000000  3465.000000  Ne5-$3426\AA$      Ne5
H1_3704A   3709.100000  3758.000000  3764.000000   H1-$3704\AA$       H1
O2_3726A   3743.700000  3754.880000  3767.500000   O2-$3726\AA$       O2
O2_3729A   3743.700000  3754.880000  3767.500000   O2-$3729\AA$       O2
H1_3750A   3756.674784  3775.220000  3792.040000   H1-$3750\AA$       H1
H1_3771A   3775.787336  3776.807987  3782.905843   H1-$3771\AA$       H1
H1_3798A   3803.676860  3807.127865  3816.774280   H1-$3798\AA$       H1
H1_3835A   3840.896149  3844.260000  3852.830000   H1-$3835\AA$       H1
Ne3_3869A  3876.597761  3895.538694  3910.048413  Ne3-$3869\AA$      Ne3
H1_3889A   3899.420000  3905.000000  3950.000000   H1-$3889\AA$       H1
H1_3970A   3976.592758  3979.317175  3989.942405   H1-$3970\AA$       H1
He1_4026A  4032.149534  4033.154014  4040.185370  He1-$4026\AA$      He1
S2_4069A   4073.198887  4080.376366  4086.974251   S2-$4069\AA$       S2
H1_4102A   4108.433334  4110.113141  4119.294537   H1-$4102\AA$       H1
H1_4340A   4347.675999  4350.262697  4360.488238   H1-$4340\AA$       H1
O3_4363A   4372.721331  4390.760000  4416.200000   O3-$4363\AA$       O3
He1_4471A  4477.442236  4480.772156  4496.909458  He1-$4471\AA$      He1
Fe3_4658A  4666.293945  4669.709024  4679.097970  Fe3-$4658\AA$      Fe3
He2_4685A  4691.509985  4718.751912  4733.650293  He2-$4685\AA$      He2
Ar4_4711A  4720.882185  4723.932671  4738.660669  Ar4-$4711\AA$      Ar4
Ar4_4740A  4748.658238  4757.090000  4776.100000  Ar4-$4740\AA$      Ar4
H1_4861A   4876.181741  4883.130000  4908.400000   H1-$4861\AA$       H1
He1_4922A  4928.081261  4930.590000  4939.710000  He1-$4922\AA$      He1
O3_4959A   4970.712011  4972.820372  4984.416360   O3-$4959\AA$       O3
O3_5007A   5024.303156  5027.743260  5043.797081   O3-$5007\AA$       O3
N2_5755A   5765.000000  5800.000000  5820.000000   N2-$5755\AA$       N2
He1_5876A  5885.578291  5888.378245  5901.854752  He1-$5876\AA$      He1
O1_6300A   6307.155360  6319.711710  6327.945925   O1-$6300\AA$       O1
S3_6312A   6319.552583  6321.428405  6328.663719   S3-$6312\AA$       S3
N2_6548A   6555.950000  6627.700000  6661.820000   N2-$6548\AA$       N2
H1_6563A   6575.950000  6627.700000  6661.820000   H1-$6563\AA$       H1
N2_6583A   6590.950000  6627.700000  6661.820000   N2-$6583\AA$       N2
He1_6678A  6688.067396  6689.800671  6698.178167  He1-$6678\AA$      He1
S2_6716A   6725.466528  6744.438091  6759.956250   S2-$6716\AA$       S2
S2_6731A   6741.466528  6744.438091  6759.956250   S2-$6731\AA$       S2
He1_7065A  7079.440000  7087.490000  7104.450000  He1-$7065\AA$      He1
Ar3_7136A  7145.645711  7149.984073  7166.354990  Ar3-$7136\AA$      Ar3
O2_7319A   7338.569503  7340.444100  7357.066344   O2-$7319\AA$       O2
O2_7330A   7338.569503  7340.444100  7357.066344   O2-$7330\AA$       O2
Ar3_7751A  7759.848767  7759.955932  7770.429591  Ar3-$7751\AA$      Ar3
H1_8392A   8398.627642  8398.730000  8409.950000   H1-$8392\AA$       H1
```

(continues on next page)

(continued from previous page)

```
H1_8413A   8418.040000  8423.876407  8432.548195   H1-$8413\AA$       H1
H1_8438A   8443.316230  8452.150208  8464.463395   H1-$8438\AA$       H1
H1_8467A   8473.595713  8474.880000  8493.320000   H1-$8467\AA$       H1
H1_8502A   8509.640000  8511.060000  8533.050000   H1-$8502\AA$       H1
H1_8545A   8552.720348  8555.470000  8574.470000   H1-$8545\AA$       H1

          transition  rel_int
Ne5_3426A        col        0
H1_3704A         rec        0
O2_3726A         col        0
O2_3729A         col        0
H1_3750A         rec        0
H1_3771A         rec        0
H1_3798A         rec        0
H1_3835A         rec        0
Ne3_3869A        col        0
H1_3889A         rec        0
H1_3970A         rec        0
He1_4026A        rec        0
S2_4069A         col        0
H1_4102A         rec        0
H1_4340A         rec        0
O3_4363A         col        0
He1_4471A        rec        0
Fe3_4658A        col        0
He2_4685A        rec        0
Ar4_4711A        col        0
Ar4_4740A        col        0
H1_4861A         rec        0
He1_4922A        rec        0
O3_4959A         col        0
O3_5007A         col        0
N2_5755A         col        0
He1_5876A        rec        0
O1_6300A         col        0
S3_6312A         col        0
N2_6548A         col        0
H1_6563A         rec        0
N2_6583A         col        0
He1_6678A        rec        0
S2_6716A         col        0
S2_6731A         col        0
He1_7065A        rec        0
Ar3_7136A        col        0
O2_7319A         col        0
O2_7330A         col        0
Ar3_7751A        col        0
H1_8392A         rec        0
H1_8413A         rec        0
H1_8438A         rec        0
H1_8467A         rec        0
H1_8502A         rec        0
```

(continues on next page)

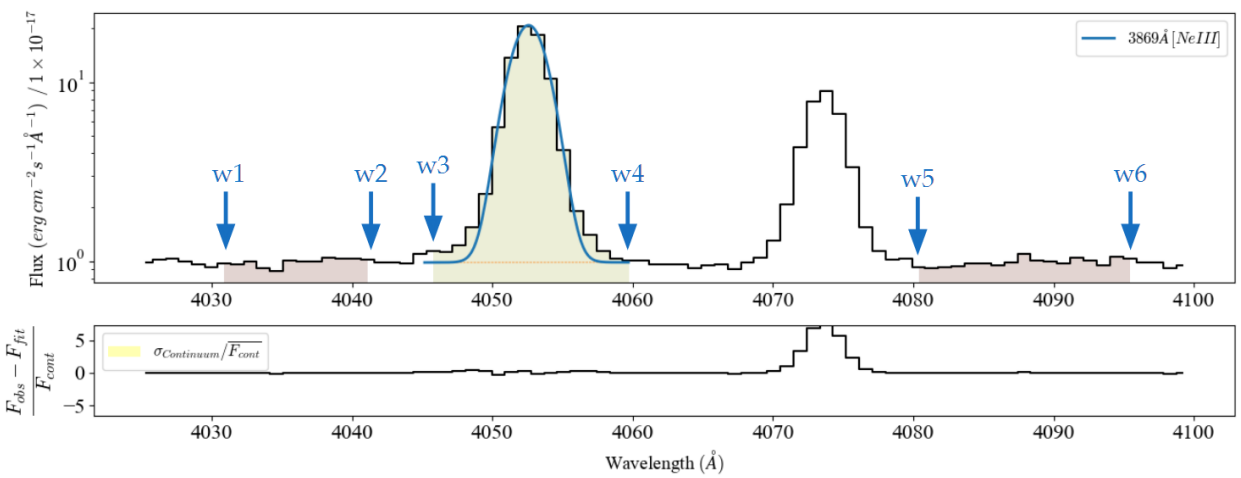
(continued from previous page)

```
H1_8545A        rec        0
```

In addition to the wavelength range, you can also constrain the output table based on a list of ionic species. Moreover, you can also state the labels and wavelength units.

A bands dataframe/file consists in a table, where the line bands wavelength limits are stored space-separated columns. The first column has the line label (in the $LiMe$ format). The remaining 6 columns specify the line location along with two adjacent continua regions. The image belows shows and example with the H1_3889A bands:

[11]:
```
display(Image(filename='../images/bands_definition.png'))
```



Please remember: The band wavelengths must be on the rest frame and sorted from lower to higher values. Finally, make sure that these wavelengths are in the same units as those from your spectrum.

We can save these bands using `lime.save_log` function:

[12]:
```
# Save to a file (if it does not exist already)
bands_df_file = Path('../sample_data/GP121903_bands.txt')
if bands_df_file.is_file() is not True:
    lime.save_log(bands_df, bands_df_file)
```

Running the `Spectrum.check.bands` function opens the interactive window:

[13]:
```
# Review the bands file
gp_spec.check.bands(bands_df_file)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

There are several ways to interact with the plots in this grid: * Right-click on any plot will remove the line from the output bands file. This will switch the background color to red. * Middle-click on any plot will change the line label suffix on the output bands file. The options are blended ("_b"), merged ("_m") and single (no suffix) lines. This will change line label the plot title correspondingly. * Left-click and drag allows you to change the bands limits.

In the lattest case, the The line or continua band adjusted depends on the initical click point. There are some caveats in the window selection: * The plot wavelength range is always 5 pixels beyond the mask bands. Therefore dragging the mouse beyond the mask limits (below w1 or above w6) will change the displayed range. This can be used to move

beyond the original mask limits. * Selections between the w2 and w5 wavelength bands are always assigned to the line region mask as the new w3 and w4 values. * Due to the previous point, to increase the w2 value or to decrease w5 value the user must select a region between w1 and w3 or w4 and w6 respectively.

Each of these adjustments is stored in the input `bands_file`.

By default, the `Spectrum.check.bands` function compares the input bands file with the default bands database. Consequently, if you ran the script above again, you will still have lines with a red background from lines in the database but not in your previous selection. You can provide your own database (file or dataframe) in the `parent_bands` attribute of the `Spectrum.check.bands` function.

At the bottom of the window you can find an slider with the *Band (pixels)*. This slider moves all the bands simultaneously either towards the blue and red limits.

Finally, the `Spectrum.check.bands` function also gives you the oportunity to seet the effect of the spectrum redshift on the bands selection on a pixel bases and save the new value to a text file (or modify the existing value):

[14]:
```
# Adding a redshift file address to store the variations in redshift
redshift_file = '../sample_data/redshift_log.txt'
redshift_file_header, object_ref = 'redshift', 'GP121903'
gp_spec.check.bands(bands_df_file, maximize=True, z_log_address=redshift_file, z_
↪column=redshift_file_header,
                    object_label='object_ref')
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

The redshift new redshift is stored to the output file but the original `redshift` of the `lime.Spectrum` does not change (this behaviour shall be change in a future update).

[1]:
```
%matplotlib notebook
```

CHAPTER

ELEVEN

3) COMPLETE SPECTRUM ANALYSIS

In this example, we shall fit all the emission lines on the spectrum of the Green Pea galaxy GP121903 (see Fernandez et al 2021). You can download this spectrum from the github sample data. This tutorial can also be found as a script and a notebook in the github examples folder.

In this exercise, we are going to follow the recommended $LiMe$ workflow: Using external files with the lines masks and fitting configuration.

Let's start by importing the script packages and defining a function to read the ISIS spectrograph *.fits* files:

[2]:
```
import numpy as np
from astropy.io import fits
from IPython.display import Image, display
from pathlib import Path
import lime
```

[3]:
```
def import_osiris_fits(file_address, ext=0):

    # Open the fits file
    with fits.open(file_address) as hdul:
        data, header = hdul[ext].data, hdul[ext].header

    # Reconstruct the wavelength array from the header data
    w_min, dw, n_pix = header['CRVAL1'],  header['CD1_1'], header['NAXIS1']
    w_max = w_min + dw * n_pix
    wavelength = np.linspace(w_min, w_max, n_pix, endpoint=False)

    return wavelength, data, header
```

11.1 Scientific inputs

Now, we declare and load the scientific data:

[4]:
```
# State the data files
obsFitsFile = '../sample_data/spectra/gp121903_osiris.fits'
lineBandsFile = '../sample_data/osiris_bands.txt'
cfgFile = '../sample_data/osiris.toml'

# Load spectrum
wave, flux, header = import_osiris_fits(obsFitsFile)
```

You can load the bands file as a pandas dataframe using the `load_log` function:1

[5]:
```
# Load line bands
bands = lime.load_log(lineBandsFile)
```

This observation data and the fitting configuration on the `cfgFile` can be read with the `.load_cfg` function

[6]:
```
# Load configuration
obs_cfg = lime.load_cfg(cfgFile)

# Get object redshfit and normalization flux
z_obj = obs_cfg['sample_data']['z_array'][2]
norm_flux = obs_cfg['sample_data']['norm_flux']
```

The `obs_cfg` variable is a dictionary of dictionaries, where each section and option names are keys of the parent and child dictionaries respectively:

[7]:
```
import pprint
pprint.pprint(obs_cfg)
```

```
{'default_line_fitting': {'Ar4_4711A_m': 'Ar4_4711A+He1_4713A',
                          'H1_3889A_m': 'H1_3889A+He1_3889A',
                          'O2_3726A_m': 'O2_3726A+O2_3729A',
                          'O2_7319A_m': 'O2_7319A+O2_7330A'},
 'gp121903_line_fitting': {'H1_6563A_b': 'H1_6563A+N2_6584A+N2_6548A',
                           'N2_6548A_amp': {'expr': 'N2_6584A_amp/2.94'},
                           'N2_6548A_kinem': 'N2_6584A',
                           'O1_6300A_b': 'O1_6300A+S3_6312A',
                           'O3_5007A_b': 'O3_5007A+O3_5007A_k-1',
                           'O3_5007A_k-1_amp': {'expr': '<100.0*O3_5007A_amp',
                                                'min': 0.0},
                           'O3_5007A_k-1_sigma': {'expr': '>2.0*O3_5007A_sigma'},
                           'S2_6716A_b': 'S2_6716A+S2_6731A',
                           'S2_6731A_kinem': 'S2_6716A'},
 'sample_data': {'files_list': ['gp030321_BR.fits',
                                'gp101157_BR.fits',
                                'gp121903_BR.fits'],
                 'norm_flux': 1e-17,
                 'object_list': ['gp030321', 'gp101157', 'gp121903'],
                 'zErr_array': [7.389e-05, 0.000129, 0.0001403],
                 'z_array': [0.16465, 0.14334, 0.19531]}}
```

11.2 Line selection and measurement

Using these data, we can now define the `lime.Spectrum` object for GP121903:

[8]:
```
# Declare LiMe spectrum
gp_spec = lime.Spectrum(wave, flux, redshift=z_obj, norm_flux=norm_flux)
gp_spec.plot.spectrum(label='GP121903', rest_frame=True)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

Before fitting the lines on the mask file, it is a good practice to confirm their presence of the spectrum. One way to do that is using the `Spectrum.line_detection` function:

[9]:
```
# Find the lines
match_bands = gp_spec.line_detection(bands, cont_fit_degree=[3, 7, 7, 7], cont_int_
↪thres=[5, 3, 2, 0.7])
```

This function normalizes the continuum in a loop to find the pixels above and below a certain threshold. Afterwards, the peaks or throughs found are compaired against the input line bands for a succesful line detection.

You can plot a set of bands in you spectrum with the `line_bands` argument in the `.plot.spectrum` function:

[10]:
```
gp_spec.plot.spectrum(label='GP121903 matched lines', line_bands=match_bands, log_
↪scale=True)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

You can save your object bands with the `.save_log` function:

[11]:
```
# Saving GP121903 bands
obj_bands_file = '../sample_data/gp121903_bands.txt'
lime.save_log(match_bands, obj_bands_file)
```

Now we are going to measure all the lines in this bands dataframe using the `Spectrum.fit.frame` function. This function will read the `obj_bands_file` and proceed to fit all the lines taking into consideration the lines fitting configuration from the `obs_cfg` dictionary.

[12]:
```
gp_spec.fit.frame(obj_bands_file, obs_cfg, id_conf_prefix='gp121903')
```

```

Line fitting progress:
[==========] 100% of 30 lines (Ar3_7136A))
```

The `Spectrum.fit.frame` function allows the user to combine a "default" fitting configuration with an "individual" fitting configuration. The former configuration parameters are specified with `default_conf_prefix` argument, while the latter uses the `id_conf_prefix`. In this example, these parameters would be:

[13]:
```
print(f'# Default line fitting')
pprint.pprint(obs_cfg['default_line_fitting'])

print(f'\n# Individual line fitting')
pprint.pprint(obs_cfg['gp121903_line_fitting'])
```

```
# Default line fitting
{'Ar4_4711A_m': 'Ar4_4711A+He1_4713A',
 'H1_3889A_m': 'H1_3889A+He1_3889A',
 'O2_3726A_m': 'O2_3726A+O2_3729A',
 'O2_7319A_m': 'O2_7319A+O2_7330A'}

# Individual line fitting
{'H1_6563A_b': 'H1_6563A+N2_6584A+N2_6548A',
 'N2_6548A_amp': {'expr': 'N2_6584A_amp/2.94'},
 'N2_6548A_kinem': 'N2_6584A',
 'O1_6300A_b': 'O1_6300A+S3_6312A',
 'O3_5007A_b': 'O3_5007A+O3_5007A_k-1',
```

(continues on next page)

(continued from previous page)

```
 'O3_5007A_k-1_amp': {'expr': '<100.0*O3_5007A_amp', 'min': 0.0},
 'O3_5007A_k-1_sigma': {'expr': '>2.0*O3_5007A_sigma'},
 'S2_6716A_b': 'S2_6716A+S2_6731A',
 'S2_6731A_kinem': 'S2_6716A'}
```

Please remember: The `Spectrum.fit.frame` function **updates** the default parameters with the new ones. This means that only the common entries in the default configurations are replaced. Consequently, in the fittings of GP121903 the configuration used would be:

[14]:
```
pprint.pprint({**obs_cfg['default_line_fitting'], **obs_cfg['gp121903_line_fitting']})
```

```
{'Ar4_4711A_m': 'Ar4_4711A+He1_4713A',
 'H1_3889A_m': 'H1_3889A+He1_3889A',
 'H1_6563A_b': 'H1_6563A+N2_6584A+N2_6548A',
 'N2_6548A_amp': {'expr': 'N2_6584A_amp/2.94'},
 'N2_6548A_kinem': 'N2_6584A',
 'O1_6300A_b': 'O1_6300A+S3_6312A',
 'O2_3726A_m': 'O2_3726A+O2_3729A',
 'O2_7319A_m': 'O2_7319A+O2_7330A',
 'O3_5007A_b': 'O3_5007A+O3_5007A_k-1',
 'O3_5007A_k-1_amp': {'expr': '<100.0*O3_5007A_amp', 'min': 0.0},
 'O3_5007A_k-1_sigma': {'expr': '>2.0*O3_5007A_sigma'},
 'S2_6716A_b': 'S2_6716A+S2_6731A',
 'S2_6731A_kinem': 'S2_6716A'}
```

11.3 Plotting and saving the measurements

The fitted profiles can be over-plotted on the input spectrum setting the `include_fits=True` parameter on the `Spectrum.plot.spectrum` function

[15]:
```
# Display the fits on the spectrum
gp_spec.plot.spectrum(include_fits=True, rest_frame=True)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

Additionally, you can also plot the results as a grid using the `Spectrum.plot.grid`

[16]:
```
# Display a grid with the fits
gp_spec.plot.grid(rest_frame=True)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

You can plot individual line fittings with the `Spectrum.plot.band` function:

[17]:
```
gp_spec.plot.bands('H1_6563A')
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

Finally, we save the tabulated measurements with:

[18]:
```
# Save the data
gp_spec.save_log('../sample_data/example3_linelog.fits', page='GP121903_a')
```

or

[20]:
```
lime.save_log(gp_spec.log, '../sample_data/example3_linelog.fits', page='GP121903b')
```

[1]:
```
%matplotlib notebook
```

CHAPTER

TWELVE

4) IFU SPATIAL MASKING

In this tutorial, we perform a pre-analysis of an IFU (Integral Field Unit) cube. These data sets provide both spatial and spectroscopic information of astronomical bodies. You can download this tutorial as a python script or a jupyter notebook.

The workflow for an IFU observation is very similar to the one we saw in the previous tutorials for long-slit spectra. However, in order to preserve the spatial information and to maximise the quality of the measurements, it is recommended to use spatial masks. These provide two advantages: * In most cases, the scientific data does not cover full IFU field of view. To maximise your time and the computational resources, the non-scientific/noisy regions should be excluded from the analysis. * In resolved observations, the phenomena responsible for the observed radiation do not remain constant. This means that the properties of the emission lines can vary dramatically in a few spaxels. Consequently, your fittings should adapt to this spatial behaviour. Binary masks provide the means to personalise $LiMe$ fittings.

Let's start by downloading one IFU data cube from the MANGA survey. In this tutorial, we will analyze SHOC579, a compact galaxy with an intense star forming region:

[2]:
```
import urllib.request
from astropy.io import fits
from pathlib import Path
from astropy.wcs import WCS
from sys import stdout
import lime
```

[3]:
```
# Web address and save file location
cube_url = 'https://data.sdss.org/sas/dr17/manga/spectro/redux/v3_1_1/8626/stack/manga-
↪8626-12704-LOGCUBE.fits.gz'
cube_address = Path('../sample_data/spectra/manga-8626-12704-LOGCUBE.fits.gz')
```

The following command will download the IFU datacube if it isn't already in the `sample_data` folder. This may take some time. To keep a better track of the process you can use the `cube_url` web link to manually download the cube and place it on the `sample_data` folder,

[4]:
```
# Function to display the download progress on the terminal
def progress_bar(count, block_size, total_size):
    percent = int(count * block_size * 100 / total_size)
    stdout.write("\rDownloading...%d%%" % percent)
    stdout.flush()

# Download the cube file if not available (this may take some time)
```

(continues on next page)

(continued from previous page)

```
if cube_address.is_file() is not True:
    urllib.request.urlretrieve(cube_url, cube_address, reporthook=progress_bar)
    print(" Download complete!")
else:
    print('Observation found in folder')
```

```
Observation found in folder
```

We proceed to load the configuration file and observation data

[5]:
```
# Load the configuration file:
cfgFile = '../sample_data/manga.toml'
obs_cfg = lime.load_cfg(cfgFile)

# Observation properties
z_obj = obs_cfg['SHOC579']['redshift']
norm_flux = obs_cfg['SHOC579']['norm_flux']

# Open the MANGA cube fits file
with fits.open(cube_address) as hdul:
    wave = hdul['WAVE'].data
    flux_cube = hdul['FLUX'].data * norm_flux
    hdr = hdul['FLUX'].header
```

$LiMe$ includes several tools to display IFU data. To include the spatial coordinates on these plots you need to import the World Coordinates System WCS from your *.fits* file header. This can be done using astropy WCS class:

[6]:
```
# Declaring the world coordinate system from the header to display on the plots
wcs = WCS(hdr)
```

```
WARNING: FITSFixedWarning: PLATEID = 8626 / Current plate
a string value was expected. [astropy.wcs.wcs]
WARNING: FITSFixedWarning: 'datfix' made the change 'Set MJD-OBS to 57277.000000 from␣
↪DATE-OBS'. [astropy.wcs.wcs]
```

At this point we can define a $LiMe$ `Cube` object for our observation:

[7]:
```
# Define a LiMe cube object
shoc579 = lime.Cube(wave, flux_cube, redshift=z_obj, norm_flux=norm_flux, wcs=wcs)
```

From this object, we can access the functions to fit and plot the data. For example, we can use the `.plot.cube` to spatially plot a spectral band:

[8]:
```
# Check the spaxels interactively
shoc579.plot.cube(6563, line_fg=4363)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
S:\anaconda3\envs\lime\lib\site-packages\astropy\visualization\wcsaxes\core.py:254:␣
↪UserWarning: Log scale: values of z <= 0 have been masked
  cset = super().contour(*args, **kwargs)
```

In the previous function the first argument (`line`) specifies the band for the plot background. Since we did not specify a `band` wavelength interval, $LiMe$ will get the band from the closest transition on the default database (in this case

$H\alpha$). The user can also specify a `percentile_bg` for the lowest flux limit. For example:

[9]:
```
shoc579.plot.cube('H1_6563A', line_fg=4363, min_pctl_bg=80)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
S:\anaconda3\envs\lime\lib\site-packages\astropy\visualization\wcsaxes\core.py:254:␣
↪UserWarning: Log scale: values of z <= 0 have been masked
  cset = super().contour(*args, **kwargs)
```

The `line_fg` argument allows the user to specify a band for the intensity contours. In the previous commands, we used the $[OIII]4363$ temperature diagnostic line.

To specify the number of contours, the user can provide a list percentiles with the `percentiles_fg` argument for the flux limits:

[10]:
```
shoc579.plot.cube('H1_6563A', line_fg='O3_4363A', cont_pctls_fg=(85, 90, 95, 99))
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
S:\anaconda3\envs\lime\lib\site-packages\astropy\visualization\wcsaxes\core.py:254:␣
↪UserWarning: Log scale: values of z <= 0 have been masked
  cset = super().contour(*args, **kwargs)
```

From these contours, you are likely to conclude that there are two star-forming clusters. You should, however, inspect the spaxels spectra on these regions. You can do that with the `.check.cube` function:

[12]:
```
# Manually add/remove spaxels to the spatial mask
shoc579.check.cube('H1_6563A', line_fg='O3_4363A')
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
S:\anaconda3\envs\lime\lib\site-packages\astropy\visualization\wcsaxes\core.py:254:␣
↪UserWarning: Log scale: values of z <= 0 have been masked
  cset = super().contour(*args, **kwargs)
```

On the right-hand side of the new plot, you have the spectrum corresponding to the spaxel with the red cross. You can change the active spaxel with a right-clik on the left-hand side plot.

Inspecting the lower region on the IFU field of view, this radiation does not correspond to SHOC579 but a star. Consequently, the flux percentiles in this region are not mapping the ionized emission from the $[OIII]4363$ transition but the continuum from a nearby stellar atmosphere.

For a more detailed analysis, we can use the `.spatial_masker` to generate a binary mask. By default, these masks are generated based in the bands flux intensity (as in the case of the `.plot.cube`). However, we can specify an emission signal-to-noise to compute this mask:

[13]:
```
# Generate a spatial mask as a function of the band flux
# Generate a spatial mask as a function of the signal to noise
spatial_mask = '../sample_data/SHOC579_mask.fits'
shoc579.spatial_masking('O3_4363A', param='SN_line', contour_pctls=[93, 96, 99], output_
↪address=spatial_mask)
```

```
D:\Pycharm Projects\lime\src\lime\observations.py:1014: RuntimeWarning: invalid value␣
↪encountered in true_divide
  param_image = (np.sqrt(2 * n_pixels * np.pi) / 6) * (Amp_image / std_image)
```

We can visualize this mask using the `.plot.cube` function:

[14]:
```
shoc579.plot.cube('H1_6563A', masks_file=spatial_mask)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

Using this criterium, the masks are now constrained to the star-forming burst. You can adjust the mapping and number of masks using the `percentiles` argument. In some cases, however, this is not enough. For a more meticulous analysis, you can load this mask with the `.check.cube` function:

[15]:
```
# Manually add/remove spaxels to the spatial mask
shoc579.check.cube('H1_6563A', masks_file=spatial_mask)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

In the new plot, you can can select the masks available within the `spatial_mask` file we generated before. If you left double-click on a spaxel it will be added to the current mask if it was not (by default this spaxel will be removed from its previous mask). Otherwise, if the spaxel was already on the current mask, it will be removed from the selection.

You can use this manual selection, for example, to generate masks with certain lines o profiles. This will be very usefull for the fittings in the next tutorial.

[1]:
```
%matplotlib notebook
```

CHAPTER

THIRTEEN

5) IFU LINE FITTING

In this tutorial, we are going to fit the emission lines on the spaxels from the masks, we defined in the previous tutorial. You can download this tutorial as a python script and a jupyter notebook.

Let's start by importing the libraries we need:

[2]:
```
from pathlib import Path
from astropy.io import fits
from astropy.wcs import WCS
from IPython.display import Image, display
import lime
```

Let's define the inputs and outputs paths

[3]:
```
# State the data location
cfg_file = Path('../sample_data/manga.toml')
cube_file = Path('../sample_data/spectra/manga-8626-12704-LOGCUBE.fits.gz')
bands_file_0 = Path('../sample_data/SHOC579_MASK0_bands.txt')
spatial_mask_file = Path('../sample_data/SHOC579_mask.fits')
output_lines_log_file = Path('../sample_data/SHOC579_log.fits')
```

Now we read the observation and treatment configuration

[4]:
```
# Load the configuration file:
obs_cfg = lime.load_cfg(cfg_file)

# Observation properties
z_obj = obs_cfg['SHOC579']['redshift']
norm_flux = obs_cfg['SHOC579']['norm_flux']
```

With this information, we can load the scientific data and define the `lime.Cube` object:

[5]:
```
# Open the MANGA cube fits file
with fits.open(cube_file) as hdul:
    wave = hdul['WAVE'].data
    flux_cube = hdul['FLUX'].data * norm_flux
    hdr = hdul['FLUX'].header

# Define a LiMe cube object
shoc579 = lime.Cube(wave, flux_cube, redshift=z_obj, norm_flux=norm_flux)
```

At this point, you have diferent strategies to fit lines on this IFU data set. For example, you can recover the spectrum from a spaxel using the flux cube coordinates.

Please remember: This is a numpy array, the first and second indeces correspond to the vertical and horizontal axis respectively. These are the spatial axis on an IFU observation. The third index corresponds to the cube depth axis or the IFU wavelength array. The origin in this numerical array is located on the upper-left-front corner on the 3D array.

[6]:
```
# Extract one spaxel (idx Y, idx X):
spaxel = shoc579.get_spectrum(39, 40)
```

We can use the `.fit.frame` function we visisted in the 3rd tutorial:

[7]:
```
spaxel.fit.frame(bands_file_0, obs_cfg, line_detection=True, id_conf_prefix='MASK_0',␣
↪progress_output='counter')
```

```
Line fitting progress:
1/35 lines) (H1_3704A)
2/35 lines) (O2_3726A_b)
3/35 lines) (H1_3750A)
4/35 lines) (H1_3771A)
5/35 lines) (H1_3798A)
6/35 lines) (H1_3835A)
7/35 lines) (Ne3_3869A)
8/35 lines) (H1_3889A_m)
9/35 lines) (H1_3970A)
10/35 lines) (He1_4026A)
11/35 lines) (S2_4069A)
12/35 lines) (H1_4102A)
13/35 lines) (H1_4341A)
14/35 lines) (O3_4363A)
15/35 lines) (He1_4471A)
16/35 lines) (Ar4_4711A_m)
17/35 lines) (H1_4861A_b)
18/35 lines) (He1_4922A)
19/35 lines) (O3_4959A_b)
20/35 lines) (O3_5007A_b)
21/35 lines) (He1_5876A)
22/35 lines) (O1_6300A)
23/35 lines) (H1_6563A_b)
24/35 lines) (He1_6678A)
25/35 lines) (S2_6716A)
26/35 lines) (S2_6731A)
27/35 lines) (He1_7065A)
28/35 lines) (Ar3_7136A)
29/35 lines) (O2_7319A_b)
30/35 lines) (Ar3_7751A)
31/35 lines) (H1_8863A)
32/35 lines) (H1_9015A)
33/35 lines) (S3_9069A)
34/35 lines) (H1_9229A)
35/35 lines) (S3_9531A_b)
```

The fitted profiles can be displayed using the `.plot.spectrum` function:

[8]:
```
spaxel.plot.spectrum(include_fits=True, rest_frame=True)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

To recover the array coordinates from your spatial mask you can use the `lime.load_spatial_mask`:

[9]:
```
masks_dict = lime.load_spatial_mask(spatial_mask_file, return_coords=True)
```

By default this dictionary contains the mask array (as boolean array) and header for every extension in the input file. However if you set the argument `return_coords=True` you will get the array coordinates for every `True` value in the spatial mask. You can use this information to analyse the spectra lines:

[10]:
```
masks_dict = lime.load_spatial_mask(spatial_mask_file, return_coords=True)
for i, coords in enumerate(masks_dict['MASK_0']):
    print(f'Spaxel {i}) Coordinates {coords}')
    idx_Y, idx_X = coords
    spaxel = shoc579.get_spectrum(idx_Y, idx_Y)
    spaxel.fit.frame(bands_file_0, obs_cfg, line_list=['H1_6563A_b'], id_conf_prefix=
↪'MASK_0', plot_fit=False, progress_output=None)
```

```
Spaxel 0) Coordinates [35 35]
Spaxel 1) Coordinates [35 36]
Spaxel 2) Coordinates [35 38]
Spaxel 3) Coordinates [35 39]
Spaxel 4) Coordinates [36 35]
Spaxel 5) Coordinates [36 36]
Spaxel 6) Coordinates [36 37]
Spaxel 7) Coordinates [36 38]
Spaxel 8) Coordinates [36 39]
Spaxel 9) Coordinates [36 40]
Spaxel 10) Coordinates [37 35]
Spaxel 11) Coordinates [37 36]
Spaxel 12) Coordinates [37 37]
Spaxel 13) Coordinates [37 38]
Spaxel 14) Coordinates [37 39]
Spaxel 15) Coordinates [37 40]
Spaxel 16) Coordinates [38 34]
Spaxel 17) Coordinates [38 35]
Spaxel 18) Coordinates [38 36]
Spaxel 19) Coordinates [38 37]
Spaxel 20) Coordinates [38 38]
Spaxel 21) Coordinates [38 39]
Spaxel 22) Coordinates [38 40]
Spaxel 23) Coordinates [39 34]
Spaxel 24) Coordinates [39 35]
Spaxel 25) Coordinates [39 36]
Spaxel 26) Coordinates [39 37]
Spaxel 27) Coordinates [39 38]
Spaxel 28) Coordinates [39 39]
Spaxel 29) Coordinates [40 35]
Spaxel 30) Coordinates [40 36]
Spaxel 31) Coordinates [40 37]
Spaxel 32) Coordinates [40 38]
```

For a more sophisticated analysis you can use the `.fit.spatial_mask` function. To treat serveral spaxels in a efficient workflow, you can use `.fit.spatial_mask`. This function allows you to fit some or all the masks in the input `spatial_mask_file`. Moreover, it will save the results into a *.fits* specified in the `output_log` parameter. Each page on the output file corresponds to a spaxel. The default page name is **idxY-idxX_LINELOG**. You can change the default extension suffix with the `log_ext_suffix='_LINELOG'` argument.

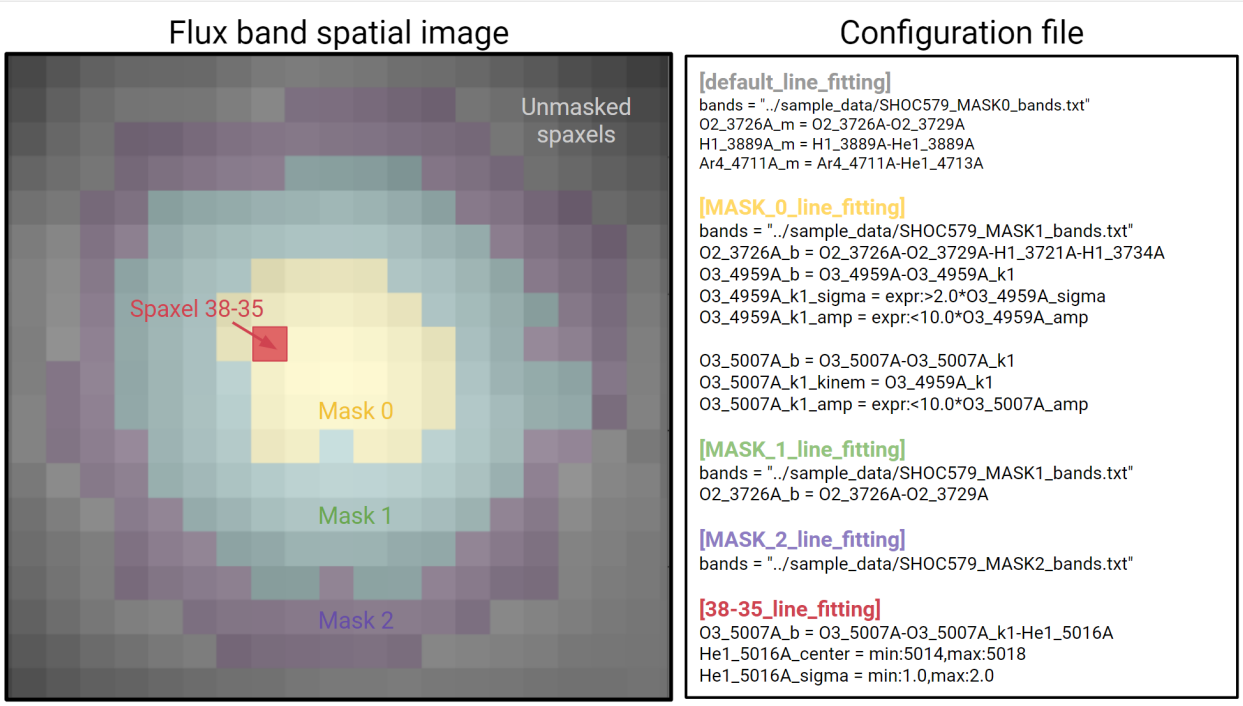
[11]:
```
# Fit the lines in all the masks spaxels
shoc579.fit.spatial_mask(spatial_mask_file, fit_conf=obs_cfg, line_detection=True,␣
↪output_address=output_lines_log_file)
```

```
Spatial mask 1/3) MASK_0 (33 spaxels)
[==========] 100% of mask (spaxel coordinate. 40-38)
1578 lines measured in 0.54 minutes.

Spatial mask 2/3) MASK_1 (97 spaxels)
[==========] 100% of mask (spaxel coordinate. 43-39)
3144 lines measured in 1.07 minutes.

Spatial mask 3/3) MASK_2 (97 spaxels)
[==========] 100% of mask (spaxel coordinate. 54-26)
1341 lines measured in 0.56 minutes.

Joining spatial log files (MASK_0,MASK_1,MASK_2) -> ..\sample_data\SHOC579_log.fits
[==========] 100% of log files combined
```

Please remember: The `.fit.spatial_mask` function cannot add or update measurements on an existing *.fits* file. It will always overwrite an existing file on the provided `output_log` path.

However, the most import feature from the `.fit.spatial_mask` is the posibility to administrate the configuration of your fittings. The image below shows the spatial masks of SHOC579 alongside part of its configuration `manga.cfg` file:

[12]:
```
display(Image(filename='../images/mask_conf_diagram.png'))
```



In these measurements the `.fit.spatial_mask` adjust the fitting configuration on three levels:

- At the lowest level, the *[default_line_fitting]* section provides a configuration fitting for all spaxels in all the masks.The name of this section can be modified with the `default_conf_prefix="default"`
- The *[MASK_0_line_fitting]*, *[MASK_1_line_fitting]* and *[MASK_2_line_fitting]* sections provide the fitting configuration for each corresponding mask. These entries **update** the information from the *[default_line_fitting]*. This means that:
  - In all the masked spaxels `H1_3889A_m = H1_3889A-He1_3889A` and `Ar4_4711A_m = Ar4_4711A-He1_4713A`
  - In `MASK_0` the line `O2_3726A_b` has four components while in `MASK_1` it only has two
  - Since `MASK_2` does not have any items, the fitting configuration is the same as in *[default_line_fitting]*. Consequently this section can be excluded from the file.
- At the highest level the *[38-35_line_fitting]* section updates the fitting properties of the spaxel on red. Consequently, just in that spaxel `O3_5007A_b = O3_5007A-O3_5007A_k1-He1_5016A`. Again this entry **updates** the information from the *[MASK_0_line_fitting]* and the *[default_line_fitting]* sections.

We can confirm this configuration by checking this spaxel, the only one with the fitting of the $HeI5016$:

[13]:
```
# Check the individual spaxel fitting configuration
spaxel = shoc579.get_spectrum(38, 35)
spaxel.load_log(output_lines_log_file, page='38-35_LINELOG')
spaxel.plot.bands('He1_5016A')
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

Additionally, you can adjust the `.fit.spatial_mask` measurements with these arguments:

- In the `mask_name_list=` you can specify the masks to use from the `spatial_mask_file` (the default is all of them).

- The `fit_conf=obs_cfg` argument assumes that there are several sections, one for spatial mask, with the **[MASK-NAME_line_fitting]** naming style. Using this formating you can apply a different fitting configuration for each mask. In general you will start with the spaxels with the highest S/N and more complex line profiles and move towards simpler configurations. You can see this on the input manga.cfg file.
- This function can only save the measurements as a *.fits* file. This file is specified using the `output_log=output_lines_log_file`. Each page on the *.fits* file corresponds to a spaxel. The default page name is **idxY-idxX_LINELOG**. You can change the default extension suffix with the `log_ext_suffix='_LINELOG'` argument.
- The `n_save=100` argument states after how many spaxels the *.fits* file with the measurements is saved on the hard drive. Additionally, the *.fits* file is always saved at the last spaxel of a mask.

Finally, you can visualize the fitted profiles on the output *.fits* using the `.plot.cube` function:

[14]:
```
shoc579.check.cube('H1_6563A', wcs=WCS(hdr), lines_log_file=output_lines_log_file)
```

```
WARNING: FITSFixedWarning: PLATEID = 8626 / Current plate
a string value was expected. [astropy.wcs.wcs]
WARNING: FITSFixedWarning: 'datfix' made the change 'Set MJD-OBS to 57277.000000 from␣
↪DATE-OBS'. [astropy.wcs.wcs]
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

In the next tutorial, we shall see how you can use $LiMe$ functions to extract the measurements from a spatial lines log such as the one we just generated.

[1]:
```
%matplotlib notebook
```

CHAPTER

FOURTEEN

6) REVIEWING IFU RESULTS

In this tutorial, we are going to explore some tools to check the measurements from the previous tutorial. You can download this tutorial as a python script and a jupyter notebook. Make sure you have downloaded the MANGA IFU of SHOC579. You can find the link in the 4th tutorial.

Let's start by importing the libraries we need:

[2]:
```
# This try import is only necessary for the developper to compile the online␣
↪documentation from this notebook
try:
    import lime
except ImportError:
    import sys
    sys.path.append('../../src')
    import lime

from pathlib import Path
from astropy.io import fits
from astropy.wcs import WCS
from matplotlib import pyplot as plt
from IPython.display import Image, display
import lime
```

Let's define the inputs and outputs paths

[6]:
```
# State the data location
cfg_file = Path('../sample_data/manga.toml')
cube_file = Path('../sample_data/spectra/manga-8626-12704-LOGCUBE.fits.gz')
bands_file_0 = Path('../sample_data/SHOC579_MASK0_bands.txt')
spatial_mask_file = Path('../sample_data/SHOC579_mask.fits')
output_lines_log_file = Path('../sample_data/SHOC579_log.fits')
```

Now we read the observation and treatment configuration

[7]:
```
# Load the configuration file:
obs_cfg = lime.load_cfg(cfg_file)

# Observation properties
z_obj = obs_cfg['SHOC579']['redshift']
norm_flux = obs_cfg['SHOC579']['norm_flux']
```

With this information, we can load the scientific data and define the `lime.Cube` object:

[8]:
```
# Open the MANGA cube fits file
with fits.open(cube_file) as hdul:
    wave = hdul['WAVE'].data
    flux_cube = hdul['FLUX'].data * norm_flux
    hdr = hdul['FLUX'].header
```

We are going to include in the `lime.Cube` the World Coordinates System (WCS) from the observation. This way, the spatial coordinates will be included on the output plots and *.fits* files products:

[9]:
```
# WCS from the obsevation header
wcs = WCS(hdr)

# Define a LiMe cube object
shoc579 = lime.Cube(wave, flux_cube, redshift=z_obj, norm_flux=norm_flux, wcs=wcs)
```

```
WARNING: FITSFixedWarning: PLATEID = 8626 / Current plate
a string value was expected. [astropy.wcs.wcs]
WARNING: FITSFixedWarning: 'datfix' made the change 'Set MJD-OBS to 57277.000000 from␣
↪DATE-OBS'. [astropy.wcs.wcs]
```

You can check the lines fitted from at a certain spaxel using the interactive `Cube.check.cube` function:

[10]:
```
shoc579.check.cube('H1_6563A', lines_log_file=output_lines_log_file, masks_file=spatial_
↪mask_file)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

Additinally, you can extract individual `lime.Spectrum` from this cube and use the `Spectrum.plot` functions:

[11]:
```
# Check the individual spaxel fitting configuration
spaxel = shoc579.get_spectrum(38, 35)
spaxel.load_log(output_lines_log_file, page='38-35_LINELOG')
spaxel.plot.grid()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

You can use $LiMe$ functions to extract and convert indivudal spaxels lines logs from the original *.fits* file:

[12]:
```
spaxel_log_df = lime.load_log(output_lines_log_file, page='38-35_LINELOG')
lime.save_log(spaxel_log_df, '../sample_data/38-35_linelog.txt')
```

14.1 Spatial parameter maps

To take advantage of the IFU spatial data, you will want to explore the evolution of the physical parameters computed from these line measurements. You can use the `lime.save_parameter_maps` function:

[13]:
```
# Export the measurements log as maps:
param_list = ['intg_flux', 'intg_flux_err', 'gauss_flux', 'gauss_flux_err', 'v_r', 'v_r_
↪err']
lines_list = ['H1_4861A', 'H1_6563A', 'O3_4363A', 'O3_4959A', 'O3_5007A', 'S3_6312A',
```

(continues on next page)

(continued from previous page)

```
↪'S3_9069A', 'S3_9531A']
lime.save_parameter_maps(output_lines_log_file, '../sample_data/', param_list, lines_
↪list,
                         mask_file=spatial_mask_file, output_file_prefix='SHOC579_',␣
↪wcs=wcs)
```

```
[========= ] 99% of spaxels from file (..\sample_data\SHOC579_log.fits) read (227 total␣
↪spaxels)
```

Please remember: The inputs and outputs of the `lime.save_parameter_maps` can only be *.fits* files.

This function produces different outputs depending on the input parameters:

- The `parameter_list` argument establishes which parameters from the input `output_lines_log_file` will be exported into a *.fits* file. There will be *.fits* file per item on the `parameter_list`. These files will be stored on the `output_folder` using `output_files_prefix` and the item name. The item name must follow the parameter notation of the logs measurements.
- The `line_list` arguments establishes, which line measurements are exported to the output parameter *.fits* files. Each file will have one page per line.
- The user can specify a binary mask with the `spatial_mask_file` argument. By default default it will use all the masks extensions on the input *.fits* unless the user specifies certain masks with the `ext_mask` argument.

Please remember: It is important to provide an input `spatial_mask_file` argument to make sure only spaxels with scientific data are queried on the input `output_lines_log_file`.

At this point we plot some flux ratio diagnostics from theses maps. Let's define them:

[14]:
```
# State line ratios for the plots
lines_ratio = {'H1': ['H1_6563A', 'H1_4861A'],
               'O3': ['O3_5007A', 'O3_4959A'],
               'S3': ['S3_9531A', 'S3_9069A']}
```

Since some of these lines are blended we are going to use the gaussian fluxes

[15]:
```
# State the parameter map file
fits_file = f'../sample_data/SHOC579_gauss_flux.fits'
```

Let's loop through these ratios and make the plots:

[16]:
```
# Loop through the line ratios
for ion, lines in lines_ratio.items():

    # Recover the parameter measurements
    ion_array, wave_array, latex_array = lime.label_decomposition(lines)
    ratio_map = fits.getdata(fits_file, lines[0]) / fits.getdata(fits_file, lines[1])

    # Get the astronomical coordinates from one of the headers of the lines log
    hdr = fits.getheader(fits_file, lines[0])
    wcs_maps = WCS(hdr)
```

(continues on next page)

(continued from previous page)

```
    # Create the plot
    fig = plt.figure(figsize=(10, 10))
    ax = fig.add_subplot(projection=wcs_maps, slices=('x', 'y'))
    im = ax.imshow(ratio_map)
    cbar = fig.colorbar(im, ax=ax)
    ax.update({'title': f'SHOC579 flux ratio: {latex_array[0]} / {latex_array[1]}',
↪'xlabel': r'RA', 'ylabel': r'DEC'})
    plt.show()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

From these plots we can draw the following conclusions:

- The $\frac{H\alpha}{H\beta}$ provides a characterisation of the extinction on the SHOC579. This ratio should be above around 2.98, the theoretical emissivity ratio for these lines which is weakly dependant on the physical condition density. However, it seems some pixels have aprubt changes: Could this be because in some spaxels of the region 1, we didn't include the wide component of $H\alpha$. Does the profile fitting improve if we include it in this region?
- The $\frac{[OIII]5007}{[OIII]4959}$ ratio should remain constant around 3 independently of the physical conditions. Could diversions be caused by the profile fitting? You could check how this map changes if we use the integrated flux instead of the gaussian narrow component. Could it be that this line is too intense and therefore the CCD measurement lies outside the linearity region?
- The $\frac{[SIII]9531}{[SIII]9069}$ ratio should remain constant around 2.47 independently of the physical conditions. This seems to be the case for most of the galaxy. In some spaxels, however, the value seems to be lower/higher. Could this be explained by the telluric features in the proximity of the lines which could be harder to correct as the ionization radiation becomes weaker?. It may be necesarry to inspect these spaxels individually to confirm the if one or both lines are heavily contaminated.

As you may have noticed, at this point we are starting to derive scientific conclusions from the data. In any implementation of $LiMe$, the user is encouraged to review every step to confirm that the measurements agree with the expected results.

INDEX