Automated GILDAS-CLASS Pipeline

(classpipeline.py, classlinesearch.py, classreduction.py, classaverage.py, classcombine.py)

User Guide

Andrés Megías Toledano
Centre for Astrobiology (CAB), Madrid

Version 1.1
April 2023

Index

1. Introduction

The Automated GILDAS-CLASS Pipeline (GCP) is a set of Python 3 scripts (classpipeline.py, classlinesearch.py, classreduction.py, classaverage.py, and classcombine.py) that provide a simple interface for running the software CLASS, from the package GILDAS,[1] in a semi-automatic way. In this way, one can extract and organize spectra from data files in CLASS format (for example, .30m and .40m), reduce them, and even combine or average them once they are reduced.

The scripts need the libraries PyYAML, NumPy, SciPy, Pandas, Astropy, and Matplotlib. Additionally, there are two more optional scripts written in Julia, classlinesearch.jl and classreduction.jl, that can be used instead of the same-name Python scripts. These scripts require the packages StatsBase, Statistics, Random, Distributions, Interpolations,

1. https://www.iram.fr/IRAMFR/GILDAS/

RollingFunctions, Formatting, ArgParse, YAML, FITSIO, DelimitedFiles, SciPy, and PyPlot.

The main script is classpipeline.py, which can call the scripts classlinesearch.py, classreduction.py, classaverage.py, and classcombine.py (and also classlinesearch.jl and classreduction.jl).

2. Running of the codes

First of all, you need to install GILDAS and CLASS to run this pipeline. Then, in order to run the codes, it is necessary to specify some options in various configuration files. In general, for running this code (classpipeline.py) in Linux or MacOS, we have to write the following command line in the terminal, being in the folder of the script:

```
python3 classpipeline.py <config-path> <arguments>
```

where <config-path> is the path of the configuration file and <arguments> are the running arguments of the script.[2] If the file classpipeline.py is not in the current folder, but in a folder with path <folder>, we should write <folder>/classpipeline.py instead of classpipeline.py. Moreover, if the script has execution permissions and is included in a folder of executable files, the term python3 can be removed from the command line, and the script will be run at any folder.

Therefore, it is recommended to have all the scripts of the pipeline in the same folder, give them execution permissions (in Linux and MacOS, this is done by writing in the terminal chmod +x <script>.py, for each script <script>.py),[3] and add the folder path (<folder>) to the list of execution files path. In Linux and MacOS,[4] this is done adding it to the system variable $PATH in the file .bashrc or .zprofile, located at the user's folder as a hidden file (in MacOS, press Shift + Command + . for making hidden files visible). Once this file is open, we just have to add the following line:

```
PATH="<folder>:${PATH}"
```

Then, next time the terminal is open, the scripts in the folder <folder> will be run just by calling them with its name, at any path.

3. Configuration files

The configuration files needed for these pipeline are plain text documents with the YAML formatting style. This way, we can specify several options and parameters by defining different

2. In Windows, if you are using a Python version not installed through the Microsoft Store, replace python3 with py. Also, in this case you must run the scripts with the argument --using_windows_py.
3. In Windows, instead of this you have to make sure that all the files ended in .py are opened with Python by default.
4. In Windows 10, to do this go to the Windows taskbar, right-click in the Windows icon, and select System; in the Settings window, under Related Settings, click Advanced system settings; then, on the Advanced tab, click Environment Variables, and edit the system variable Path, adding the path of the scripts folder. You will need to restart the terminal to apply the changes.

variables that can be nested. For the sake of simplicity, you can use one configuration file for the different working modes.

In order to define the values of the variables, we have to write, in a single line, the name of the variable followed by a colon (:), and its value, separated with a space. For the logical variables, the possible values are `yes/no`. There are variables whose value can be a list of elements, in which case they can be written between brackets and separated by commas or in single lines preceded by a hyphen (–). Similarly, in the case of nested variables, which work like Python dictionaries, next-level variables must be written in a different line and with an indent. Text variables can be written between simple quotation marks ('), but this is optional. If the definition of a variable is not written,[5] its default value will be used.

4. Working modes

This pipeline has several working modes, each of one requiring different variables to be defined in the corresponding configuration files.

4.1. Selection mode

This mode allows us to explore our data with CLASS, combine the desired spectra and save the output in different CLASS files, as well as in plain text files and in FITS format. To run this working mode, you should write in the terminal:[6]

```
classpipeline.py <config-path> --selection
```

,

where `<config-path>` is the path of the configuration file. For example, if we are located in the same folder as the configuration file, and its name is `config.yaml`, the command line should be:

```
classpipeline.py config.yaml --selection
```

.

Input files

As an input we need files in a CLASS format (for example, `.30m` or `.40m`). This kind of files can include lots of spectra, each of them with information stored in different variables associated. Some of them, which can be visualized in CLASS with the commands `find /all` and `list`, are:

- **Observation number (`N`)**. Number that identifies each spectrum in the file.
- **Source (`Source`)**. Name that identifies the source of the spectrum.
- **Line (`Line`)**. Name used for identifying the region (frequency range) of the spectrum. Usually, there are four different spectra for the same source and line, corresponding to different values of the variable 'telescope'.

5. Here, 'not written' means that neither the variable name nor its value are written in the configuration file. If the variable name is written (and the colon) but its value is not, the corresponding Python variable will get the value `None`.

6. Assuming that `classpipeline.py` has execution permissions and that is located in a path of executables files.

- **Telescope receiver (`Telescope`)**. Name used for identifying the receiver of the telescope used for the taking the observation, so is usually related to the frequency region around the zone defined by the source and the line. It can also include information of the polarization of the observation.
- **Scan number (`Sca`)**. Number that identifies a group of observations with the same source and line.

Apart from these variables, there are other several ones. For example, the Doppler correction, which is a number used to correctly calibrate the frequency of the spectra, and that can be obtained by typing `modify doppler` (it will not be modified if you only write that).

Output files

Depending on the specified parameters in the configuration file, a number of CLASS files will be generated in the specified output folder, with the spectra grouped as written in the configuration file, besides a CLASS file that contains all the spectra for each source. Moreover, each individual spectrum analyzed will be exported in plain text (`.dat`) and FITS format (`.fits`) in the specified folder.

Additionally, there is one tex file file in YAML formatting style that will be created:

- **`doppler_corrections.yaml`**. This file contains, for each spectrum analyzed, the Doppler correction variable, which is a number used to correctly calibrate the frequency of the spectrum. This is needed when importing a spectrum in CLASS in FITS format (`.fits`).

Configuration file variables

Below is a list of the variables of the configuration file that correspond to this working mode:

- **`data folder`**. The folder which contains the CLASS files.
- **`output folder`**. The folder in which the output CLASS files will be saved.
- **`exporting folder`**. The folder in which the output plain text files (`.dat`) and FITS files (`.fits`) of the spectra will be saved.
- **`input files`**. Nested variable (dictionary) which contains all the CLASS files that will be explored, along with the parameters of the spectra that will be merged. Each next-level variable should be the name of the CLASS file, including its extension (for example, `.30m`). Then, in the next level, the possible variables are:
  - **`sources-lines-telescopes`**. This variable contains the names of the `source`, `line` and `telescope` tags of the data that will be analyzed.[7] For each value of `source`, in the next level, there can be one variable for each of the `line` tags that will be explored. Lastly, for each value of `line`, there must be a list containing the `telescope` tags to be selected in CLASS; alternatively, its value can be instead `'common'`, which means that the script will use the `telescope` tags specified in the variable `common telescopes`. Different output files will be created for different values of `source` and `line`.

7. The names for the values of the tags `source`, `line` and `telescope` can have asterisks (*) to indicate 'any text' at the beginning (until the asterisk), at the end (from the asterisk), between some text (between two asterisks), or any text for the whole name (only an asterisk).

- **note**. Text variable used to name the resulting files for this input file. This is only used if the variable combine all is set to no. As a recommendation, the value of this variable can be the date of the observations for this input file.
- **bad scans**. List with the scan numbers that will be ignored for this input file. This variable is optional.

As a note, you can make the script ignore a certain observation file by preceding its name by two hyphens (--). This is useful in case that an observation file has no valid data within the specified parameters (sources-lines-telescopes and bad scans), in which case CLASS will fail and crash. This feature will not work if the variable use only daily bad scans is set to yes.

- **combine all**. Logical variable that determines if the data from different input files (that is, from different observation dates) are merged into common output files (for each source and line) or instead are saved in a different file for each input date (apart from being in different files for each source and line). If its value is set to no, you must specify the variable note for all the input files, in the variable input files.
- **weighting mode**. Kind of average performed when combining spectra. Possible values are 'time' (weighted with the observation time) and 'equal' (with equal weights for each spectrum).
- **fold**. Logical variable that determines if the resulting spectra for the specified tags of source, line and telescope are folded in CLASS or not.
- **common telescopes**. List of the telescope tags that will be used when the value of the line variable, inside of the source, is set as 'common'.
- **observatory**. Name of the observatory in CLASS where the observations were taken.
- **check Doppler corrections**. Logical variable that determines if the scripts checks for the Doppler effect corrections to the frequency. This is not needed if the frequencies are already in rest frequency.
- **extra note**. Text that will be added to the name of the output files.
- **bad scans**. List with the scan numbers that will be ignored, for all the input files.
- **use only daily bad scans**. Logical variable that can be set to yes in case we want to ignore the bad scans specified in the variable bad scans, which refer to all the input files. This way, the script will only consider the bad scans specified inside the variable input files, that is, the bad scans that are specified only for one observation file.

Default variable values

Below are the default values of the variables of the selection mode, that is, the values that they will get if they are not declared in the configuration file.

```
data folder: './'
output folder: './output/'
exporting folder: './exported/'
input files: []
combine all: yes
weighting mode: 'time'
```

```
fold: no
common telescopes: ['*']
observatory: ''
check Doppler corrections: no
extra note: ''
bad scans: []
use only daily bad scans: no
```

Example case

Below you have an example of configuration file for this working mode.

```
data folder: './input'
input files:
    FTSOdp20200930.30m:
        sources-lines-telescopes:
            L1517B:
                L84*: 'common'
            L1517BOFF1:
                L84*: 'common'
                L94*: 'common'
    FTSOdp20201001.30m:
        sources-lines-telescopes:
            L1517B:
                L84*: 'common'
                L94*: 'common'
            L1517BOFF1:
                L84*: 'common'
                L94*: 'common'
    FTSOdp20201002.30m:
        sources-lines-telescopes:
            L1517B:
                L84*: 'common'
    FTSOdp20201003.30m:
        sources-lines-telescopes:
            L1517B:
                L94*: 'common'
            L1517BOFF1:
                L94*: 'common'
    FTSOdp20201004.30m:
        sources-lines-telescopes:
            L1517B:
                H2CO21:  ['*UI*', '*UO*']
                CH3OH21: 'common'
                CH3CN:   'common'
            L1517BOFF1:
                L94*:    'common'
                CH3OH21: 'common'
                CH3CN:   'common'
combine all: yes
common telescopes: ['*LI*', '*UI*', '*LO*', '*UO*']
```

As an input, we would need the five CLASS files mentioned in this configuration file, with the data according to the specified parameters in the variable sources-lines-telescopes. If we run the script classpipeline.py in the selection mode (--selection), with this configuration

file, it should take a few seconds to run,[8] and the following CLASS files should have been created in the folder output:

- L1517B-CH3CN.30m.
- L1517B-CH3OH21.30m.
- L1517B-H2CO21.30m.
- L1517B-L84.30m.
- L1517B-L94.30m.
- L1517BOFF1-CH3CN.30m.
- L1517BOFF1-CH3OH21.30m.
- L1517BOFF1-L84.30m.
- L1517BOFF1-L94.30m.
- L1517B-all.30m.
- L1517BOFF1-all.30m.

Here, the two last files, which end with -all, contain all the spectra for each source. Apart from these files, all the corresponding spectra should have been saved, in .dat and .fits, in the folder exported/. For example, here are the file names of the files corresponding to the source L1517B, and the line L84:

- L1517B-L84-LI.dat.
- L1517B-L84-LI.fits.
- L1517B-L84-LO.dat.
- L1517B-L84-LO.fits.
- L1517B-L84-UI.dat.
- L1517B-L84-UI.fits.
- L1517B-L84-UO.dat.
- L1517B-L84-UO.fits.

That is, there is a file in .dat and .fits for each value of telescope (LI, LO, UI, UO).

4.2. Line search mode

This working mode is required before the reduction is done (with the reduction mode). In this mode, the script classpipeline.py calls the script classlinesearch.py (or, alternatively, classslinesearch.jl), which searches for visible emission or absorption lines in the selected spectra and writes in a file the regions with lines (the line windows). In order to do so, the script does an iterative reduction of the spectra with the parameters specified in the configuration file.

To run this working mode, you should write in the terminal:[9]

```
classpipeline.py <config-path> --line_search
```

where <config-path> is the path of the configuration file. This way the script classpipeline.py will call the Python script classlinesearch.py. However, it can call instead the Julia script classlinesearch.jl, which is usually a bit faster. To do so, we just have to add the argument --

8. The first time CLASS is open, it takes about 15 seconds to be opened, but next times it opens very quickly.
9. Assuming that classpipeline.py has execution permissions and that is located in a path of executables files.

use_julia to the command line:

```
classpipeline.py <config-path> --line_search --use_julia
```

.

Input files

The input files for this working mode are determined by the files specified in the variable input files, and correspond to the spectrum files in .dat and .fits generated by the selection mode and saved in the exporting folder (by default: exported/).

Output files

After the run of the script, a plain text file in YAML formatting style will have been created in the exporting folder:

- **frequency_windows.yaml**. It contains, for each spectrum analyzed, a list with the frequencies of the line windows (inferior and superior limits). The units are the same as the input files (usually, MHz).

Configuration file variables

The variables of the configuration file for this mode are basically the same as in the selection mode (pages 3 and 4), plus this nested variable:

- **reduction**. This nested variable (dictionary) includes the parameters of the reduction (which is needed to do the line search) and some graphical options.
  - **show plots**. Logical variable that determines if the plots of the identified lines in each spectrum are shown while the script runs.
  - **save plots**. Logical variable that determines if the plots of the identified lines in each spectrum are saved, in a folder called plots, while the script runs.
  - **rolling sigma clip**. Logical variable that determines if a rolling sigma clip is used instead a normal sigma clip in the algorithm of the line search. This is useful when the noise of the spectra can vary considerably along the frequency range. However, it increases the execution time.
  - **reference width**. Approximately, the maximum width of the lines present in the spectra, in number of channels.
  - **smoothing factor**. The smoothing factor, in units of number of channels, that will be used to smooth the spectra in order to search for lines.[10]
  - **intensity threshold (rms)**. Intensity value, in units of the root mean squared (RMS) noise for each spectrum, which is used to identify the lines; that is, all the candidate regions with intensities greater than this threshold will be identified as lines.

Default variable values

Below are the default values of the specific variables of the selection mode, that is, the values that they will get if they are not declared in the configuration file.

10. Basically, the reduction consists in several rolling medians and rolling averages whose size is this smooth factor (the actual reduction is more complex.)

```
reduction:
    show plots: no
    save plots: no
    rolling sigma clip: no
    reference width: 14
    smoothing factor: 20
    intensity threshold (rms): 8
```

Example case

As an example of a configuration file for this working mode, we can use the same example file for the selection mode (pages 4 and 5), adding the following parameters:

```
reduction:
    show plots: no
    save plots: yes
    reference width: 10
    smooth factor: 40
    intensity threshold (rms): 8
```

Prior to running the line search mode, we should have run the selection mode, in order to obtain the corresponding spectra in CLASS format and in .fits and .dat. For this example, the script should take about 2 minutes to run. Several plots should have been created in the folder plots/, like figures 1 and 2. Moreover, for each spectrum in .dat and .fits, in the exporting folder (exported/ by default) a reduced spectrum, with the name ended in -r (-r.dat, -r.fits), will have been saved.

Additionally, the file frequency_windows.yaml will have been created. It will be used to do the final reduction in the reduction working mode. Below you can see some of the content of this file.

```
L1517B-CH3CN-LI:
- [96889.04510000096, 96889.77709999903]
- [97073.21790000096, 97073.94989999903]
- [97080.37095000096, 97081.10294999903]
- [97087.52400000097, 97088.25599999903]
- [97472.07980000097, 97473.10479999904]
- [97610.74645000097, 97611.47844999903]
- [97754.46665000096, 97755.19864999903]
- [97761.59530000096, 97762.32729999903]
- [97768.74830000097, 97769.48029999904]
- [97973.42815000097, 97974.16014999904]
- [97980.58115000096, 97981.31314999903]
- [97987.70985000096, 97988.44184999903]
- [97994.83845000096, 97996.32729999903]
- [98001.94270000096, 98002.67469999903]
- [98011.26850000097, 98012.95259999904]
- [98393.18770000097, 98393.91969999904]
- [98400.31630000097, 98401.04829999903]
- [98597.28160000096, 98598.01359999903]
L1517B-CH3CN-LO:
- [93709.06730000097, 93709.79929999904]
- [93840.06820000097, 93841.58149999904]
- [93862.60160000097, 93863.33359999904]
- [93869.70580000097, 93870.43779999904]
                [...]
```

4.3. Reduction mode

In this mode, the script classpipeline.py calls the script classreduction.py (or, alternatively, classreduction.jl) which reduces the selected spectra and saves the results in CLASS files. In order to do so, we have to provide a file containing frequency windows that will be ignored when fitting the baseline of the corresponding spectrum and then interpolated; usually, this is done with transition lines, or also with bad channels.

To run this working mode, you should write in the terminal:[11]

classpipeline.py <config-path> --reduction ,

where <config-path> is the path of the configuration file. This way the script classpipeline.py will call the Python script classreduction.py. However, it can call instead the Julia script classreduction.jl, which is usually a bit faster. To do so, we just have to add the argument --use_julia to the command line:

classpipeline.py <config-path> --reduction --use_julia .

Input files

The input spectra files for this working mode are determined by the files specified in the variable input files, and correspond to the spectrum files in .dat and .fits generated by the selection mode and saved in the exporting folder (by default: exported/). Additionally, there is one plain text file in .yaml that is needed:

- **doppler_corrections.yaml**. This file contains, for each spectrum analyzed, the Doppler correction variable, which is a number used to correctly calibrate the frequency of the spectrum. This file is created in the selection mode. This is needed when importing the reduced spectra in .fits to CLASS.

Output files

Firstly, for each spectrum in .dat and .fits, in the exporting folder a reduced spectrum, with the name ended in -r (-r.dat, -r.fits), will have been saved. Then, those spectra will be imported to CLASS, so that several CLASS file will be created (ended also in -r) depending on the value of the variable input files of the configuration file. Also, a file containing all the spectra for each source will be created. If you specified the telescope efficiencies, the spectra will be calibrated in main beam temperature. If you set save plots to yes, figures similar to figure 1 will have been saved in the plots folder. Lastly, five text files in .yaml will have been created in the exporting folder:

- **rms_noises.yaml**. This file contains, for each reduced spectrum, the value of the RMS noise (the root median squared intensity on the continuum), in mK. It is needed by
- **rms_regions.yaml**. It contains, for each reduced spectrum, the frequencies of a small region where the RMS noise is very similar to the full spectrum value, and also where the mean of the intensity is very near to zero.

11. Assuming that classpipeline.py has execution permissions and that is located in a path of executables files.

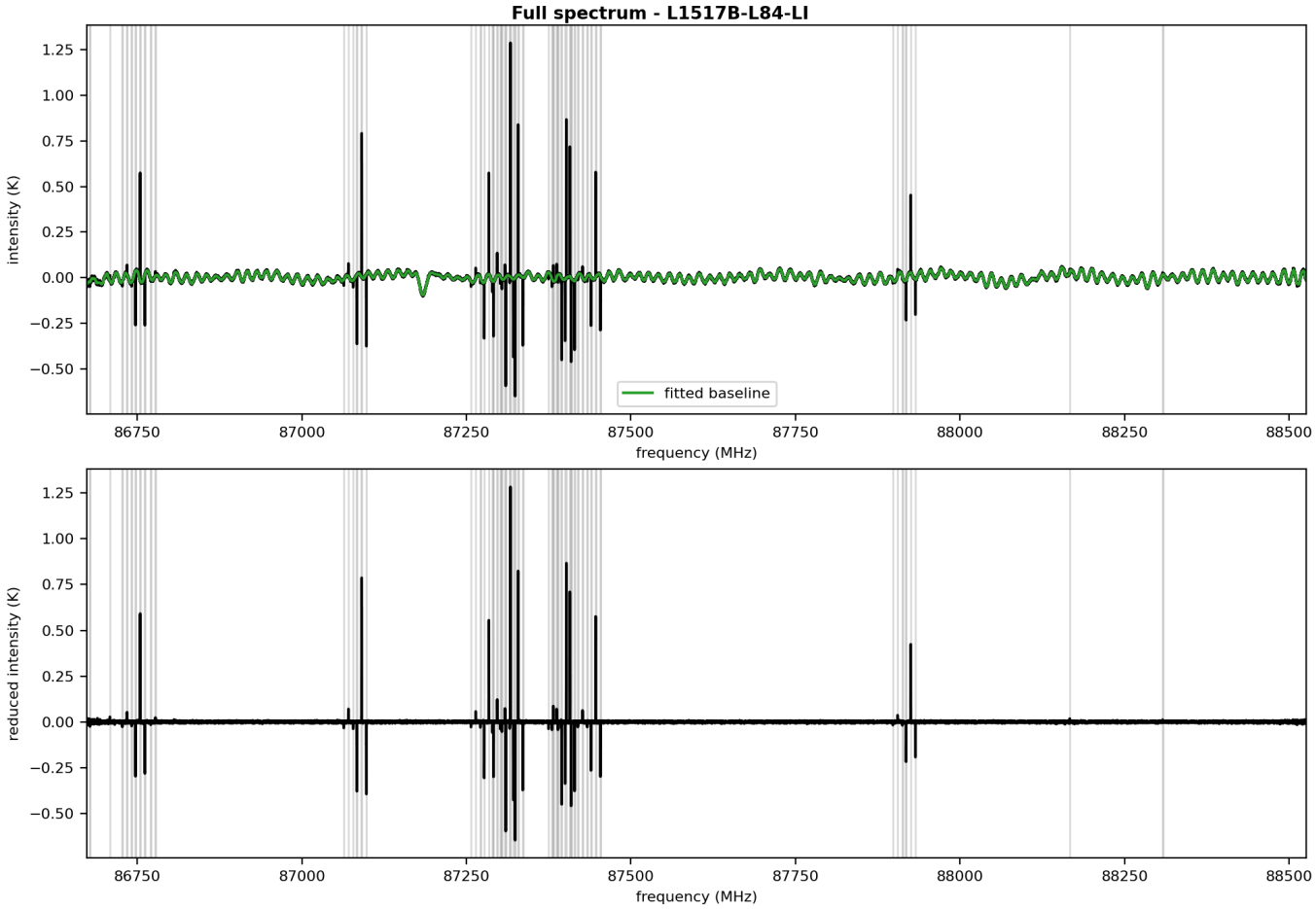

Figure 1. Plot of the spectrum of the files L1517B-L84-LI.dat and L1517B-L84-LI.fits, from the example case, with the detected lines marked in gray. *Upper*: Original spectrum and fitted baseline. *Lower*: Reduced spectrum.

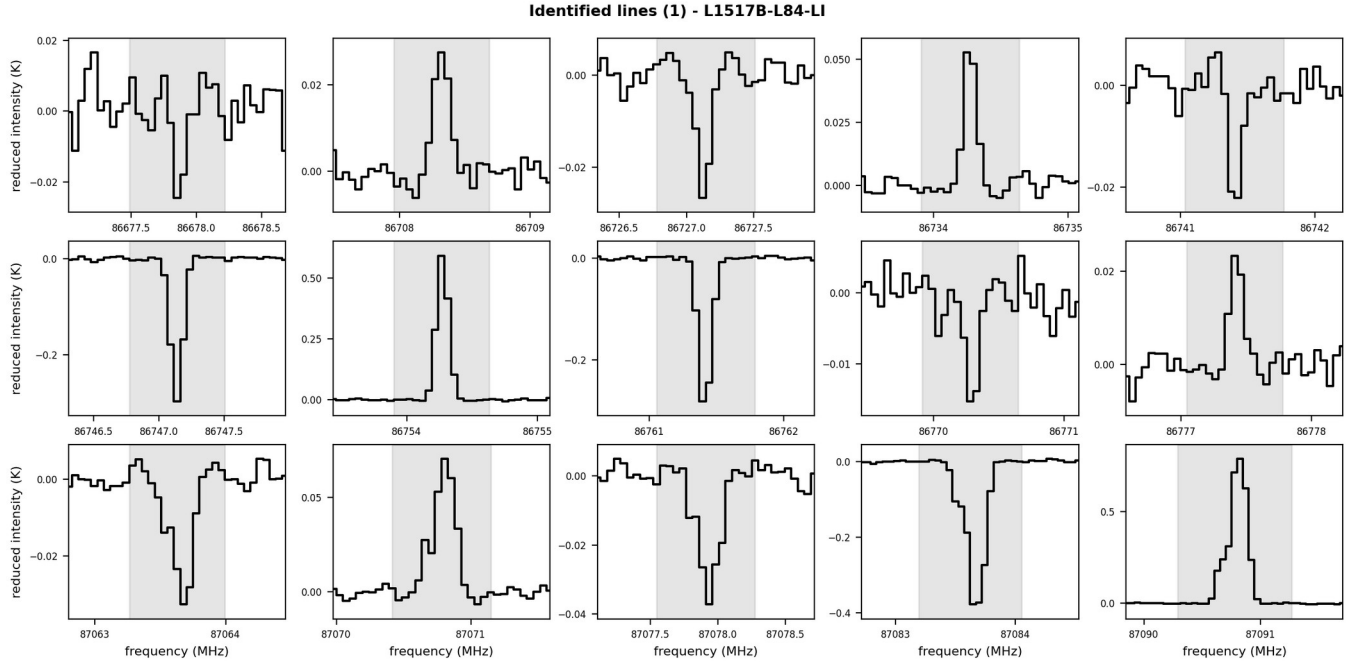

Figure 2. Plot of some of the identified lines for the files L1517B-L84-LI.dat and L1517B-L84-LI.fits, from the example case, with the line windows marked in gray. This is the first out of the four plots generated for these files; in total, 57 lines were identified.

- **reference_frequencies.yaml**. It contains, for each spectrum analyzed, the reference frequency (or rest frequency) used in the CLASS files and the files in .fits.
- **frequency_ranges.yaml**. It contains, for each spectrum analyzed, its minimum and maximum frequency.
- **frequency_resolutions.yaml**. It contains, for each spectrum analyzed, the resolution of each channel in frequency.

All the frequencies in these files are in the same units as the input files (usually, MHz).

Configuration file variables

The variables of the configuration file for this mode are basically the same as in the selection mode (pages 3 and 4) and the line search mode (page 7), plus one variable called telescope efficiencies, and another more inside the variable reduction (apart from the ones explained in page 7) They are described below:

- **reduction**. This nested variable (dictionary) includes the parameters of the reduction (which is needed to do the line search) and some graphical options. The possible next-level variables are the same as in the line search mode (see page 8), plus an additional one:
  - **relative frequency margin for the rms noise**. This determines a lower and upper margin, relative to the total frequency range of the spectrum, that will be ignored to compute the RMS noise.
- **telescope efficiencies**. This nested variable (dictionary) includes the main beam efficiency of the observing telescope for several frequencies. This is used to convert the intensity units of the spectra from antenna temperature to main beam temperature. It should have the following next level variables:
  - **frequency (GHz)**. List of frequencies, in GHz, in which the beam efficiency will be given.
  - **beam efficiency**. List of the main beam efficiencies (from 0 to 1) for the given frequencies, in the same order.

  Additionally, you can specify different beam efficiency relationships for different telescope receivers (telescope tag). To do so, before putting the two variables explained above (frequency (GHz) and beam efficiency), you have to write an entry with the name of the telescope whose beam efficiency you want to specify. You can also write part of the name between asterisks (*) and to specify all the telescope values which contain that text. In these cases, you can also specify a default beam efficiency relationship to be used for the telescope values not specified in the entries of the variable telescope efficiencies, using the name default for that entry.

The parameters of the reduction are stated in the variable reduction (see page 8).

Default variable values

Below are the default values of the variables of the selection mode, that is, the values that they will get if they are not declared in the configuration file.

```
reduction:
    show plots: no
    save plots: no
    rolling sigma clip: no
    reference width: 14
    smooth factor: 20
    intensity threshold (rms): 8
telescope efficiencies: {}
```

Example case

As an example of a configuration file for the reduction mode, we can use the same example file for the line search mode (page 7) adding the variable telescope efficiencies; that is, we can use the same file for the selection mode (pages 4 and 5) adding the variables reduction and telescope efficiencies. Hence, we should add the following variables to the example configuration file of the selection mode:

```
reduction:
    show plots: no
    save plots: yes
    reference width: 10
    smooth factor: 40
    intensity threshold (rms): 8
telescope efficiencies:
    beam efficiency:  [0.81, 0.78,  0.74, 0.63, 0.59, 0.49, 0.35, 0.34]
    frequency (GHz):  [86,   115,   145,  210,  230,  280,  340,  345]
```

Before running this working mode, we should have run the selection and line search modes, in order to obtain the needed files, that is, the spectra files and the file frequency_windows.yaml. The script should take about 2 minutes to run. For each spectrum in .dat and .fits, in the folder exported/, a reduced spectrum, with the name ended in -r (-r.dat, -r.fits), will have been saved. In the same folder, the four .yaml files listed previously, in the 'Output files' subsection, will have been created. In the folder plots/, several plots will have been saved, similar to figure 1. Lastly, and most important, the following CLASS files should have been created in the folder output/:

- L1517B-CH3CN-r.30m.
- L1517B-CH3OH21-r.30m.
- L1517B-H2CO21-r.30m.
- L1517B-L84-r.30m.
- L1517B-L94-r.30m.
- L1517BOFF1-CH3CN-r.30m.
- L1517BOFF1-CH3OH21-r.30m.
- L1517BOFF1-L84-r.30m.
- L1517BOFF1-L94-r.30m.
- L1517B-all-r.30m
- L1517BOFF1-all-r.30m

In the case that we wanted to specify one different beam efficiency relationship for one telescope value (for example, UO) we should write something as the following:

```
telescope efficiencies:
    default:
```

```
    beam efficiency:  [0.81, 0.78,  0.74, 0.63, 0.59, 0.49, 0.35, 0.34]
    frequency (GHz):  [86,   115,   145,  210,  230,  280,  340,  345]
'UO':
    beam efficiency:  [0.81, 0.78,  0.74, 0.63, 0.59, 0.49, 0.35, 0.34]
    frequency (GHz):  [86,   115,   145,  210,  230,  280,  340,  345]
```

Similarly, we could have specified different relationships for the telescope values containing the letter L and U, writing '*L*' and '*U*' instead of 'UO'. In this case, the entry default could be omitted, as all the telescope tags for this example contain either L or U in its name.

4.4. RMS noise check mode

This is a working mode that enables us to check for the values of the root mean squared (RMS) noise of the selected spectra in a specific frequency range, once they are reduced.

For the reduction, the script classreduction.py is used. Moreover, several images of the reduced spectra in the selected frequency range will be created, so that we can check their quality (for example, if there are ripples). Moreover, it creates a plot that shows both the RMS noise of each individual spectrum and the RMS noise of the cumulative spectrum that is generated by combining all the previous spectra.

To run this working mode, you should write in the terminal:[12]

classpipeline.py <config-path> --rms_check ,

where <config-path> is the path of the configuration file. Moreover, there is an additional action that can be done within this mode, which is the checking of the RMS noise plots. It consists of an interactive interface that shows the produced plots of the spectra of each observation, and allows us to identify bad scans writing it in the terminal. It is activated writing in the terminal:

classpipeline.py <config-path> --check_rms_plots .

For each plot, the following message will appear:

Options: Next (Enter), Previous (<), Select (x), Skip (s). .

Then, doing the actions written between parenthesis we can skip the current plot, go back to the previous one, mark the current one as a bad scan and go to the next one, or skip to the next group spectra, respectively.

Input files

The input files for this working mode are the CLASS files determined in the configuration file by the variable input files.

Output files

Depending on the specified parameters in the configuration file, a number of CLASS files will be generated in the specified output folder, with the spectra grouped as written in the config-

12. Assuming that classpipeline.py has execution permissions and that is located in a path of executables files.

uration file. Moreover, each individual spectrum analyzed will be exported in plain text (.dat) and FITS format (.fits) in the specified folder, with a name starting with rms-.

For each spectrum in .dat and .fits, in the exporting folder a reduced spectrum, with the name starting with rms- and ending in -r (-r.dat, -r.fits), will have been saved. Then, those spectra will be imported to CLASS, so that several CLASS files will be created (starting also in rms- and ending in -r) depending on the value of the variable input files of the configuration file.

Additionally, there are several text files in YAML formatting style that will be created (or modified, if it already existed). Two of them are the following:

- **doppler_corrections.yaml**. This file contains, for each spectrum analyzed, the Doppler correction variable, which is a number used to correctly calibrate the frequency of the spectrum. This is needed when importing a spectrum in CLASS in FITS format (.fits).
- **rms_noises.yaml**. This file contains, for each reduced spectrum, the value of the RMS noise (the root median squared intensity on the continuum), in mK.
- **bad_scans.yaml**. This file is only produced if we do the checking of the RMS noise plots (see page 15). It contains the variable bad scans, used for the selection mode, filled with the scans of the observations that were manually identified as bad scans.

Then, in the exporting folder, (by default, exported/) there will be one file for each set of parameters in CSV format (.csv), with a name starting with rms- and ending in -filenames, that contains information of the scans analyzed. Each of the rows of its data correspond to a scan number, and has the following four columns:

- **Row number**. Just the number of the row, starting in 1.
- **File number (file)**. Number which identifies the observation file which correspond to the scan of this row, with the same order as they are written in the variable input files.
- **Observation number (obs)**. Number that identifies each spectrum in CLASS.
- **Scan number (scan)**. The scan number of the observation.

There is also, for each set of parameters, a .yaml file which contains the values of the RMS noise for each scan group and for the cumulative spectra, starting with rms-. This data points are used to generate several plots, that will be saved in the plots folder (by default, plots/). The main plot file, starting with rms- and the name of the source, shows the evolution of the RMS noise with time, showing both the individual RMS noise of each scan group and the RMS noise of the cumulative spectrum. In case there are too much scan groups, additional plots will be generated with a reduced range of observations. Finally, another plot about the variation of the RMS noise starting with rms-metric will be created. It shows the variation of the RMS noise of the cumulative spectra and also this value weighted by the number of observations included in the cumulative spectra, for each time, over the total number of observations, so that we can observe the decrease in the RMS noise of the total spectra produced by each individual scan group.

Configuration file variables

The variables of the configuration file for this mode are basically the same as in the selection mode (pages 3 and 4), the line search mode (page 6) and the reduction mode (page 10), plus these variables:

- **rms noise check**. This nested variable (dictionary) can include several groups of parameters, so that the RMS check will be done with each of them; to specify them, we have to write first their name (for example, `parameters 1`, `parameters 2`...) and then we can write the following variables:
  - **scans per group**. This is the number of scans that will be grouped and averaged. In each of these groups, the RMS noise will be calculated.
  - **source-line-telescopes**. This is a nested variable similar to the variable `sources-lines-telescopes` of the variable input files (see page 4). It contains the names of the `source`, `line` and `telescope` tags of the data that will be analyzed.[13] Only one value of `source` is allowed, but in the next level there can be one variable for each of the `line` tags that will be explored. Lastly, for each value of `line`, there must be a list containing the `telescope` tags to be selected in CLASS. However, usually it is enough having only one value of `source`, `line` and `telescope`.
  - **frequency ranges (GHz)**. List containing the frequency ranges (which are also a list) where the reduction will be done and the RMS noise will be calculated.
- **only rms noise plots**. Logical variable that, if set to `yes`, prevents the script to do the calculations and instead plot the results from previous runs stored in the corresponding `.yaml` files (which start with `-rms` and continue with the source and the rest of parameters specified inside the variable `rms noise check`).

The parameters of the reduction are stated in the variable `reduction` (see page 8). Plus, in case you want to convert the intensity from antenna temperature to main beam temperature, remember to write the variable `telescope efficiencies` (see page 10).

Default variable values

Below are the default values of the variables of the selection mode, that is, the values that they will get if they are not declared in the configuration file.

```
rms noise check: []
only rms noise plots: no
```

And here are the default values of each of the parameters group of the variable `rms check`.

```
scans per group: 1
source-lines-telescopes: []
frequency ranges (GHz): []
```

Example case

As an example of a configuration file for the reduction mode, we can use the same example

13. The names for the values of the tags `source`, `line` and `telescope` can have asterisks (*) to indicate 'any text' at the beginning (until the asterisk), at the end (from the asterisk), between some text (between two asterisks), or any text for the whole name (only an asterisk).

file for the reduction mode (page 12) adding the variable rms check; that is, we can use the same file for the selection mode (pages 4 and 5) adding the variables reduction, telescope efficiencies, and rms check. However, we do not need to run any other working mode before running the RMS check mode. Hence, we should add the following variables to the example configuration file of the selection mode:

```
reduction:
    show plots: no
    save plots: yes
    reference width: 10
    smooth factor: 40
    intensity threshold (rms): 8
telescope efficiencies:
    beam efficiency:  [0.81, 0.78,  0.74, 0.63, 0.59, 0.49, 0.35, 0.34]
    frequency (GHz):  [86,   115,   145,  210,  230,  280,  340,  345]
rms noise check:
    parameters 1:
        scans per group: 1
        source-line-telescopes:
            L1517B:
                L94*: ['*LI*']
        frequency ranges (GHz):
        - [81.600, 81.650]
        - [82.430, 82.480]
        - [83.080, 83.130]
```

After three minutes, more or less, we would get the image shown in figure 3, saved in the folder ./plots, where we can see the evolution of the RMS noise with time for three frequency regions of the spectrum L1517B-L94-LI. This plot shows that the RMS noise is con-

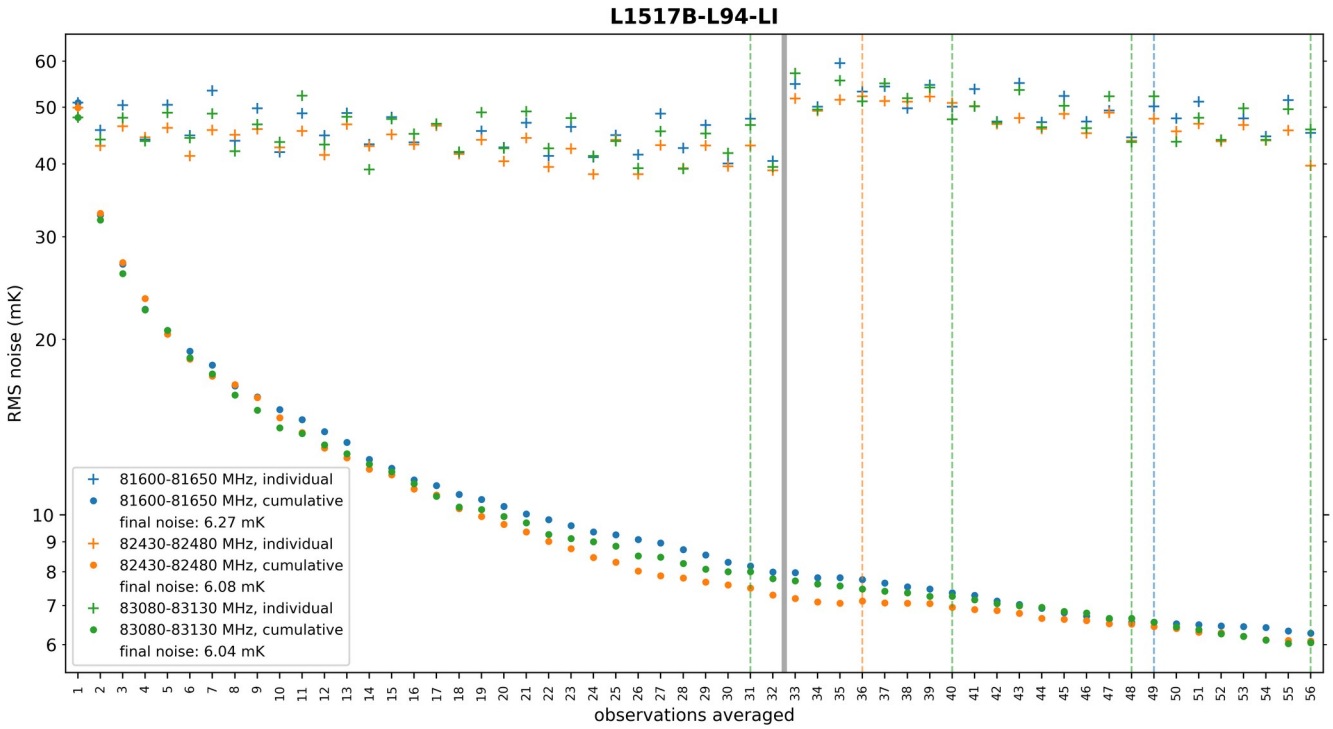

Figure 3. Evolution of the RMS noise with time for one of the observations of L1517B in three frequency regions. Crosses indicate the individual RMS noise of each observation, while circles represent the RMS noise of the cumulative spectrum. The wide gray line indicates the start of a new observation file (that is, a new day).

stantly decreasing over time, which is a good signal. There are just a few observations, marked with dashed lines, that increases the cumulative noise for a certain frequency region each time. They could be treated as bad scans, but it does not have to be so, as it could have happened by chance. Moreover, this behavior is not shared between the different frequency regions, which suggest the hypothesis that this is just happening by chance.

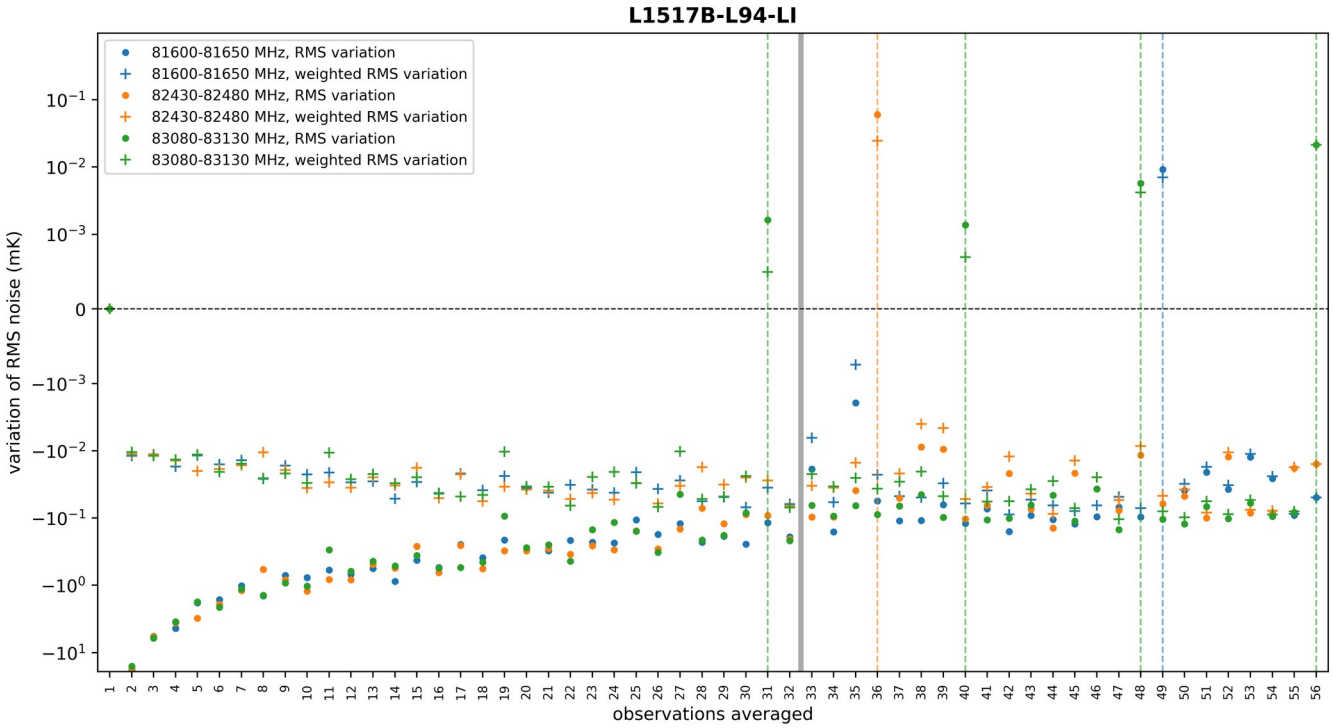

Figure 4. Evolution of the variation of the RMS noise for the cumulative observation of L1517B over time, in the region of 81.60 – 81.65 GHz. The dashed line indicates an observation that increased RMS noise.

These variations of the noise can be better seen in figure 4, where we can see explicitly the variation of the noise value over time. Note that the absolute variations start being relatively high, in absolute value, and they decrease with time. This is just because the variation produced by just one observation is less noticeable as we increase the number of observations of the cumulative spectrum. To account for this, we can see the weighted RMS noise variation, which indicates the decrease in the RMS noise of the cumulative spectrum produced by the addition of each individual observation.

To see more information about which one is each of the observations mentioned in the horizontal axis, we can see the file rms-L1517B-L94-LI-1-filenames.csv, located in the folder ./exported, which is reproduced below.

Table 1. Extract of the file rms-L1517B-L94-LI-1-filenames.csv, produced in the RMS noise check mode. It contains links the observation number shown in the plots (as figures 3 and 4) with the observation file (in the same order as they are written in the configuration file), the CLASS observation number, and the scan number.

	file	obs	scan
1	2	209	29
2	2	211	29
3	2	217	30

4	2	219	30
5	2	233	32
6	2	235	32
[...]	[...]	[...]	[...]

If we want to see the individual spectra in the selected region, we can go to the folder ./plots and see the corresponding plots, starting with spectrum-rms-. For example, figure 5 shows the cumulative spectrum of the 38 first observations in the specified frequency range. To see all the spectra in an easy way, we could run the checking of the RMS noise plots, with --check_rms_plots. In this case, for each group of spectra, determined by groups of parameters set in the variable check rms of the configuration file, the individual spectra will be shown first (that is, the averaged ones obtained with the number of spectra set in the variable scans per group), and then the cumulative spectra will be shown. Once we get to the cumulative ones, it does not make sense to mark any of them as bad observations (bad scans), so you can type s for skip to the individual spectra of the next group of observations, In that case, the file bad_scans.yaml would be created, with all the identified bad scans.

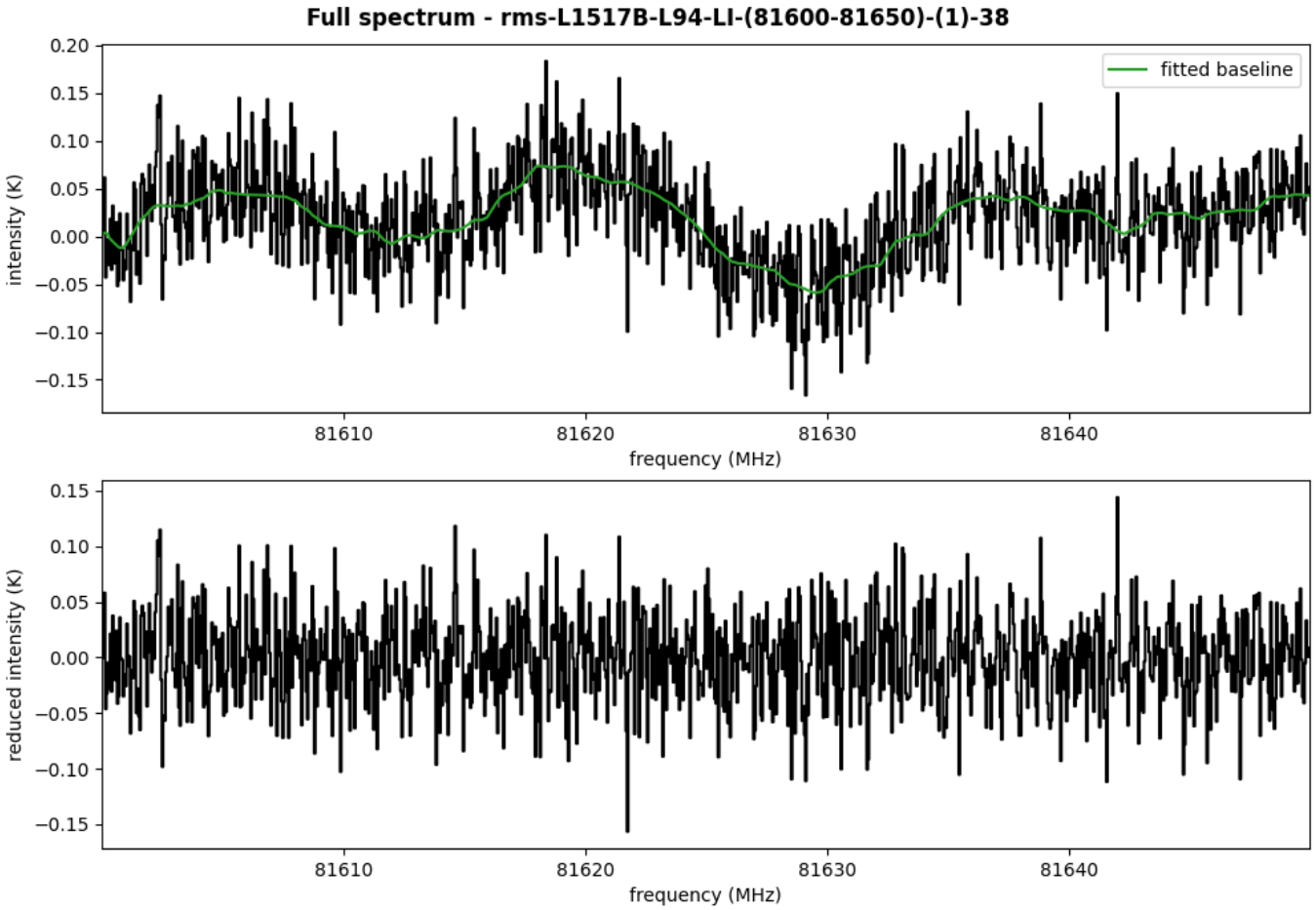

Figure 5. Cumulative spectrum of L1517B in the selected region where the RMS will be calculated. The top plot shows the original averaged spectrum with the fitted baseline, while the bottom one shows the reduced spectrum.

4.5. Averaging mode

This working mode can only be used if the variable combine all of the configuration file is set to yes. It enables us to combine and average the overlapping reduced spectra within CLASS, using the RMS noise of each spectrum for weighting them. It uses the script classaverage.py, writing automatically the needed configuration file from the one that we have used for the previous working modes.

To run it, you should write in the terminal:[14]

```
classpipeline.py <config-path> --average      ,
```

where <config-path> is the path of the configuration file. We would need the output of the reduction mode, and as a result, several CLASS files with the joint spectra will be created.

For the averaging of the overlapping regions of the spectra, each of them will be weighted by the inverse squared value of their RMS noise (ΔI), that is, the individual weights would be $w = 1 \,/\, \Delta I^{\,2}$. (Then, all weights are normalized, dividing them by the sum of all the weights.) With this weighting, the resulting RMS noise ($\Delta I_{\,\mathrm{tot}}$) should be the harmonic sum of the individual noises, that is: $\Delta I_{\,\mathrm{tot}} = 1 \,/\, \left(\sum_i 1 \,/\, \Delta I_i^{\,2} \right)^{½}$.

Input files

The input files for this working mode are mainly the CLASS files of the reduced spectra, produced in the reduction mode. In particular, we need the files that contains all the spectra for each source (ended in -all-r). Moreover, we need the following .yaml files:

- **rms_regions.yaml**. It contains, for each reduced spectrum, the frequencies of a small region where the RMS noise is very similar to the full spectrum value, and also where the mean of the intensity is very near to zero. These regions are used for calculating the RMS noise of each spectrum within CLASS.
- **frequency_ranges.yaml**. It contains, for each spectrum analyzed, its minimum and maximum frequency. They are used for determining the two frequency windows to ignore all the frequency range of each spectrum but the region determined in rms_regions.yaml
- **reference_frequencies.yaml**. It contains, for each spectrum analyzed, the reference frequency (or rest frequency) used in the CLASS files and the files in .fits. They are used for constructing the two frequency windows to ignore all the frequency range of each spectrum but the region determined in rms_regions.yaml, as the windows in CLASS must be written with respect to the reference frequency (rest frequency) of each spectrum.

All the frequencies in these files are in the same units as the input files (usually, MHz).

Output files

First of all, a configuration file called config-average-auto.yaml will be created, just before

14. Assuming that classpipeline.py has execution permissions and that is located in a path of executables files.

calling the script classaverage.py. Then, depending on the specified parameters in the configuration file, a number of CLASS files will be generated in the specified output folder, with the overlapping spectra combined and averaged within CLASS. Moreover, one plot covering all the spectra will be generated for each source specified in the configuration file, so that one can see the overlapping spectra. The name of this image will be the name of the source, ended in -all-r.

Configuration file variables

The needed variable of the configuration file for this mode are basically those of the selection mode (pages 3-4). Therefore, we can just use the same configuration file as before. However, it is important to note that this mode can only be used if the variable combine all is set to yes.

Example case

The same example configuration file as the one in the selection mode (pages 5-6) can be used for the averaging mode. Before running the averaging mode, we should have run the selection, line search, and reduction modes, in order to obtain all the needed files. After less than a minute of running, we will get the following CLASS files as an output:

- L1517B-all-r-a.30m.
- L1517BOFF1-all-r-a.30m.

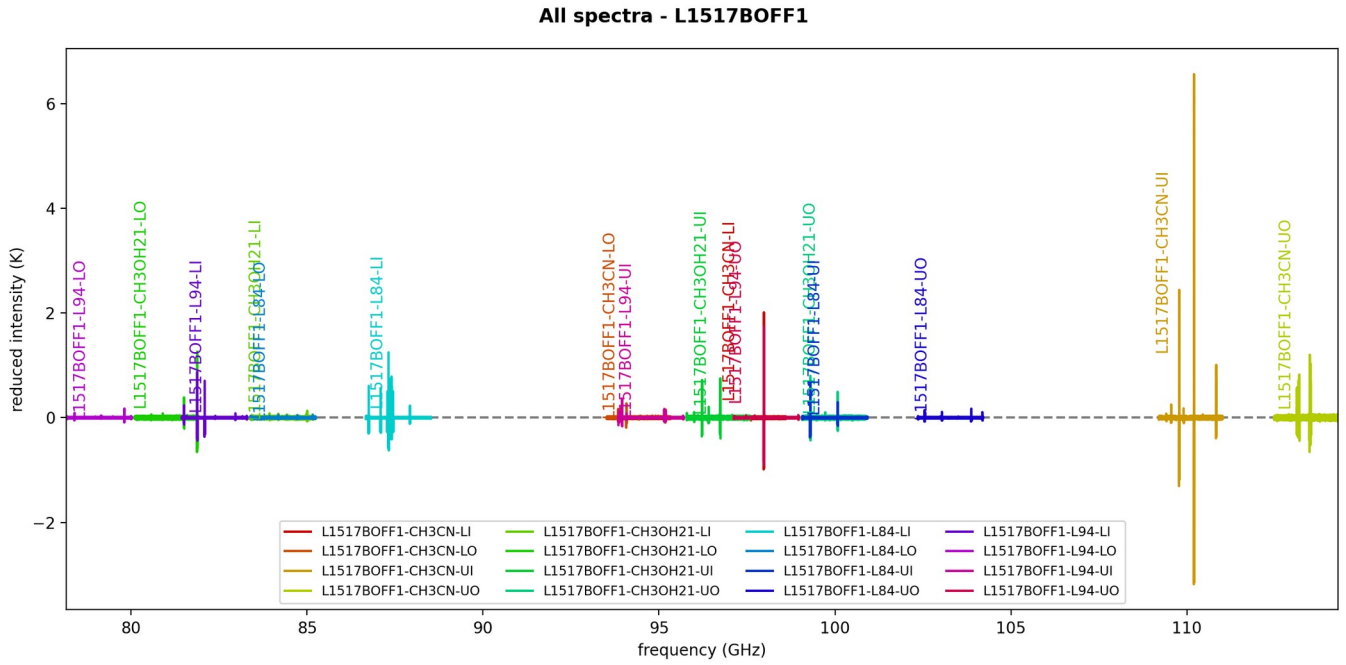

Figure 6. Plot of all the observations for the source L1517B in its methanol peak, with the names of the individual spectra.

Plus, figure 6 shows one of the created plots, where we can see all the spectra for one source.

4.6. Combine mode

This working mode should only be used with spectra which have roughly the same frequency

range, and also if the variable combine all of the configuration file is set to no.[15] It enables us to combine and average the spectra for the different observation days, using the RMS noise of each spectrum for weighting them. Moreover, it can compare the spectra to be averaged to search for ghost lines, that is, lines which are not consistent through all the spectra, and remove them. It uses the script classcombine.py, writing automatically the needed configuration file from the one that we have used for the previous working modes.

To run it, you should write in the terminal:[16]

```
classpipeline.py <config-path> --combine     ,
```

where <config-path> is the path of the configuration file. We would need the output of the reduction mode, and as a result, several CLASS files with the joint spectra will be created.

For the averaging of the overlapping regions of the spectra, each of them will be weighted by the inverse squared value of their RMS noise (ΔI), that is, the individual weights would be $w = 1 \,/\, \Delta I^{\,2}$. (Then, all weights are normalized, dividing them by the sum of all the weights.) With this weighting, the resulting RMS noise ($\Delta I_{\rm tot}$) should be the harmonic sum of the individual noises, that is: $\Delta I_{\rm tot} = 1 \,/\, \left(\sum_i 1 \,/\, \Delta I_i^{\,2} \right)^{½}$.

Input files

The input files for this working mode are mainly the CLASS files of the reduced spectra, produced in the reduction mode. Moreover, we need the following .yaml files:

- **doppler_corrections.yaml**. This file contains, for each spectrum analyzed, the Doppler correction variable, which is a number used to correctly calibrate the frequency of the spectrum. This is needed when importing a spectrum in CLASS in FITS format (.fits).
- **rms_noises.yaml**. This file contains, for each reduced spectrum, the value of the RMS noise (the root median squared intensity on the continuum), in mK.
- **frequency_windows.yaml**. It contains, for each spectrum analyzed, a list with the frequencies of the line windows (inferior and superior limits). The units are the same as the input files (usually, MHz).

Output files

First of all, a configuration file called config-combine-auto.yaml will be created, just before calling the script classcombine.py. Then, through the running of the code, the created spectra will be saved in CLASS format as well as in .fits and .fits, with the file names ending in -c. Also, the YAML files rms_noises.yaml and frequency_windows.yaml will be updated with the values of the averaged spectra. Lastly, several plots showing all the averages made will be generated, so that one can see the resulting spectra and the identified ghost lines, marked with gray lines.

Configuration file variables

15. It also works with combine all set to yes, producing a similar output than the averaging mode, but it has not been fully tested.

16. Assuming that classpipeline.py has execution permissions and that is located in a path of executables files.

The variables of the configuration file for this mode are basically the same as in the selection mode (pages 3-4) and the line search mode (page 7), plus this variable:

- **ghost lines**. This nested variable (dictionary) includes the parameters needed to perform the checking of the ghost lines. It contains the following variables:
  - **clean lines**. Logical variable that determines if the search for ghost lines is done.
  - **absolute intensity threshold (rms)**. Intensity value that will be used for identifying ghost lines, in units of the maximum RMS noise of the spectra to be averaged. If a channel of any of the spectra has a difference with any other spectra greater than this value, it will be identified as a ghost line channel.
  - **relative intensity threshold**. Intensity value that will be used for identifying ghost lines. If a channel of any of the spectra to be averaged has a difference with any of the other spectra greater than this value times the median of that channel values, all with absolute values, it will be identified as a ghost line channel.
  - **smoothing factor**. Number that defines the size of the smoothing that will be applied to the interpolation of the regions with ghost lines. Firstly, a median filter with this value as size will be applied, and then a median filter with a size equal to this value divided by 3.

Default variable values

Below are the default values of the variables of the selection mode, that is, the values that they will get if they are not declared in the configuration file.

```
ghost lines:
    clean lines: no
    absolute intensity threshold (rms): 10
    relative intensity threshold: 0.3
    smoothing factor: 40
```

Example case

This time, we will use as an example observations of a different source than the previous examples, G0693, as we need to have similar frequency ranges for all the spectra and that the value of the variable combine all is set to no. Below you have an excerpt of the used configuration file.

```
data folder: 'input/'
input files:
    21A014_084_20210325.40m:
        note: '21-03-25'
        sources-lines-telescopes:
            G0693:
                L42300: 'common'
    21A014_085_20210326.40m:
        note: '21-03-26'
        sources-lines-telescopes:
            G0693:
                L41400: 'common'
    21A014_086_20210327.40m:
        note: '21-03-27'
        sources-lines-telescopes:
```

```
            G0693:
                L42300: 'common'
    21A014_087_20210328.40m:
        note: '21-03-28'
        sources-lines-telescopes:
            G0693:
                L41400: 'common'
    [...]
observatory: 'Yebes40m'
common telescopes: ['*S5H*', '*S5V*']
extra note: 'S5'
combine all: no
bad scans:
  '*S5H*': ['10479:10494', '10945:10960', '10961:10973', '10961:10973',
            '11190:11205', '25619:25634', '26709:26724', [...] '64756:64771']
  '*S5V*': ['10479:10494', '10479:10494', '10961:10973', '11190:11205',
            '12398:12410', '12411:12426', '25619:25634', [...] '64802:10477']
reduction:
    reference width: 200
    intensity threshold (rms): 6
    smooth factor: 800
    show plots: no
    save plots: yes
    rolling sigma clip: no
ghost lines:
    clean lines: yes
    absolute intensity threshold (rms): 10
    relative intensity threshold: 0.3
    smoothing factor: 40
```

Prior to running the combine mode, we should have run the selection, line search and reduction modes, in order to obtain all the needed files. Then, after around 2 minutes of running,

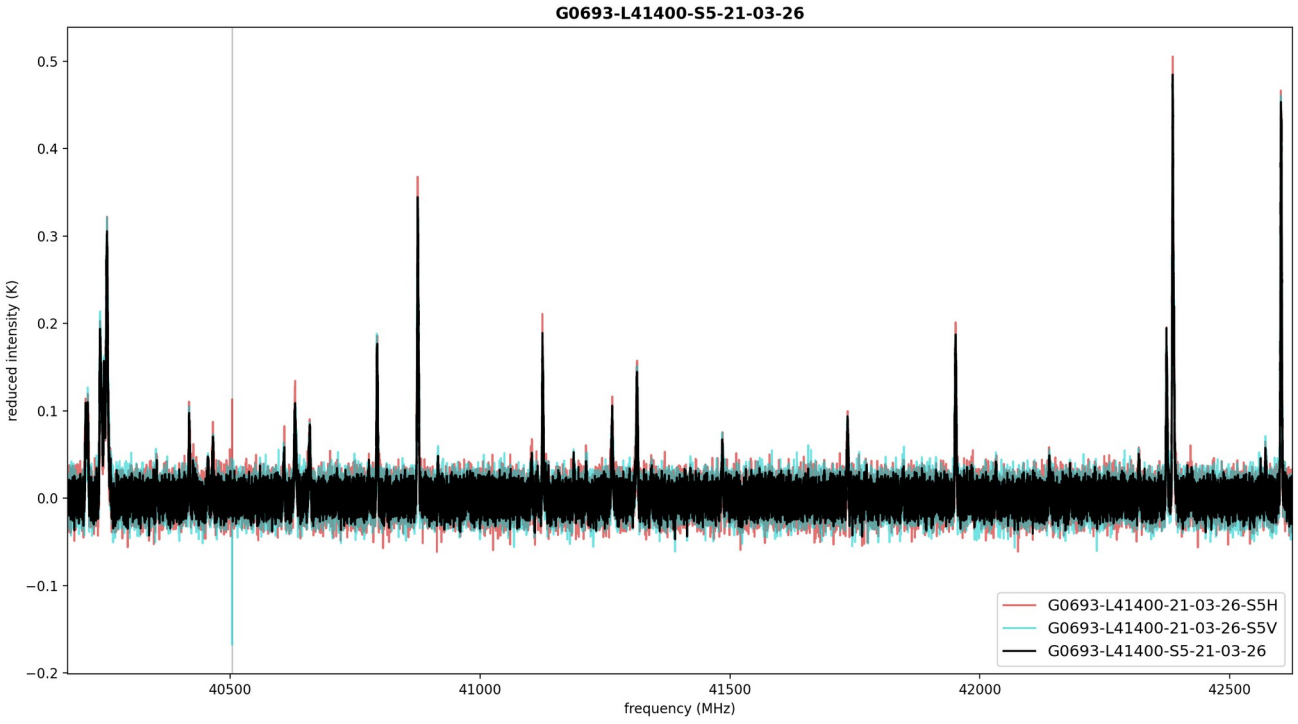

Figure 7. Plot showing one of the performed averages for the spectra of one day for each polarization, for the frequency setup L41400. The gray line indicates the detected ghost line.

we would get as a final output the following CLASS files:

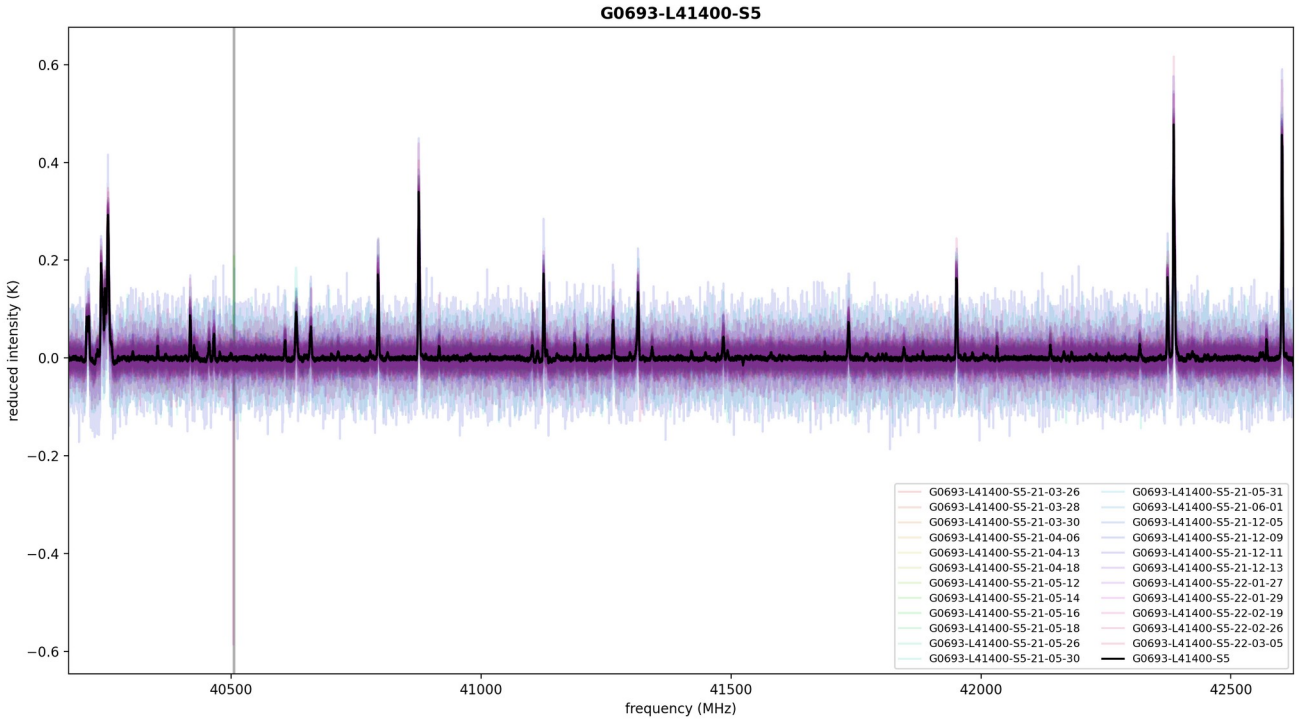

Figure 8. Plot showing one of the performed averages of the spectra of all the observation days, for the frequency setup L41400. The gray line indicates the detected ghost line.

- `G0693-L41400-S5-r-c.30m`.
- `G0693-L42300-S5-r-c.30m`.

Moreover, figures 7 and 8 show two of the created plots, where we can see the average between polarizations and between observation days.

4.7. Spectra table mode

This is a simple working mode that creates a table in `.csv` format which contains some information about each of the individual spectra that we have created in the previous modes, as stated in the previous configuration files: the frequency range, the frequency resolution, and the RMS noise.

A single table will be created for each of the observed sources. Plus, if there are overlapping spectra and we set the variable `combine all` to `yes` in the configuration file, another table will be generated for each source, taking into account those overlapping regions, and calculating the resulting RMS noise in those cases.[17]

To run this working mode, you should write in the terminal:[18]

```
classpipeline.py <config-path> --spectra_table     ,
```

17. In theory, and assuming gaussian noise, the resulting RMS noise ($\Delta I_{\rm tot}$) would be the harmonic sum of all the individual RMS noises (ΔI_i), that is: $\Delta I_{\rm tot} = 1\ /\ (\ \sum_i\ 1\ /\ \Delta I_i^{\,2}\ )^{½}$.

18. Assuming that `classpipeline.py` has execution permissions and that is located in a path of executables files.

Shortly after, the corresponding tables will be created in the folder stated in the variable exporting folder, of the configuration file.

The resulting tables are particularly useful for using the SLIM Table Generator (slimtables.py), in order to obtain tables in LaTeX format of the transition lines and the abundances of the observed species.

Input files

The input files for this working mode are only three of the .yaml files produced in the reduction mode:

- **frequency_ranges.yaml**. It contains, for each spectrum analyzed, its minimum and maximum frequency.
- **rms_noises.yaml**. This file contains, for each reduced spectrum, the value of the RMS noise (the root median squared intensity on the continuum), in mK.
- **frequency_resolutions.yaml**. It contains, for each spectrum analyzed, the resolution of each channel in frequency.

All the frequencies in these files are in the same units as the input files (usually, MHz).

Output files

For each of the observed sources, a file in .csv format will be created, starting with the name of the source and ending with -table. Plus, if there are overlapping regions, one more table will be created for each one of the previous ones, with the same name but ending in -table-joint instead. Each table will have the following columns:

- **Name (spectrum).** The name of the spectrum. In case of multiple overlapping spectra, all the names separated by a sum sign (+).
- **Number of channels (channels).** This is useful for looking at first glance the size of each spectrum, specially if there are overlapping spectra.
- **Minimum frequency (min. frequency (MHz)).** Minimum frequency of the spectrum, in MHz.
- **Maximum frequency (max. frequency (MHz)).** Maximum frequency of the spectrum, in MHz.
- **RMS noise (rms noise (mK)).** RMS noise of the spectrum, in mK.
- **Frequency resolution (resolution (MHz)).** Frequency resolution of the spectrum, in MHz.

Configuration file variables

The only needed variable of the configuration file for this mode is input files, used also in all the previous working modes (see pages 4-5). Therefore, we can just use the same configuration file as before.

Example case

The same example configuration file as the one in the selection mode (pages 5-6) can be used

for the averaging mode. Before running this mode, we should have run the selection, line search and reduction modes, in order to obtain all the needed files. Then, as an output, we would get four tables:

- `L1517B-table.csv`
- `L1517B-table-joint.csv`
- `L1517BOFF1-table.csv`
- `L1517BOFF1-table-joint.csv`

Below, in Tables 2 and 3, you can see the beginning of the two first mentioned `.csv` files.

Table 2. Table of the file `L1517B-table.csv`, generated in the spectra table mode for the individual observations of the source L1517B.

spectrum	channels	min. frequency (MHz)	max. frequency (MHz)	rms noise (mK)	resolution (MHz)
L1517B-L94-LO	37947	78164.2624	80017.0699	5.236642577923483	0.0488261915743
L1517B-CH3OH21-LO	37124	80105.2567	81917.9785	12.159479734109485	0.048828125
L1517B-L94-LI	37947	81444.1325	83296.94	4.986720609961181	0.0488261915743
L1517B-CH3OH21-LI	37124	83385.1279	85197.8498	11.125980025835494	0.048828125
L1517B-L84-LO	37946	83393.6424	85246.4001	3.6463362570786786	0.0488261654973
[...]		[...]	[...]	[...]	[...]

Table 3. Table of the file `L1517B-table-joint.csv`, generated in the spectra table mode for the joint observations of the source L1517B, that is, taking into account the overlapping spectra.

spectrum	channels	min. frequency (MHz)	max. frequency (MHz)	rms noise (mK)	resolution (MHz)
L1517B-L94-LO	37947	78164.2624	80017.0699	5.236642577923483	0.0488261915743
L1517B-CH3OH21-LO	27420	80105.2567	81444.1325	12.159479734109487	0.048828125
L1517B-CH3OH21-LO + L1517B-L94-LI	9704	81444.1325	81917.9785	4.613795070107724	0.048828125
L1517B-L94-LI	28242	81917.9785	83296.94	4.986720609961181	0.0488261915743
L1517B-CH3OH21-LI	174	83385.1279	83393.6424	11.125980025835494	0.048828125
L1517B-CH3OH21-LI + L1517B-L84-LO	36950	83393.6424	85197.8498	3.464997099759476	0.048828125
[...]		[...]	[...]	[...]	[...]

Citation of the code

If you use this pipeline for your work, you can put the link to the GitHub repository:

https://github.com/andresmegias/gildas-class-python/ .

Moreover, if you can cite the paper in the following link, as the data were reduced using this pipeline.

https://ui.adsabs.harvard.edu/abs/2023MNRAS.519.1601M/abstract/ .

Useful links

The following links may be of interest:

- **GILDAS.**
  https://www.iram.fr/IRAMFR/GILDAS/
- **Python.**
  https://www.python.org/
- **PyYAML.**
  https://pyyaml.org/wiki/PyYAMLDocumentation/
- **NumPy.**
  https://numpy.org/
- **SciPy.**
  https://scipy.org/
- **Pandas.**
  https://pandas.pydata.org/
- **Astropy.**
  https://www.astropy.org/
- **Matplotlib.**
  https://matplotlib.org/
- **Julia.**
  https://julialang.org/
- **MADCUBA.**
  https://cab.inta-csic.es/madcuba/

Credits

This software has been developed at the Centre for Astrobiology (*Centro de Astrobiología*, CAB), in Madrid (Spain), within the group of Chemical Complexity in the Interstellar Medium and Star Formation (Department of Astrophysics).

Coding and testing

Andrés Megías Toledano

Supervising

Izaskun Jiménez Serra
Víctor M. Rivilla Rodríguez
Jesús Martín-Pintado Martín

Testing

Antonio Martínez Henares
Álvaro López Gallifa
Laura Colzi
Sarah Massalkhi
Marina Centenera Merino
Samantha Scibelli

License

The Automated GILDAS-CLASS Pipeline (classpipeline.py, classlinesearch.py, classreduction.py, classaverage.py, classcombine.py, classlinesearch.jl, and classreduction.jl) is published under a GNU General Public License (GPL) version 3.