TRends Analysis Guided Interfaces COllection: TRAGICO

1 Introduction

This paper provides a comprehensive guide on the application of routines included in the `TRAGICO` package.

`TRAGICO` (TRends Analysis Guided Interfaces COllection) is a collection of functions for the extraction and analysis of experimental parameters from 1D and pseudo-2D NMR spectra acquired on Bruker instruments, designed to streamline the process of identifying trends in NMR data and facilitate various analytical tasks.

Key features include:

- Versatility: These functions are highly adaptable and can be integrated with external tools to accommodate a wide range of analytical needs.
- Customizability: Easily modify the functions to suit specific analysis requirements, ensuring flexibility and precision.
- User-Friendly: Clear examples and step-by-step instructions guide users through the application of the functions, making the process accessible to users of all levels.

The primary application of these functions is to analyze intensity trends of modeled signals or regions within experimental NMR data. By default, the functions employ an exponential decay model to estimate longitudinal relaxation rates.

Beyond the default application, this tutorial explores several other analytical techniques that can be achieved using these functions. These examples demonstrate the versatility and power of the tools provided.

By the end of this section, you will be able to:

- Understand the purpose and capabilities of the included functions.
- Apply the functions to analyze intensity trends in NMR data.
- Customize the functions for specific analytical tasks.
- Explore additional analytical techniques supported by these tools.

Additionally, this small package can be considered an example application of KLASSEZ functions (*vide infra*).

In this guide, the sections indicated as:

```
# example code
```

represent code listings, while those reported as:

```
$ python main.py
Output
```

represent code printouts in the execution terminal. The content of input and output files will be presented as:

```
Output file ...
```

2 Requirements

The code folder contains three python scripts: `main4test.py`, which contains the example codes used in this tutorial, `f_fit.py`, containing all the functions which can be called in the main code, and `f_functions.py`, a collection of all-purpose functions used in by the analysis tools. Additionally, all the data obtained from the examples are saved in the same folder.

In order to run the codes you need to install `python` (version 3.12). The additional dependencies and their versions are: `numpy` (version 2.0.2)[1], `matplotlib` (version 3.9.2)[2], `lmfit` (version 1.3.2)[3] and `nmrglue` (version 0.10)[4]. However, the program could work also for different versions of the packages. The one that could be the most labile is the `matplotlib` version, due to the heavy use of graphical interfaces.

The functions have been tested on Ubuntu 22.04 LTS inside a dedicated Anaconda environment.

One way to set up the proper environment to run the codes is through the use of miniconda, available at https://docs.anaconda.com/miniconda/. Miniconda is a free, lightweight version of Anaconda that installs only Python and the conda package manager. It's a good option if you only need Python and don't need all the scientific packages that come with Anaconda. Once installed, it can be activated using:

```
$ conda activate
```

the confirmation of the correct activation is confirmed by the presence of `(base)` in the terminal line:

```
(base) $
```

All the packages can be in principle installed with the command:

```
(base) $ conda install <package_name>
```

The `nmrglue` library could be not available on the default channel, therefore one could try with the `conda-forge` or `bioconda` ones (complete explanation at: https://docs.conda.io/projects/conda/en/latest/user-guide/concepts/channels.html). The complete guide to environment managing and package installation is given at https://docs.conda.io/projects/conda/en/latest/user-guide/

index.html.

Some functions in the `f_functions.py` file are directly taken from the package for monodimensional and multidimensional NMR data processing and visualization: KLASSEZ (available at https://github.com/MetallerTM/klassez). In fact, this tutorial serves as a practical example of how to integrate routines from KLASSEZ within an external package, demonstrating its versatility and potential for broader applications.

Warning: Since these are simply python scripts, and NOT an actual installable package, they need always to be present in the execution folder!

3 Analysis of intensity trends via integration

3.1 Single 1D spectrum or series of 1D spectra

In this example, the objective is to extract the intensities from a series of monodimensional spectra acquired at different time steps for the monitoring of a reaction. The main in this case is composed as follows:

```
1
2  # saved in main4test.py
3
4  from f_fit import *
5
6  path = "path/to/spectra"
7
8  # automatic definition of the list of experiment folder inside path.
9  # the name of the spectra must conduct to the processed data.
10 # the followign two lines could be substituted with a list of strings, e.g.:
       list_sp = ['1/pdata/1', '2/pdata/1', ...]
11 num_sp = list(np.arange(101,128,1))
12 list_sp = [str(i)+'/pdata/1' for i in num_sp]
13
14 # this is the list of delays characterizing the experiments in list_sp
15 # can be just list of progressive numbers with no meaning
16 delays = np.linspace(0, 1, len(list_sp))
17
18 # intensity_fit_1D is the function for the estimation of the intensity of the
       peaks in the 1D spectra via integration of a defined spectral region.
19 intensity_fit_1D(
20                  path,
21                  delays,
22                  list_sp,
23                  area=True,
24                  cal_lim = (-72.52,-72.94),
25                  baseline=True,
26                  delta = 10
27                  )
```

The variable `path` is the path to the spectra folder, while `list_sp` is the list of experiment names down to the processed data (inside the `"pdata"` folder). If the processed-data folder is not specified, by default the path `"/pdata/1"` will be used for each experiment. `delays` is a list of variables, containing the same number of elements as the `list_sp` list, which in this case represents the time steps of the spectra acquisition.

The function for intensity collection and plot is called with `intensity_fit_1D()`. Besides already specified variables, the additional flags have the following meaning:

- `area = True or False` defines the way of estimation of the peaks intensity. If `True` the integral of the peak is computed using the trapezoid rule implemented in `numpy`, whereas if `False` the intensity is estimated as the maximum point in the peak region.
- `cal_lim = (<min>,<max>) or None` defines the calibration region enclosed between `<min>` ppm and `<max>` ppm. If `None` is given, no calibration is performed. The calibration method consists of the estimation of a shift value which is applied to the experimental spectra to minimize the difference between the first spectral calibration region and the regions in the subsequent spectra.
- `baseline = True or False` defines whether or not the baseline correction, defined as a polynomial function of fourth degree, is performed (*vide infra*).
- `delta = <value> or 0.0` sets the augmentation of `<value>` in ppm of the region for the baseline correction for each peak (*vide infra*).

Once the code is started, a folder named `<spectra folder>_integral/` is created, where the result files will be saved.

```
(base) $ python main4test.py
Writing directory <spectra folder>_integral
Performing calibration...
...done
```

In case `cal_lim` is not `None`, the calibration is performed and the first part of the output file, named `<spectra folder>.out`, is saved inside the result folder, is generated:

```
SPECTRA PATH:
path/to/spectra

CALIBRATION: (-72.52000:-72.94000) ppm
in points
0   -1   0   -4   -4   -5   -2   -2   -5   -5   -6   -3   -5   -7   -7   -4   -6
      -7   -5   -4   -7   -8   -8   -6   -7   -6   -7
in ppm
0.00000   -0.00307   0.00000   -0.01228   -0.01228   -0.01535   -0.00614
    -0.00614   -0.01535   -0.01535   -0.01842   -0.00921   -0.01535   -0.02149
     -0.02149   -0.01228   -0.01842   -0.02149   -0.01535   -0.01228   -0.02149
      -0.02457   -0.02457   -0.01842   -0.02149   -0.01842   -0.02149

Points:
1 0.000
2 0.038
3 0.077
4 0.115
5 0.154
6 0.192
7 0.231
8 0.269
9 0.308
10  0.346
11  0.385
12  0.423
13  0.462
```

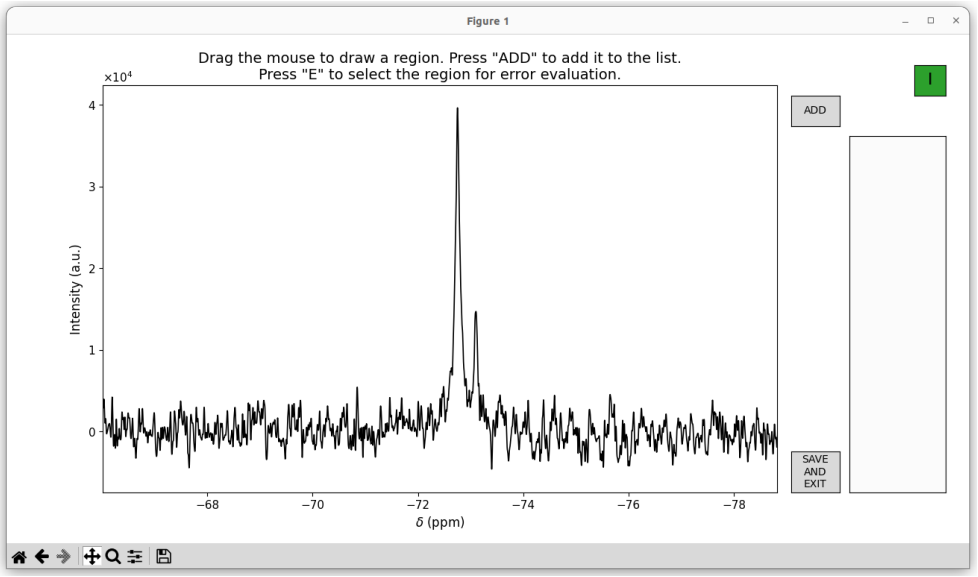

Figure 1: First graphical interface for the selection of the integration regions.

```
14   0.500
15   0.538
16   0.577
17   0.615
18   0.654
19   0.692
20   0.731
21   0.769
22   0.808
23   0.846
24   0.885
25   0.923
26   0.962
27   1.000
```

where the path to the spectra and calibration parameters are saved. The list of parameters, preceded by spectra indices, is also reported.

At this point, the interface for the selection of the peak regions opens (see figures 1-3):

1. Drag the mouse to select the region of the spectra (in red) and then press `ADD` to save the region (in green). The list of saved regions will appear on the left. The image can be zoomed using the button 🔍 or using the mouse wheel to zoom the spectra. When dragging the rectangle for the regions selection make sure that none of the widgets of the `matplotlib` panel is active.

2. Once all the peak regions are selected, press the key "E" on the keyboard to change from the mode for peaks selection I to the mode for error estimation E, then select a region with no peaks and wider than the peaks region and press `ADD`. Only the last region selected in this mode will be considered for the error estimation.

3. Lastly, press `SAVE AND EXIT` to save everything and exit.

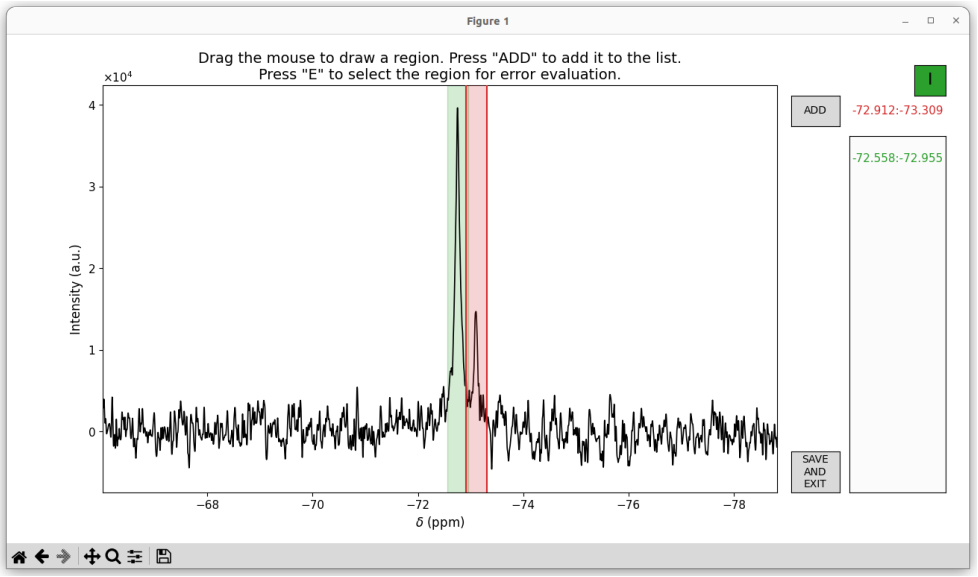

Figure 2: First graphical interface for the selection of the integration regions at step 1.

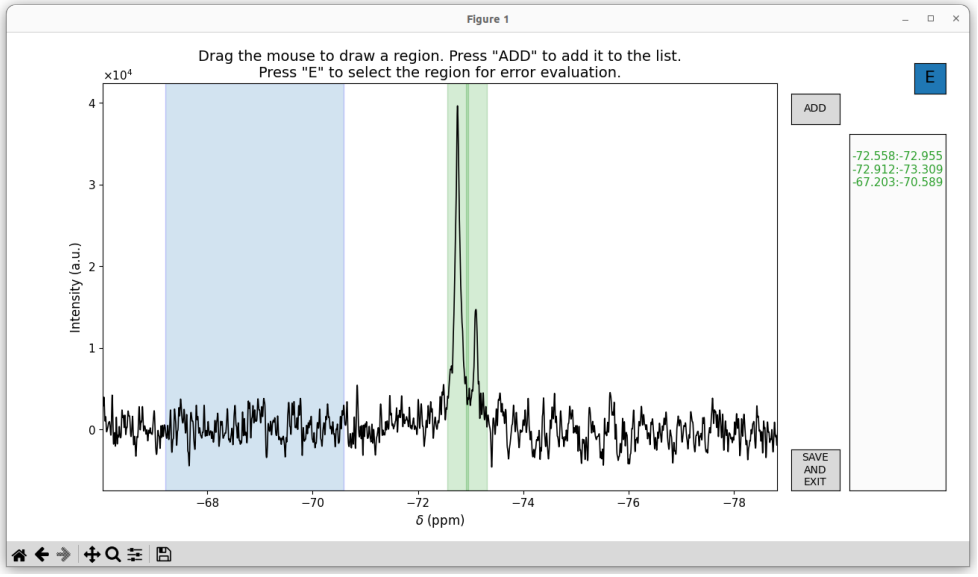

Figure 3: First graphical interface for the selection of the integration regions at step 2.

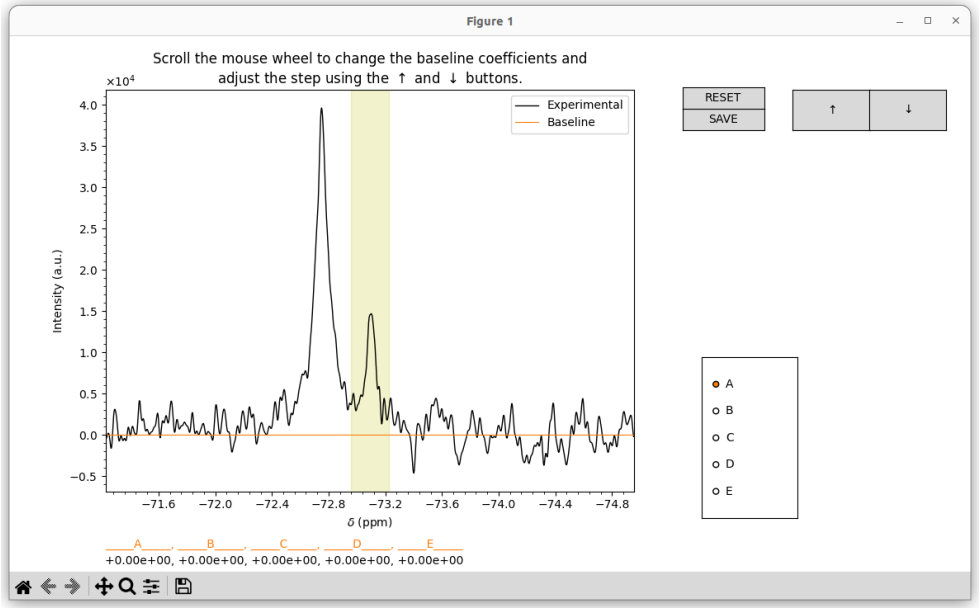

Figure 4: Second graphical interface for the definition of the baseline coefficients.

If the flag `baseline` is set to `True` a second interface, one for each selected region, will open (see figure 4):

1. The panel opens over one of the previously defined regions, enlarged depending on the `delta` value specified in the main. The actual integration region is highlighted in yellow. To adjust the baseline coefficients, move the mouse wheel. The parameters variation step can be increased or decreased using the buttons ↑ and ↓ respectively.

2. Once the coefficients are satisfactory, press `SAVE` and close the window.

At this point, if the flag `area` is set to `True` the integral of each peak region is computed, after baseline subtraction. The same spectral regions and the same baseline coefficients are used throughout the series of spectra. The integration error is estimated based on the selected error region as:

$$\varepsilon = \overline{(S(n) - B(n))}N \tag{1}$$

where $\overline{(S(n) - B(n))}$ is the average value of the difference between the real-valued spectral region S and the baseline B, defined for the points n in the selected region and N is the number of points in the peak region. Hence, each peak region has a different error value.

The integrals and corresponding errors, together with the baseline coefficients, are saved in the `<spectra folder>.out` file:

```
Selected intervals (ppm):
1 -72.9740   -73.2620
2 -72.5580   -72.9100

================================================================================

N. interval: 1
Coefficients
```

```
A 0.00000e+00
B 0.00000e+00
C 0.00000e+00
D 0.00000e+00
E 0.00000e+00
N. point  Integral
0 165985.546 +/- 121902.903
1 345173.870 +/- 121296.881
2 317284.956 +/- 116528.034
3 213759.163 +/- 125611.570
4 341583.156 +/- 119606.831
5 244432.883 +/- 123446.407
6 462923.259 +/- 123916.245
7 359819.559 +/- 122942.927
8 496746.308 +/- 128721.257
9 369831.767 +/- 125325.200
10  307188.018 +/- 128180.112
11  383870.162 +/- 123464.404
12  547830.707 +/- 128261.690
13  360459.458 +/- 130269.551
14  453306.566 +/- 119857.537
15  364775.979 +/- 120367.580
16  446547.264 +/- 122656.430
17  409706.469 +/- 124273.685
18  517661.461 +/- 115579.361
19  473147.565 +/- 121448.076
20  418503.219 +/- 118947.072
21  446486.903 +/- 122518.851
22  411248.529 +/- 116832.207
23  448733.743 +/- 115934.702
24  508997.587 +/- 119867.107
25  458755.576 +/- 119941.677
26  553345.718 +/- 115752.788

N. interval: 2
Coefficients
A 0.00000e+00
B 0.00000e+00
C 0.00000e+00
D 0.00000e+00
E 0.00000e+00
N. point  Integral
0 552970.749 +/- 150740.148
1 807934.473 +/- 149990.767
2 1031435.615 +/- 144093.806
3 1153124.162 +/- 155326.134
4 1262848.945 +/- 147900.920
5 1291019.505 +/- 152648.783
6 1489867.487 +/- 153229.765
7 1450096.096 +/- 152026.200
8 1486294.875 +/- 159171.446
9 1574112.449 +/- 154972.021
10  1629921.066 +/- 158502.289
11  1579737.123 +/- 152671.037
12  1667395.005 +/- 158603.165
13  1429108.013 +/- 161086.004
14  1551804.589 +/- 148210.933
```

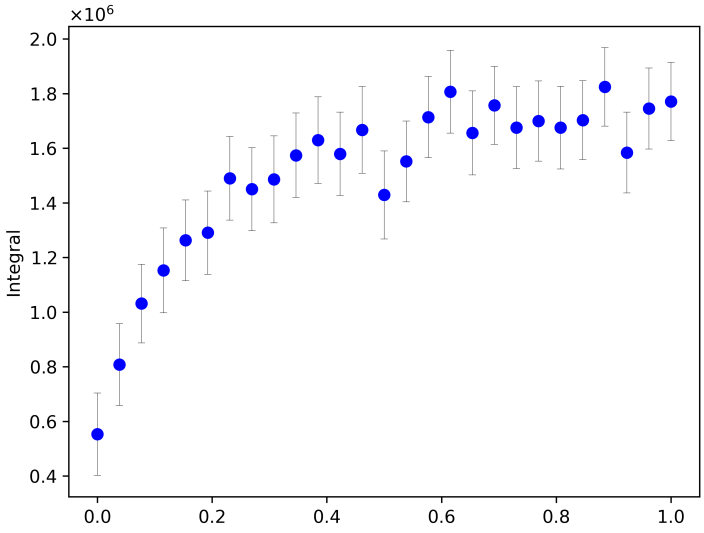

Figure 5: Integrals values are plotted against the `delays` list, with the corresponding error bars.

```
15   1713897.457 +/- 148841.632
16   1806804.722 +/- 151671.930
17   1656301.272 +/- 153671.761
18   1757448.220 +/- 142920.715
19   1675773.427 +/- 150177.729
20   1699829.333 +/- 147085.089
21   1675404.019 +/- 151501.805
22   1703131.560 +/- 144469.934
23   1824859.505 +/- 143360.115
24   1584325.292 +/- 148222.766
25   1745509.078 +/- 148314.976
26   1771370.330 +/- 143135.168
```

The integrals, errors and delays are saved in separate files named: `y_<n>.txt`, `Err_<n>.txt` and `x_<n>.txt` respectively, where `<n>` is the identification number of the interval. Lastly, plots of `y_<n>.txt` versus `x_<n>.txt`, with error bars from `Err_<n>.txt`, are saved as `Interval_<n>.png` (see figure 5).

If the flag `area` is set to `False` the highest intensity value in the peak region is used. The result folder will be named as `<spectra folder>_intensity/` and the error will be estimated as the standard deviation in the error region, therefore, the same error is used for all the peak regions. For comparison, the plot for the same region but estimated using the intensity "version" of the function is reported in figure 6.

In order to use the program for a single monodimensional spectra just define two single-element lists as `list_sp` and `delays`.

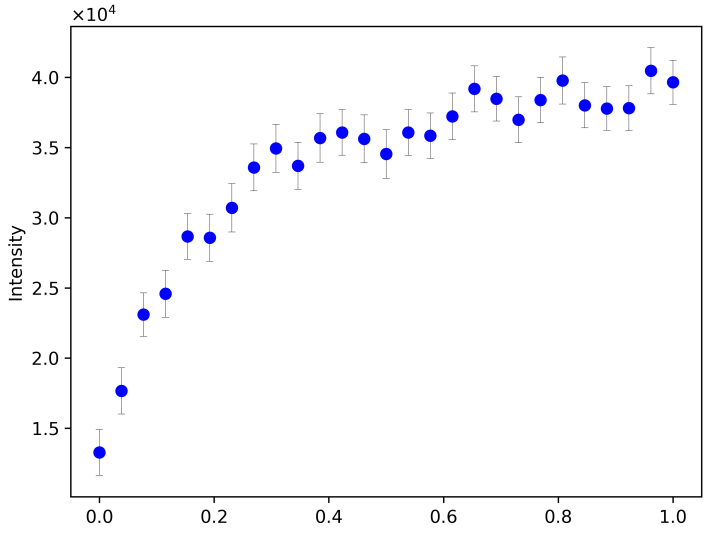

Figure 6: Intensity values are plotted against the `delays` list, with the corresponding error bars.

3.2 Single pseudo-2D spectrum or series of pseudo-2D spectra

In this part of the tutorial, the program is applied for the analysis of a series of pseudo-2D experiments acquired for the study of longitudinal relaxation rates (R_1) at different fields. Each transient of the single pseudo-2D corresponds to a certain t_1 (named `delay` in the code) and each pseudo-2D corresponds to a particular "relaxation-field" (named `VCLIST` in the code).

```
1
2  # saved in main4test.py
3
4  from f_fit import *
5
6  path = "path/to/spectra"
7
8  list_sp = [int(f) for f in os.listdir(path) if not f.startswith('.')]
9  list_sp = [str(f) for f in np.sort(list_sp)]
10
11 #list of delays values for each experiment
12 delays_list = [np.loadtxt(path+"/"+list_sp[i]+"/vdlist")+0.003 for i in range(
       len(list_sp))]
13
14 intensity_fit_pseudo2D(
15                      path,
16                      delays_list,
17                      list_sp,
18                      prev_lims = True,
19                      prev_coeff = True,
20                      area=True,
21                      VCLIST=None,
22                      cal_lim = (1.2,0.888),
23                      baseline=True,
24                      doexp=True,
25                      )
```

The variables defined in subsection 3.1 maintain the same meaning also in this case, with only a few differences:

- `delays_list` is not a single list, but is a list of `delays`, one for each experiment (since we are dealing with pseudo-2D and not with monodimensional spectra).
- `prev_lims = True or False` defines whether the integration regions used for all the pseudo-2Ds are the same as those selected for the first spectrum.
- `prev_coeff = True or False` defines whether the baseline coefficients used for all the pseudo-2Ds are the same as those selected for the first spectrum.
- `VCLIST = <list> or None` represent the identification parameter for the series of pseudo-2D. If `None` is passed, no `VCLIST.txt` is generated at the end of the analysis (*vide infra*).
- `doexp = True or False` activates the fit of the extracted integrals (or intensities) using the default exponential function (*vide infra*).

The types of interfaces generated when running this script are the same as those of the previous subsection. The main difference in the result folder is that, inside the principal one, a folder is generated for each pseudo-2D, named as the experiments. In each of these folders, the same files and plots as those saved for the monodimensional case are stored: `y_<n>.txt`, `Err_<n>.txt`, `x_<n>.txt` and `<spectra folder>_sp<name of experiment>.out`. The information collected in the latter are the following:

```
SPECTRA PATH:
path/to/spectra

CALIBRATION: (1.20000:0.88800) ppm
in points
0   0   -1   0   0   0   0   0   0   1   1   1   1   2   4   10
in ppm
0.00000   0.00000   -0.00037   0.00000   0.00000   0.00000   0.00000   0.00000
     0.00000   0.00037   0.00037   0.00037   0.00037   0.00073   0.00147
    0.00367

Points:
1 0.004
2 0.013
3 0.053
4 0.083
5 0.103
6 0.153
7 0.253
8 0.353
9 0.453
10  0.553
11  0.703
12  0.903
13  1.203
14  2.003
15  4.003
16  5.003

VCLIST point: 14092.38 T
```

```
Selected intervals (ppm):
1 5.4200  5.3460
2 5.5580  5.4250

================================================================================

N. interval: 1
Baseline coefficients
A 0.00000e+00
B 0.00000e+00
C 0.00000e+00
D 0.00000e+00
E 0.00000e+00
N. point  Integral
0 10883151.711 +/- 247847.095
1 10842182.793 +/- 221316.031
2 10453531.484 +/- 240856.177
3 10121184.129 +/- 244094.444
4 9861177.082 +/- 246413.179
5 9399847.906 +/- 245201.390
6 8521047.199 +/- 211876.305
7 7678852.941 +/- 191353.523
8 6851356.035 +/- 178254.787
9 6122744.348 +/- 171963.327
10  5184510.312 +/- 140945.652
11  4093542.816 +/- 116844.732
12  2869853.590 +/- 78003.623
13  1045959.492 +/- 36695.294
14  80827.332 +/- 4354.397
15  15026.227 +/- 7535.681

N. interval: 2
Baseline coefficients
A 0.00000e+00
B 0.00000e+00
C 0.00000e+00
D 0.00000e+00
E 0.00000e+00
N. point  Integral
0 64161591.539 +/- 444161.626
1 63026166.023 +/- 396615.858
2 61119717.199 +/- 431633.346
3 59565170.301 +/- 437436.578
4 58689047.344 +/- 441591.934
5 56127886.422 +/- 439420.312
6 51364765.262 +/- 379699.121
7 46921859.371 +/- 342920.670
8 42890940.488 +/- 319446.698
9 39151038.102 +/- 308171.904
10  33974751.348 +/- 252585.772
11  28107159.000 +/- 209395.014
12  21027525.922 +/- 139788.671
13  9598216.477 +/- 65760.873
14  1280816.441 +/- 7803.425
15  326027.543 +/- 13504.537
```

```
------------------------------
N. PEAK: 1
------------------------------

Fit Parameters:

Mono
y = a + A exp(-t/T1)
fit: T1=-2.6979e-02, a=-1.259e+05, A=1.118e+07
[[Fit Statistics]]
    # fitting method   = leastsq
    # function evals   = 9
    # data points      = 16
    # variables        = 1
    chi-square         = 1.0485e+11
    reduced chi-square = 6.9898e+09
    Akaike info crit   = 363.650761
    Bayesian info crit = 364.423350
[[Variables]]
    t1: -0.02697852 +/- 0.00655241 (24.29%) (init = 0)

Bi
y = a + A (f exp(-t/T1a)+ (1-f) exp(-t/T1b))
fit: f=5.006e-01, T1a=-2.6979e-02, T1b=-2.6976e-02, a=-1.259e+05, A=1.118e+07
[[Fit Statistics]]
    # fitting method   = leastsq
    # function evals   = 38
    # data points      = 16
    # variables        = 3
    chi-square         = 1.0485e+11
    reduced chi-square = 8.0651e+09
    Akaike info crit   = 367.650761
    Bayesian info crit = 369.968527
[[Variables]]
    f:    0.50064330 +/- 22059.5395 (4406238.81%) (init = 0.5)
    t1a: -0.02697872 +/- 4446.27987 (16480689.99%) (init = 0)
    t1b: -0.02697646 +/- 4457.74644 (16524578.30%) (init = 0)
[[Correlations]] (unreported correlations are < 0.100)
    C(t1a, t1b) = -1.0000

------------------------------
N. PEAK: 2
------------------------------

Fit Parameters:

Mono
y = a + A exp(-t/T1)
fit: T1=4.6071e-02, a=-6.235e+05, A=6.496e+07
[[Fit Statistics]]
    # fitting method   = leastsq
    # function evals   = 11
    # data points      = 16
    # variables        = 1
    chi-square         = 1.1449e+12
    reduced chi-square = 7.6328e+10
```

```
    Akaike info crit   = 401.900298
    Bayesian info crit = 402.672887
[[Variables]]
    t1:  0.04607109 +/- 0.00397011 (8.62%) (init = 0)

Bi
y = a + A (f exp(-t/T1a)+ (1-f) exp(-t/T1b))
fit: f=5.051e-01, T1a=4.6071e-02, T1b=4.6071e-02, a=-6.235e+05, A=6.496e+07
[[Fit Statistics]]
    # fitting method   = leastsq
    # function evals   = 47
    # data points      = 16
    # variables        = 3
    chi-square         = 1.1449e+12
    reduced chi-square = 8.8071e+10
    Akaike info crit   = 405.900298
    Bayesian info crit = 408.218065
[[Variables]]
    f:    0.50512397 +/- 283655.021 (56155526.24%) (init = 0.5)
    t1a:  0.04607102 +/- 75170.7217 (163162711.93%) (init = 0)
    t1b:  0.04607099 +/- 76727.3588 (166541600.17%) (init = 0)
[[Correlations]] (unreported correlations are < 0.100)
    C(t1a, t1b) = -1.0000
    C(f, t1a)   = -0.2663
    C(f, t1b)   = +0.2663
================================================================================
```

The last section of the output collects the result relative to the fit of the integral (or intensity) trends, for each spectral region, numbered accordingly (but called `N. PEAK` since this type of fit assumes that each region contains a single peak). The exponential fit is performed to estimate the longitudinal relaxation rates, fitting the $\log_{10}$ of $T_1 = 1/R_1$, either in a monoexponential or a biexponential fashion. The fit reports, automatically generated by the `lmfit` library, are also saved (more information on the content of this kind of report can be found at https://lmfit.github.io/lmfit-py/fitting.html).

In this example, the absurdly high values of uncertainties for the parameters of the biexponential fit are due to the high correlations among parameters, meaning that the monoexponential model is enough to reproduce the experimental trend.

Additionally, the T_1 values, with errors, are saved in `t1.txt` files in each experiment folder:

```
n.peak  T1 (s)  err (s) f
1 9.3977e-01  2.2825e-01
1 9.3977e-01  1.5488e+05  9.3977e-01  1.5529e+05  0.5006
2 1.1119e+00  9.5818e-02
2 1.1119e+00  1.8142e+06  1.1119e+00  1.8518e+06  0.5051
```

where the elements, for each peak region, are

```
n T1  error
n T1a error1  T1b error2  f
```

and plots, of the type in figure 7, are generated for each interval.

Lastly, a function for the global analysis of the extracted R_1 at different fields can be used

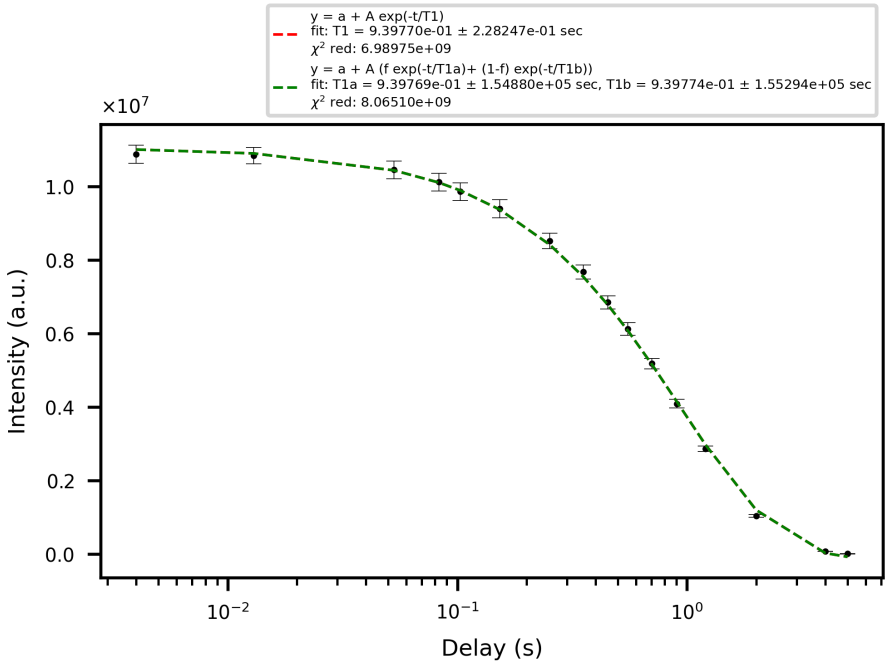

Figure 7: Intensity fit with monoexponential and biexponential functions. The fit results are reported in the figure captions.

at this point,

```
# saved in main4test.py

# function for the global analysis of the T1 determined at each VCLIST points
theUltimatePlot(dir_result, list_sp, bi_list=[], colormap = 'hsv', area=True)
```

where:

- `dir_result = "<folder name>"` is the name of the folder generated by the principal function, where all the data from the previous analysis are saved.
- `bi_list = [<n. peak 1>, <n. peak 2>,]` contains the list of peak indices whose intensity decay has to be treated with the biexponential model.
- `colormap = "<name of colormap>"` defines the color palette used in the plots (the complete list of colors can be found at https://matplotlib.org/stable/users/explain/colors/colormaps.html).
- `area = True or False` refers to the flag used in the intensity extraction.

The `theUltimatePlot()` function reads the files contained in each experiment folder and generates files `R1_<n.peak>.csv` of the type,

```
VCLIST      R1                  err
14092.38  1.06409038888853  0.258440984476471
9437.8      1.41200949035261  0.0467318739515697
```

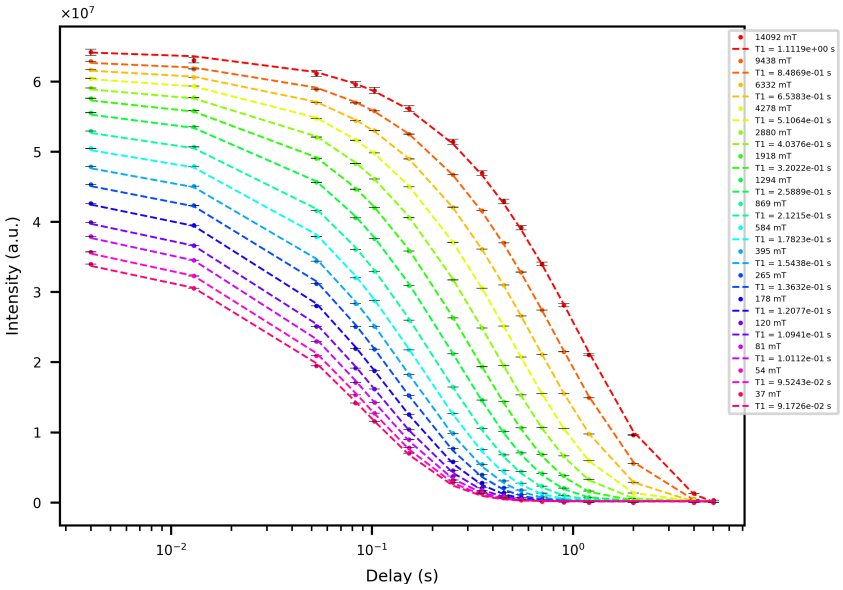

Figure 8: Intensity trends for each `VCLIST` element, with corresponding error bars and T_1 values from monoexponential curve fitting.

```
6331.66      2.02480929250211   0.029168530591605
4278.02      2.85065936512452   0.0166335078566074
2879.67      3.94691678812708   0.0209886838188778
1917.61      5.46601111816607   0.0447457733261291
1293.54      7.27254732450316   0.0557456589534411
868.59       9.29066447965837   0.0560800385773115
584.1       11.7595424156602    0.0625965114973621
395.07       14.5487546597909   0.0805150817280265
264.67       17.6661522591158   0.136887472802053
178.23        20.384758032573   0.217390022971099
120.07        21.984904994356   0.252407370211354
80.69       23.8009281723884    0.29504808485878
54.37       23.5809838485677    0.274279943954481
36.66       22.8147956437819    0.174327125094509
```

and plots of the intensity trends `<n.  peak>.png` (see figure 8) and of the R_1 dispersion curve `R1_<n.  peak>.png` (see figure 9).

The use of `doexp = True` and of the function `theUltimatePlot()` is an example of an analysis of trends that can be performed in this program (for `doexp = False` the plot generated are the same as the one in the previous section). In the following examples, a more detailed explanation of how to add your routine to the main functions for the analysis of the trends with different functions than those in `doexp` is presented.

Also in this case, the very same functions can be used for the analysis of a single pseudo-2D spectrum, just by definition of a single element `list_sp` and `delays_list` as

```
1
2 list_sp = ["spectrum1/pdata/10"]
3
4 delays_list = [np.array([0.0, 0.1, 0.2, 0.3, 0.4, 0.5])]
```

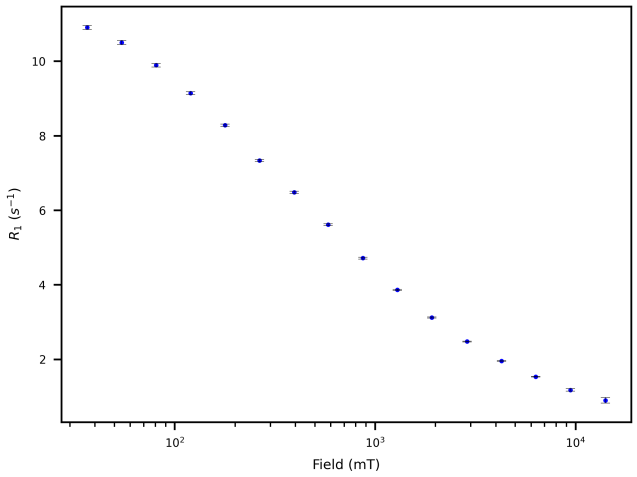

Figure 9: R_1 dispersion curve from intensity decays in figure 8.

4 Analysis of intensity trends via spectra modeling

In these functions for signal fitting, the cost function is defined by the minimization of difference between the real-valued NMR signal and the spectra model composed of a hard model for the peak shapes combined with a fourth-order polynomial representing the baseline. To compensate for slight distortions of the pure Lorentzian shape, different fractions of Gaussianity are introduced through the application of the Voigt model. The model signals are thus simulated, in the time domain, according to the equation:

$$s(t) = k \exp[i(\omega t + \phi)] \exp[-(1 - x_g)\sigma t/2] \exp[-x_g \Gamma^2 t^2/2] \tag{2}$$

where $\Gamma = \sigma/(2\sqrt{2\ln 2})$. The parameters that describe each peak are: the relative intensity k, the frequency with respect to the carrier ω (which will then be translated into the chemical shift δ), the full width at half maximum σ, the phase distortion ϕ, and the fraction of gaussianity x_g ($x_g = 0$ for pure Lorentzian, $x_g = 1$ for pure Gaussian). The timed-domain signal of the peaks is then transformed and summed to the baseline model:

$$y = A + Bx + Cx^2 + Dx^3 + Ex^4 \tag{3}$$

where the coefficients A, B, C, D and E are included as fitting variable.

4.1 Single 1D spectrum or series of 1D spectra

The same series of monodimensional spectra used in subsection 3.1 is used also in this example.

The main for the modeling of the peaks and baseline looks like the following:

```
1
2 # saved in main4test.py
3
4 from f_fit import *
5
```

```
# folder containing the spectra
path = "path/to/spectra/"

# in this case the experiment names correspond to consecutive numbers (from 101 to 128)
num_sp = list(np.arange(101,128,1))
list_sp = [str(i)+"/pdata/1" for i in num_sp]

delays = np.linspace(0, 1, len(list_sp))

# POSSIBLE KEYS: "shift", "k", "lw", "ph", "xg", "A", "B", "C", "D", "E"

# limits for the fit parameters defined in absolute terms as "key":(min,max)
# if the max and min are defined equal, the parameter is fixed
# if the limit is not defined, the parameter is free to vary between +np.inf and -np.inf
lim1 = {"shift":(-1,1), "lw":(1e-4,2.5), "ph":(-np.pi/20,np.pi/20),"A":(0,0), "B":(0,0), "C":(0,0), "D":(0,0), "E":(0,0)}

# the second limits are defined not in absolute terms but as a percentage of the initial value
lim2 = {"shift":(0.9,1.1), "lw":(0,0), "ph":(0,0), "xg":(0,0)}

model_fit_1D(
            path,
            delays,
            list_sp,
            cal_lim = (-72.52,-72.94),
            fast = True,
            limits1 = lim1,
            limits2 = lim2
            )
```

The variables `path`, `delays`, `list_sp` and `cal_lim` keep the same meaning as those of the function `intensity_fit_1D()` in subsection 3.1, the additional one are specific for the peak modeling, where:

- `fast = True or False` defines the fitting algorithm used for the minimization of the target function. If `fast = True` the least squares minimization is used, else, a two cycle minimization is used, where the first cycle of minimization is performed via the Nelder-Mead simplex algorithm, while the second cycle is performed using the least squares minimization[5]. The second option should be used in presence of strong baseline distortions and/or complex multiplets since the simplex algorithm, even if less efficient, is very effective in sampling the variables space and therefore is better than least squares in finding the global minima (or deeper minima), while the least squares is more efficient and better for refinement of already good guesses and/or less complex variable spaces.

- `limits1 = <dict> or None` is a dictionary where limits for the fit parameters are defined in absolute terms as `"key":(<min>,<max>)`. This dictionary will be used for the first spectra of the series. The shift is the only parameter whose limits are defined in terms of a delta value to be applied to the initial guess. Possible keys include: `"shift"`, `"k"`, `"lw"`, `"ph"`, `"xg"`, `"A"`, `"B"`, `"C"`, `"D"` and `"E"`. If the `<max>` and `<min>` are defined equal, the parameter is fixed to the initial guess value, and, if the limits are not defined (hence if the parameters `"key"` is not written in the dictionary), the parameter is free to vary between $+\infty$ and $-\infty$. If some of the keys are not included in the dictionary, default limits will be used, equivalent to the application of the follow-

ing dictionary: `lim1 = "shift":(-2,2), "k":(0,1), "lw":(0.0001,3.5), "ph":(-np.pi,np.pi), "xg":(0,1).`

- `limits2 = <dict> or None` is a dictionary where limits for the fit parameters are defined as a percentage of the initial value, e.g. if `"shift": (0.9, 1.1)` means that the value of the shift is allowed to vary between 0.9∗initial value and 1.1∗initial value. This second dict will be used for the spectra subsequent to the first one. If `lim2` is `None`, the same limits as those defined in `lim1` are used also for the subsequent spectra. In either of the two cases, the parameters value will be set equal to the fitting result of the preceding fit.

Once the script is executed, the result folder will be generated, named as `<spectra folder>_modelfit`, and the terminal will ask for the input files:

```
(base) $ python main4test.py
Writing directory <spectra folder>_modelfit
DIR:  <spectra folder>_modelfit
PATH TO SPECTRA:  path/to/spectra
Write new input1? ([y]|n) n
Type input1 filename: inp1
Write new input2? ([y]|n) n
Type input2 filename: inp2
```

Two input files are needed for this fit. The first input, if not already available in the execution folder, can be generated with the graphical interface in figures 10-12 with the following steps:

1. Select the fit intervals around single peaks or group of peaks by dragging regions around them and pressing `ADD` (analogously to the graphical interface described in section 3.1). Be sure to include portions of baseline around in the selected window.

2. Once all the fit regions are selected, perform peak picking by left-double click with the mouse over the signal to be selected. A red line will be added at the click position. To delete unwanted peak selections double click with the right button on the red line. The radio button `true/false` refers to the intensity estimation, if `false` is selected a black line will be placed at the selection position and the positioned peaks will be used just for fitting purposes, hence the integral will not be evaluated for them. For each selected signal, a Voigt model will be placed centered at the selected frequency.

3. Lastly, press `SAVE AND EXIT` to save all the selections and peaks.

The generated input file looks like the following:

```
name   ppm1   ppm2   v mult
true   -71.17900 -74.75500 -73.09469 0
true   -71.17900 -74.75500 -72.74464 0
```

where the type of peak, limits of the fitting region and chemical shift of each component are saved. The last column represents a flag for the definition of peaks multiplicity (*vide infra*).

Analogously, the graphical interface for the generation of the second input looks like figure 13:

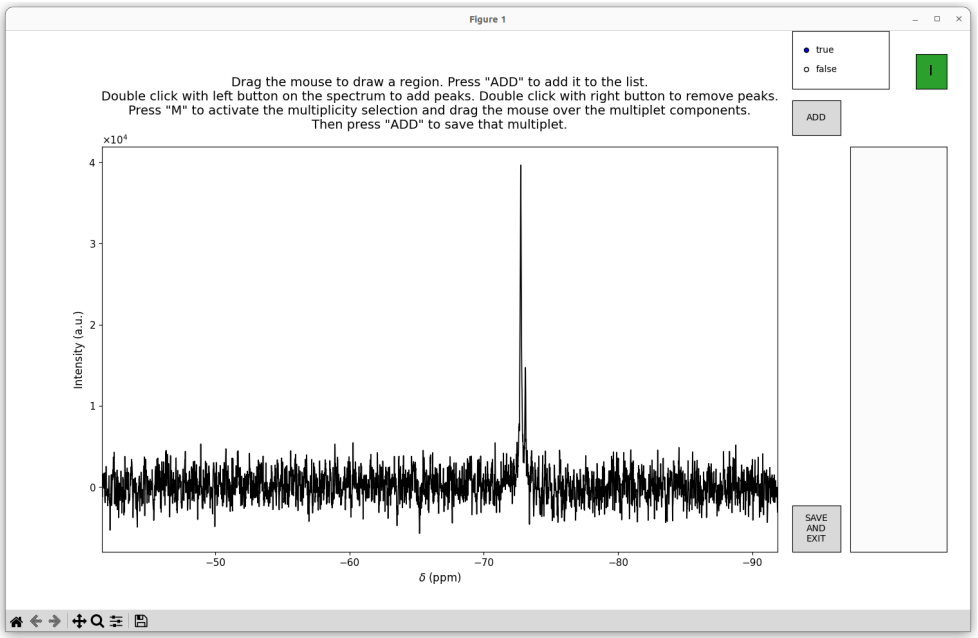

Figure 10: Graphical interface for the generation of the first input file for spectra fit.

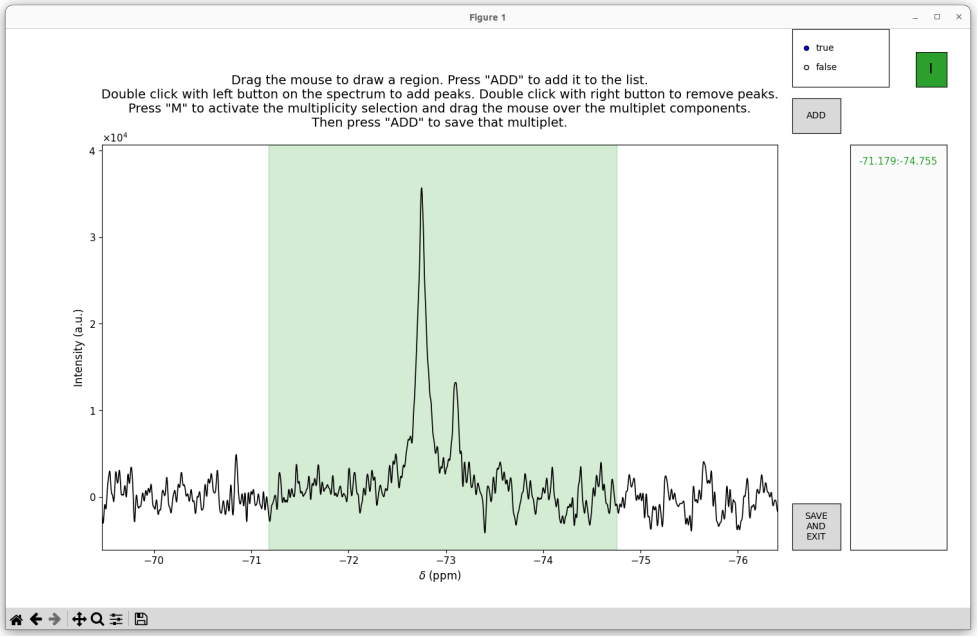

Figure 11: Graphical interface for the generation of the first input file for spectra fit at the first step.

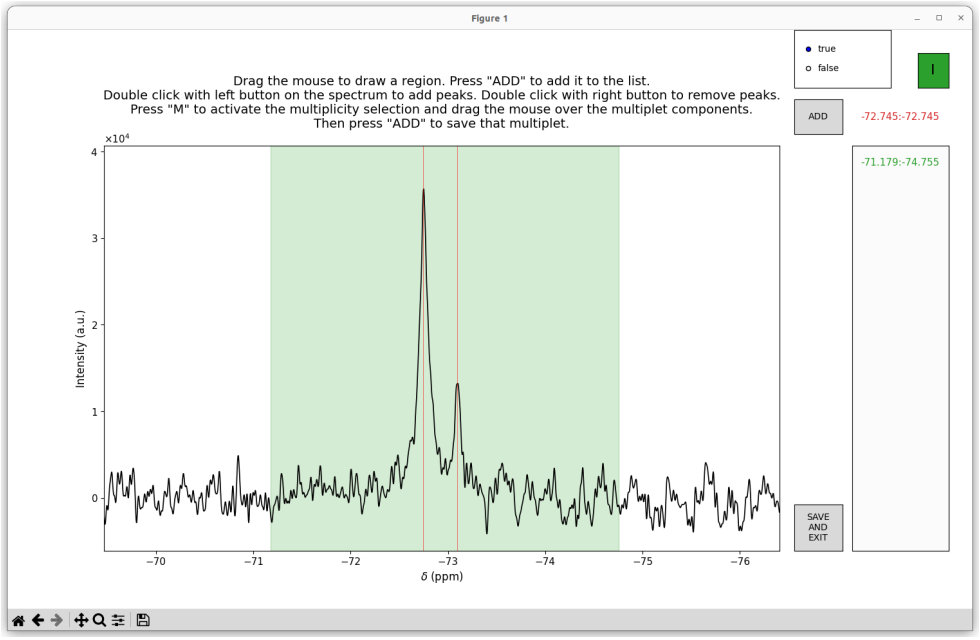

Figure 12: Graphical interface for the generation of the first input file for spectra fit at the second step.

1. Use the sliders to adjust the fit parameters of the model peaks and the mouse wheel to set the baseline coefficients. Don't panic if you don't see the experimental spectra when the second interface opens! It only means that it's very low in intensity and you just have to lower the `k_<n>`s and zoom more.

2. Once the guess is satisfactory, press `SAVE AND EXIT`.

This second input looks like the following:

```
i ppm1   ppm2   k    fwhm     phi    xg     A    B    C    D    E
0 -71.2 -74.8 1.98815e-03 7.56087e-02 0.00000e+00 2.00000e-01 0.000e+00 0.000e
    +00 0.000e+00 0.000e+00 0.000e+00
1 -71.2 -74.8 4.83487e-03 8.24822e-02 0.00000e+00 2.00000e-01 0.000e+00 0.000e
    +00 0.000e+00 0.000e+00 0.000e+00
```

The output file `<spectra folder>.out`, saved in the result folder together with a copy of the input files, is generated as:

```
SPECTRA PATH:
path/to/spectra

Points:
1 1.000
2 0.962
3 0.923
4 0.885
5 0.846
6 0.808
7 0.769
8 0.731
```

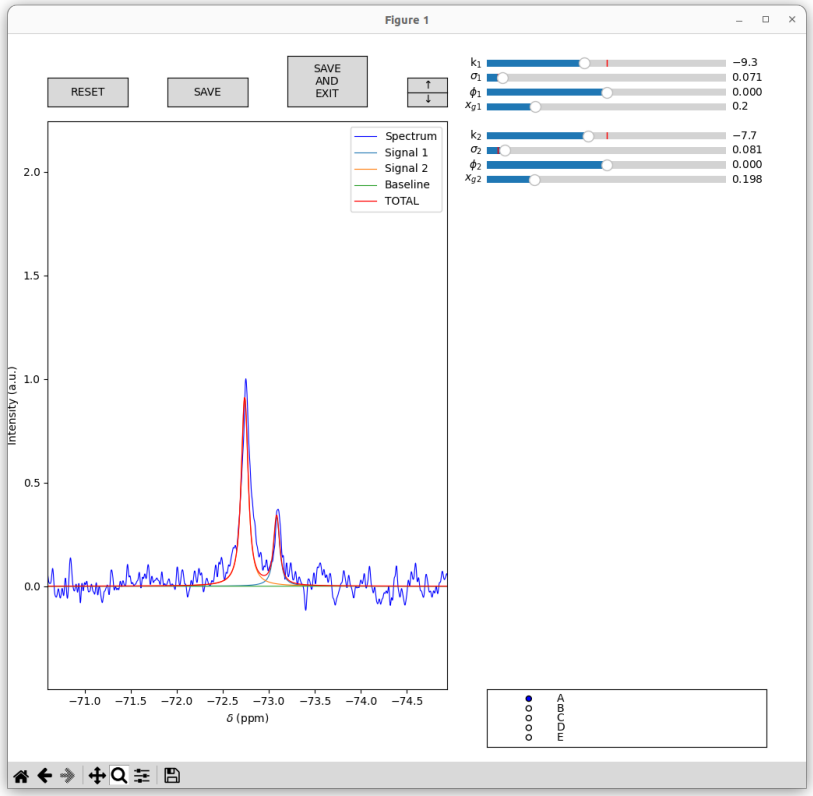

Figure 13: Graphical interface for the generation of the second input file for spectra fit.

```
9 0.692
10   0.654
11   0.615
12   0.577
13   0.538
14   0.500
15   0.462
16   0.423
17   0.385
18   0.346
19   0.308
20   0.269
21   0.231
22   0.192
23   0.154
24   0.115
25   0.077
26   0.038
27   0.000

INPUT1: inp1_3
n. peak name   ppm1   ppm2   v mult
1 true   -71.17900 -74.75500 -73.09469 0
2 true   -71.17900 -74.75500 -72.74464 0

INPUT2: inp2_3
```

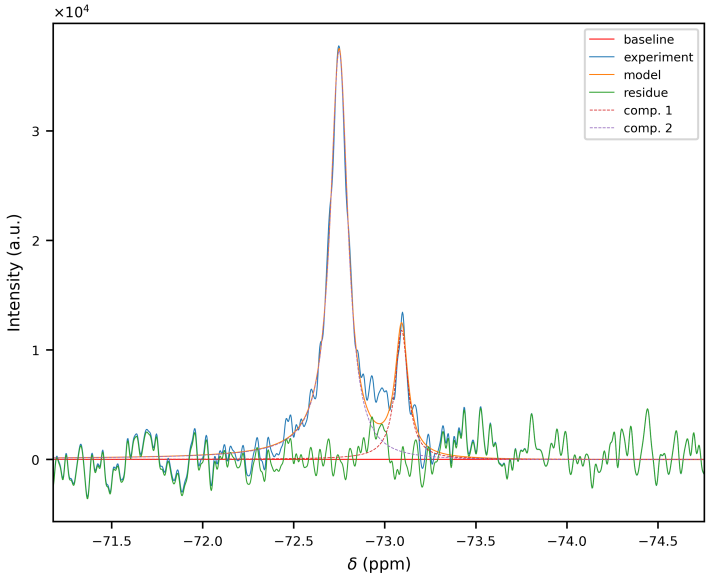

Figure 14: Plot of the target function for a single fitting interval.

```
n. peak i ppm1   ppm2   k    fwhm     phi    xg     A    B    C    D    E
1 0 -71.2 -74.8 1.98815e-03 7.56087e-02 0.00000e+00 2.00000e-01 0.000e+00 0.000e
    +00 0.000e+00 0.000e+00 0.000e+00
2 1 -71.2 -74.8 4.83487e-03 8.24822e-02 0.00000e+00 2.00000e-01 0.000e+00 0.000e
    +00 0.000e+00 0.000e+00 0.000e+00
================================================================================
```

At this point the fit starts. Each selected region is fitted separately. At the end of each fit iteration, two plots are saved, i.e. the plot of the target function `<spectra folder>_P<n>_I<n>.png` (see figure 14) and the histogram of the residuals `<spectra folder>_P<n>_I<n>_hist.png` (see figure 15). For the latter, the more the histogram resembles the Gaussian shape the better.

Once every interval of every spectrum of the series has been fitted, the fit reports for each interval (I) and point in the series (P), of the type showed in the previous section, are saved in the output as:

```
(I: fit interval, P: N. point)

------------------------------
FIT RANGE: (-71.18:-74.75) ppm
I: 1 P: 1
------------------------------

Fit Report:
[[Fit Statistics]]
    # fitting method   = leastsq
    # function evals   = 165
    # data points      = 1165
    # variables        = 10
    chi-square         = 3.1192e+09
```

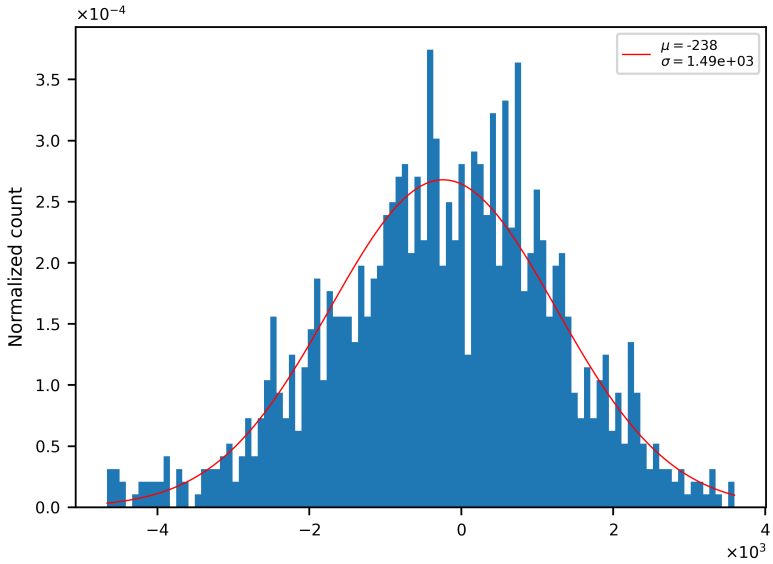

Figure 15: Histogram of the residuals for the fit of the region in figure 14.

```
    reduced chi-square = 2700592.12
    Akaike info crit   = 17262.4204
    Bayesian info crit = 17313.0252
[[Variables]]
    shift_1: -73.1046328 +/- 0.00235897 (0.00%) (init = -73.09469)
    k_1:      0.00186435 +/- 1.1553e-04 (6.20%) (init = 0.00198815)
    lw_1:     0.07553313 +/- 0.00633717 (8.39%) (init = 0.0756087)
    ph_1:     0.01739490 +/- 0.04433994 (254.90%) (init = 0)
    xg_1:     4.0618e-08 +/- 0.14777031 (363808566.49%) (init = 0.2)
    shift_2: -72.7600742 +/- 0.00107935 (0.00%) (init = -72.74464)
    k_2:      0.00668928 +/- 1.3199e-04 (1.97%) (init = 0.00483487)
    lw_2:     0.10728705 +/- 0.00272457 (2.54%) (init = 0.0824822)
    ph_2:     0.05484881 +/- 0.01504421 (27.43%) (init = 0)
    xg_2:     5.7016e-12 +/- 0.01404417 (246321977052.96%) (init = 0.2)
    A:        0 (fixed)
    B:        0 (fixed)
    C:        0 (fixed)
    D:        0 (fixed)
    E:        0 (fixed)
[[Correlations]] (unreported correlations are < 0.100)
    C(shift_2, ph_2)    = -0.7731
    C(shift_1, ph_1)    = -0.7665
    C(k_1, xg_1)        = +0.6897
    C(k_2, xg_2)        = +0.6830
    C(lw_1, xg_1)       = -0.6519
    C(lw_2, xg_2)       = -0.6140
    C(k_1, k_2)         = -0.5538
    C(k_1, ph_2)        = +0.5458
    C(ph_1, k_2)        = -0.5041
    C(k_1, xg_2)        = -0.4298
    C(xg_1, k_2)        = -0.4104
    C(k_1, shift_2)     = -0.3566
```

```
    C(shift_1, k_2)     = +0.3316
    C(xg_1, ph_2)       = +0.3162
    C(xg_1, xg_2)       = -0.2835
    C(ph_1, xg_2)       = -0.2602
    C(k_2, ph_2)        = -0.2185
    C(ph_2, xg_2)       = -0.1998
    C(k_1, ph_1)        = +0.1887
    C(xg_1, shift_2)    = -0.1875
    C(ph_1, xg_1)       = +0.1640
    C(shift_1, xg_2)    = +0.1511
    C(ph_1, shift_2)    = +0.1346
    C(shift_1, shift_2) = -0.1344
    C(shift_1, ph_2)    = +0.1179
    C(shift_2, xg_2)    = +0.1162
    C(shift_2, k_2)     = +0.1091

n.peak  Shift Integral
1    -73.105    605441.220 +/- 126303.074
2    -72.760    2172318.941 +/- 238216.701

------------------------------
FIT RANGE: (-71.18:-74.75) ppm
I: 1 P: 2
------------------------------

Fit Report:
[[Fit Statistics]]
    # fitting method   = leastsq
    # function evals   = 21
    # data points      = 1165
    # variables        = 4
    chi-square         = 2.9012e+09
    reduced chi-square = 2498848.91
    Akaike info crit   = 17166.0051
    Bayesian info crit = 17186.2470
[[Variables]]
    shift_1: -73.1048076 +/- 0.00169733 (0.00%) (init = -73.10463)
    k_1:      0.00155852 +/- 4.4432e-05 (2.85%) (init = 0.001864352)
    lw_1:     0.07553313 (fixed)
    ph_1:     0.0173949 (fixed)
    xg_1:     4.06176e-08 (fixed)
    shift_2: -72.7547437 +/- 6.5362e-04 (0.00%) (init = -72.76007)
    k_2:      0.00653607 +/- 5.2118e-05 (0.80%) (init = 0.006689284)
    lw_2:     0.1072871 (fixed)
    ph_2:     0.05484881 (fixed)
    xg_2:     5.70155e-12 (fixed)
    A:        0 (fixed)
    B:        0 (fixed)
    C:        0 (fixed)
    D:        0 (fixed)
    E:        0 (fixed)

n.peak  Shift Integral
1    -73.105    516586.117 +/- 231830.279
2    -72.755    2166440.198 +/- 287535.613
```

As you can notice from these two fit reports for the first and second points in the series,

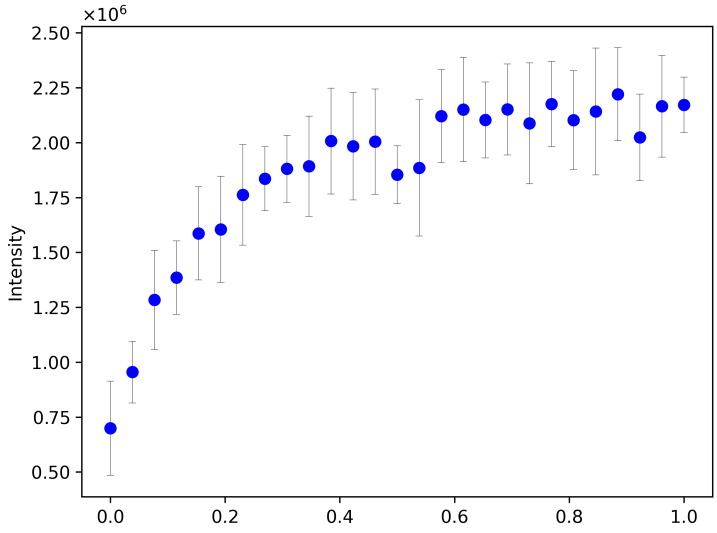

Figure 16: Intensity values are plotted against delays, with corresponding error bars.

`lw_<n>`, `ph_<n>` and `xg_<n>` are fixed to the optimized values from the previous fit (as dictated by `lim2`). The same holds also for the initial values of `shift_<n>` and `k_<n>`. The baseline coefficients are fixed to `0.0` (so to the value they have in `INPUT2`) as dictated by `lim1`.

The integrals are also printed, together with the estimated error. The integral of each signal is computed with the trapezoidal rule implemented in `numpy` and the error is estimated as the integral of the magnitude of the residual over a set of points that encloses a user-defined percentage of the total integral (in this case it is set to 95%), defined with the flag `err_conf = <value from 0 to 1>`. For peaks whose residual regions extend beyond the fitting interval, a contribution is added to the residuals proportional to the percentage of the missing peak area. For instance, if a peak is cut in half by the fitting region, the error would consist of the error in the evaluated region plus an additional 50% to account for the portion that lies outside the fitting interval.

Notice that, since in this case the integration is performed on the model, the truncation error is not present, in contrast to the integrals computed with `intensity_fit_1D()` and `intensity_fit_pseudo2D()`.

In the result folder, a series of files are also saved, containing the fitted parameters and named as `<spectra folder>_I<n>_P<n>`. They are generated with the `.dump()` method of the `lmfit Parameter` object. They can be read using the flag `dofit = False`, to not run the fit again and just analyze the results. An additional variable, `prev_fit = "path/to/directory"` can be used, that specifies the directory where these files are located (if `dofit = False`) or where they will be saved (if `dofit = True`).

The integrals, errors and delays list are saved in `y_<n>.txt`, `Err_<n>.txt`, `x_<n>.txt` files, respectively.

Since in this case `doexp = False`, the intensities are plotted as figure 16, named `peak_<n>.png`.

4.2 Single pseudo-2D spectrum or series of pseudo-2D spectra

Also in this case, the function analogous for the treatment of a series of pseudo-2D spectra is applied to the analysis of experiments acquired for the study of longitudinal relaxation rates at different fields.

```
1
2  # saved in main4test.py
3
4  from f_fit import *
5
6  # folder containing the spectra
7  path = "path/to/spectra"
8
9  list_path = [int(f) for f in os.listdir(path) if not f.startswith('.')]
10 list_path = [str(f) for f in np.sort(list_path)]
11
12 #list of delays values for each experiment
13 delays_list = [np.loadtxt(path+list_path[i]+'/vdlist')+0.003 for i in range(len(
       list_path))]
14
15 # POSSIBLE KEYS: "shift", "k", "lw", "ph", "xg", "A", "B", "C", "D", "E"
16
17 # limits for the fit parameters defined in absolute terms as "key":(min,max)
18 # if the max and min are defined equal, the parameter is fixed
19 # if the limit is not defined, the parameter is free to vary between +np.inf and
       -np.inf
20 lim1 = {'shift':(-0.05,0.05), 'lw':(1e-4,0.05), 'ph':(-np.pi/20,np.pi/20), 'k'
      :(0,1), 'C':(0,0), 'D':(0,0), 'E':(0,0)}
21
22 # the second limits are defined not in absolute terms but as a percentage of the
       initial value
23 lim2 = {'shift':(0.95,1.05), 'lw':(0,0), 'ph':(0,0), 'xg':(0,0)}
24
25 model_fit_pseudo2D(
26             path,
27             delays_list,
28             list_path,
29             cal_lim = (1.90,1.70),
30             fast = True,
31             dofit = True,
32             prev_fit = None,
33             limits1 = lim1,
34             limits2 = lim2,
35             prev_guess = True,
36             L1R = 8,
37             L2R = None,
38             doexp=True
39             )
```

Few more flags appear in this example. Few of them are unique for the pseudo-2D spectra, while others can be also applied in the monodimensional case. In the first category, there are:

- `delays_list`, which, analogously to the example in subsection 3.2, needs to be a list of arrays and not a simple list,
- `prev_guess = True or False`, refers to the use of the same initial guess for all the pseudo-2D. If `true` the same initial guess will be used, if `false` the program will ask for new `INPUT1` and `INPUT2` at each step of the `VCLIST` (so for each pseudo-2D). Notice that the initial guesses for the transients after the first one are derived from the fit result of the preceding transient.

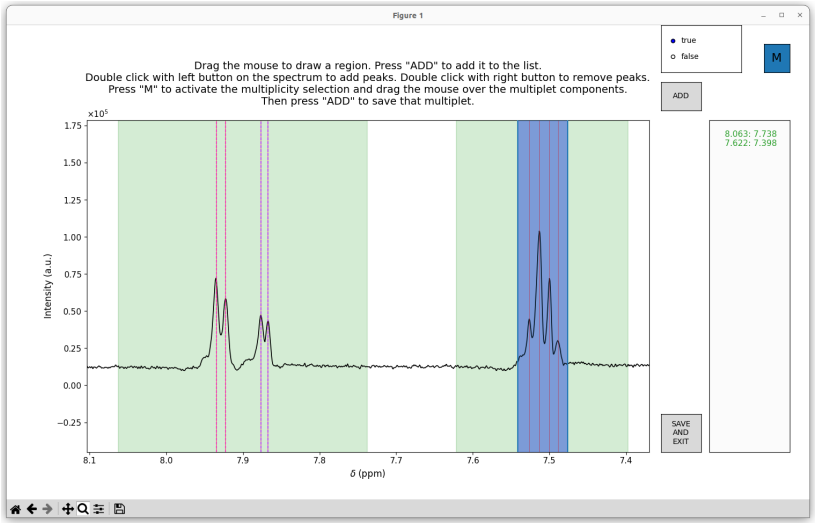

Figure 17: Enter Caption

The general flags are:

- `L1R = <value> or None` is the flag for the activation of the Lasso regularization method. If this value is not `None`, the sum of the absolute value of the parameters multiplied by the `L1R` factor is summed to the fit residue. This could favor a more stable fit for situations where the peak shape varies a lot among the various transients and/or in the presence of strong baseline distortions.

- `L2R = <value> or None` is the flag for the activation of the Ridge regularization method. If this value is not `None`, the sum of the squared value of the parameters multiplied by the `L2R` factor is summed to the fit residue. The function is the same the Lasso factor, but it is quadratic, so more strict.

In this example, some peaks are not singlets, in this case the multiplicity of the peak needs to be defined during the generation of `INPUT1` using the `mult` flag as follows (see figures 17 and 18):

1. Before pressing `SAVE AND EXIT`, hence after the selection of all integration regions and after the peak picking, press "M" on the keyboard to activate the multiplicity-selection mode, testified by the change of  in  .

2. Drag a region over the red lines (peaks) that you want to include in a single multiplet (the components can be also non-consecutive) and press `ADD`. At this point the blue region will disappear and in their place, a dotted line will appear over the peak selection lines. Equal color means that the peaks belong to the same multiplet.

The corresponding `INPUT1` looks like the following:

```
name   ppm1   ppm2   v mult
true   7.62200 7.39800 7.48912 3
true   7.62200 7.39800 7.50012 3
true   7.62200 7.39800 7.51332 3
```

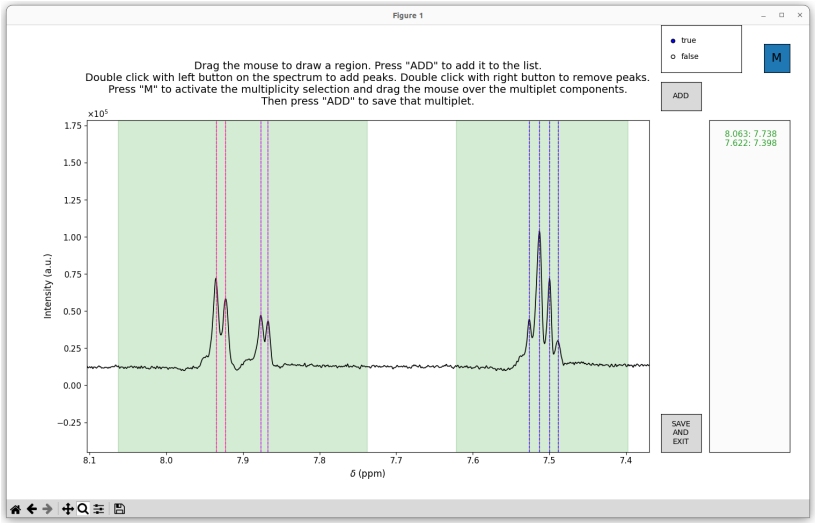

Figure 18: Enter Caption

```
true   7.62200 7.39800 7.52652 3
true   8.06300 7.73800 7.86755 2
true   8.06300 7.73800 7.87635 2
true   8.06300 7.73800 7.92256 1
true   8.06300 7.73800 7.93502 1
```

where, in the last column, an equal number means that the peaks belong to the same signal.

The result folder is organized as described in subsection 3.2.

References

[1] Travis E Oliphant et al. *Guide to numpy*. Vol. 1. Trelgol Publishing USA, 2006.

[2] Sandro Tosi. *Matplotlib for Python developers*. Packt Publishing Ltd, 2009.

[3] Matthew Newville et al. "LMFIT: Non-linear least-square minimization and curve-fitting for Python". In: *Astrophysics Source Code Library* (2016), ascl–1606.

[4] Jonathan J Helmus and Christopher P Jaroniec. "Nmrglue: an open source Python package for the analysis of multidimensional NMR data". In: *Journal of biomolecular NMR* 55 (2013), pp. 355–367.

[5] Per Christian Hansen, Victor Pereyra, and Godela Scherer. *Least squares data fitting with applications*. JHU Press, 2013.