pyIHM

Indirect Hard Modelling, in Python

Author:
Francesco Bruno, Letizia Fiorucci

Version: 2.0.0

Documentation release date:
November 14, 2024

Contents

1. Introduction

`pyIHM` is a python software designed in order to offer a comprehensive interface to perform quantitative analyses on NMR spectra of mixtures, using the Indirect Hard Modelling[1] approach.

The Indirect Hard Modelling consists into performing a deconvolution of the spectrum of the mixture using the spectra of the individual components as basis set. Conceptually, the algorithm is made of four steps:

1. fit the spectra of the components of the mixture with a hard model (e.g. Voigt);
2. read and process the spectrum of the mixture;
3. make the initial guess using the set of peaks generated at point 1;
4. get the relative concentrations of the components in the mixture.

The routines for reading and processing of the spectra and for the generation of the models rely on the `KLASSEZ`[2] package.

[1] Ernesto Kriesten et al. "Fully automated indirect hard modeling of mixture spectra". In: *Chemometrics and Intelligent Laboratory Systems* 91.2 (2008), pp. 181–193; Anton Duchowny et al. "Quantification of PVC plasticizer mixtures by compact proton NMR spectroscopy and indirect hard modeling". In: *Analytica Chimica Acta* 1229 (2022), p. 340384.

[2] *KLASSEZ: a package for the management of NMR data.* 2023. URL: `https://github.com/MetallerTM/klassez`.

2. User guide

2.1 Installation

`pyIHM` can be installed from the associated PyPI repository using `pip` from the command line by typing:

```
pip install pyihm
```

Alternatively, it is possible to download the `.whl` file, located in the `dist/` folder of the GitHub repository, and install it with `pip`:

```
pip install <filename>.whl
```

The dependencies of the program are downloaded and installed automatically.

2.2 Make the deconvolution of the spectra

A script for the deconvolution of the spectra is provided in table 2.1. It uses the `KLASSEZ` package to read, process and deconvolve the spectrum. The script must be edited in order to adapt the specific user's need. At the end of the run, the .fvf file will be saved in the folder of the spectrum. Alternatively, one can set a custom filename, for saving the .fvf files in different locations.

If you do not have an experimental spectrum of one (or more) component, you can simulate it with `KLASSEZ`. You have to produce an input file like the one in table 2.2, then use it in the given script. This generates a .fvf in the current working directory.

Table 2.1: Script for creating a .fvf file, to be used as input for `pyIHM`.

```
#! /usr/bin/env python3

import sys
import klassez as kz

filename = sys.argv[1] # Spectrum
spect = sys.argv[2]   # Format
custom_filename = None # Custom filename for the files

# Read the spectrum
S = kz.Spectrum_1D(filename, spect=spect)
# Do FT
S.process()
# Set "if 1" to phase correct
if 0:
    S.adjph()

# Create/read the initial guess for the deconvolution
S.F.iguess(filename=custom_filename)

# Perform the fit...
S.F.dofit( # ...with the following options:
        u_lim=0.2,        # variation on chemical shift /ppm
        f_lim=5,          # variation of FWHM /Hz
    k_lim=(0,2)    # variation of relative intensity (from, to)
        vary_phase=False, # Phase correction on the peaks
        vary_b=True,      # Fraction of gaussianity
    filename=custom_filename  # Makes <custom_filename>.fvf
        )
# Save the figures of the fit, with the residuals
S.F.plot('result', show_res=True, res_offset=0.1, filename=custom_filename)
```

Table 2.2: Example of input file for `kz.Spectrum_1D`, used to simulate the spectrum of a component. The file consists of two sections. In the first one, one must declare the experimental parameters to set up a virtual experiment, writing them in a `<key> <value>` format. Then, the parameters of the signals follow. They must be declared as a sequence of comma-separated values (`<key> <value 1>, <value 2>, <value 3>, ...`). `KLASSEZ` is able to simulate multiplets according to their splitting structure, with the given scalar coupling constants. This file should be used in the script provided at the bottom, for the conversion to a .fvf file to be used in `pyIHM`.

Input file:

```
## PARAMETERS OF THE VIRTUAL SPECTROMETER
B0  16      # Magnetic field /T
nuc 1H      # Observed nucleus
o1p 4.7     # Carrier position /ppm
SWp 50      # Spectral window /ppm
TD  2**16       # Number of sampled complex points

## PARAMETERS OF THE SIGNALS
#   Chemical shifts /ppm
shifts 4.69, 3.22, 2.79, 8.36, 12.69, 8.43, 4.96, 3.30, 3.15, 1.53
#   FWHM /Hz
fwhm   2, 2, 2, 2, 2, 2, 2, 2, 2, 2
#   Intensities (area) of the signals, i.e. the number of nuclei the signals integrate for
amplitudes 1, 1, 1, 1, 1, 1, 1, 1, 1, 3
#   Fraction of Gaussianity (0 = pure Lorentzian, 1 = pure Gaussian)
beta   0, 0, 0, 0, 0, 0, 0, 0, 0, 0
#   Multiplet structure: s = singlet, d = doublet, t = triplet, q = quartet
mult   dqd, dd, dd, s, s, d, ddd, dd, dd, d
#   Scalar coupling constants /Hz. They must match the correspondant multiplet (e.g. mult=ddt requres three J constants)
Jconst [11.5, 3.5, 6.3], [17.5, 11.5], [17.5, 3.5], 0, 0, 6.9, [6.9, 7.4, 5.4], [14.2, 5.4], [14.2, 7.4], 6.3
```

Script:

```
#! /usr/bin/env python3

import sys
import klassez as kz

filename = sys.argv[1] # Path to the input file

# Read the spectrum
S = kz.Spectrum_1D(filename, isexp=False)
# Do FT
S.process()
# Check if the spectrum looks like it should
S.plot()
# Create <filename>.fvf file
S.to_vf()
```

2.3 Writing the input file

The input file for PYIHM is a plain text file that consists in a series of keywords, that act as separators, followed by their arguments in the next lines. The sections of the file, identified by these keywords, are separated by **ONE** empty line.

A template for the input file is shown in table 2.3. The order does not matter. Not all of them are mandatory!

A detailed explanation of the keyword meanings and the syntax of the related parameters follows. The mandatory keywords are marked with a '*' sign, whereas the default value for the optional ones is reported after a '&'.

- `BASE_FILENAME`*
  Root of the name of all files that the program will save. It can also be a relative/absolute path.

- `MIX_PATH`*
  Path to the input spectrum (raw). The folder/file to be read is the first argument, followed by comma-separated additional parameters. It is very important to specify the spectrometer format to allow proper reading, using the `spect='format'` keyword. The accepted formats are: `bruker` for Bruker, `varian` for Varian/Agilent, `magritek` for SpinSolve benchtop, `oxford` for Oxford Instruments and general .jdx files.

- `PROC_OPTS`
  Processing options for the mixture spectrum.

  - `wf` & `wf: no`
    Window function options. Syntax: `wf: <mode>, <option>=<value>`. Examples:
    - No apodization:
    - `wf=no`
    - Exponential modulation: `wf: lb, lb=<broadening factor /Hz>`
    - Sine bell: `wf: sin, ssb=<ssb>`
    - Squared sine bell: `wf: qsin, ssb=<ssb>`
    - General Lorentzian to Gaussian: `wf: gm, lb_gm=-<sharpening factor>, gb_gm=<gaussia modulation>, gc=<center of gaussian>`
    - Lorentzian to Gaussian, Bruker-style: `wf: gmb, lb=-<sharpening factor>, gb=<gaussia modulation>`
  - `zf` & Do not write
    Zero-filling option. Syntax: `zf: <final shape>`
  - `blp` & Do not write
    Backward linear prediction. Syntax: `blp: pred=<number of points to predict & 1>, order=<number of lp coefficients & 8>`
  - `pknl` & Do not write
    Correct the effect of the group delay with a 1st order phase correction. No extra options required.
  - `adjph` & Do not write
    Apply manual phase correction on the spectrum. No extra options required.

- `MIX_SPECTRUM_TXT` & None
  Path to a plain text file that contains the intensity values of the real part of the spectrum. This has to be set if the mixture spectrum was processed with an external software: in this way you can use your own. Note that calibration of the chemical shift scale will have no effect. The use of this keyword overwrites all `PROC_OPTS`.

- `COMP_PATH`*
  List of the `.fvf` files that contain the parameters of the signals of the components, generated by `KLASSEZ`. See section 2.2 for details. Write one path per row. After the filename, place a comma and write the number of nuclei that the spectrum integrates, followed by an 'H'. Example: `component1.fvf, 10H`

- `FIT_LIMITS` & Interactive selection with GUI
  Limits of the fitting region, in ppm, separated by a comma. Multiple regions can be selected by writing them in multiple lines. If this parameter is not set in the input file, the program starts a GUI to select them interactively.

- `FIT_BDS`*
  Tolerances for the parameters during the fit.

  - `utol`: tolerance for the chemical shifts, in ppm. Given the starting chemical shift δ, they will vary in the interval $[\delta - \texttt{utol}, \delta + \texttt{utol}]$. If one of the boundaries happens to fall out the fitting window, the limit of the fitting window il used instead.
  - `utol_sg`: tolerance for the chemical shifts of the signals within the same group, in ppm. Given the starting chemical shift δ, they will vary in the interval $[\delta - \texttt{utol\_sg}, \delta + \texttt{utol\_sg}]$.
  - `stol`: tolerance for the linewidths. Given the starting linewitdh Γ, in Hz, they will vary in the interval $[\Gamma - \texttt{stol}, \Gamma + \texttt{stol}]$. If the lower boundary happens to be negative, 0 is used instead.
  - `ktol`: tolerance for the relative intensities of the signals in the same spectrum. Given the starting intensity k, they will vary in the interval $[k - \texttt{ktol}, k + \texttt{ktol}]$. If the lower boundary happens to be negative, 0 is used instead.

- `FIT_KWS` & `method='leastsq', max_nfev=10000, tol=1e-5`
  Parameters for `lmfit.Minimizer.minimize`.

- `PLT_OPTS` & `ext='tiff', dpi=300`
  Set specific resolution (`dpi`) and format (`ext`) for the figures that will be saved.

- `I0` & 1 for all components
  Initial guess for the relative concentrations.

Table 2.3: Example of the input file, used to run the test located in the `test/` folder of the GitHub repository.

```
BASE_FILENAME
mix

MIX_PATH
test_spectrum.fid, spect='varian'

PROC_OPTS
wf: em, lb=1
zf: 2**16
adjph

COMP_PATH
comp/bzac.fvf, 5H
comp/dmso.fvf, 6H
comp/EC.fvf, 4H

FIT_BDS
utol=0.1
utol_sg=0.01
stol=5
ktol=0.01

FIT_KWS
method='leastsq', tol=1e-5, max_nfev=10000

PLT_OPTS
ext=png, dpi=600

FIT_LIMITS
  8.032,  7.858
  7.676,  7.437
  4.548,  4.412
  2.553,  2.426
```

2.4 Starting a `pyIHM` run

The software can be operated from the command line by typing:

```
    python3 -m pyihm --input <input_file> <options>
```

where `<input_file>` is the path to the input file that contains the parameters for the run, and `<options>` are flags for specific functions (see below).

Multiple input files can be given at once, writing their paths in sequence without punctuation signs between them.

```
    python3 -m pyihm --input <input_file_1> <input_file_2> <input_file_3> <options>
```

Here, the possible options for a `PYIHM` run follow:

- `--debug`: during the fitting routines, a figure that shows how the optimization process is going is saved in the current working directory and updated every 20 iterations.
- `--cal`: before to start the fit, the initial guess can be refined with a dedicated GUI, that allows to shift the spectra to correct for field drifts, and provide a better estimation of the starting intensities.
- `--rav`: shows the GUIs with a colorblind palette
- `--opt_method=`: optimization method for the core fit of pyIHM
  - `tight` (default): two-step optimization, first with Nelder-Mead simplex and then with Levenberg-Marquardt least-squares
  - `fast`: single-step optimization with Levenberg-Marquardt least-squares
  - `custom`: reads the `'FIT_KWS'` section of the input file, and performs the optimization accordingly
- `--noalgn`: skips the alignment fit
- `--help`: displays this message on the terminal.

2.5 Calibrations

If `pyihm` is run with the `--cal` option, the input guess can be refined though a dedicated GUI. An example is shown in figure 2.1a. This interface displays the mixture spectrum, and the spectra of the components, superimposed. You can shift the spectrum of each component left or right, in order to compensate for large field drifts, and to set the initial concentrations, by using the radiobutton and either the mouse scroll or the UP/DOWN keys. To change the active spectrum, you can move the slider, or use the PAG-UP and PAG-DOWN keys.

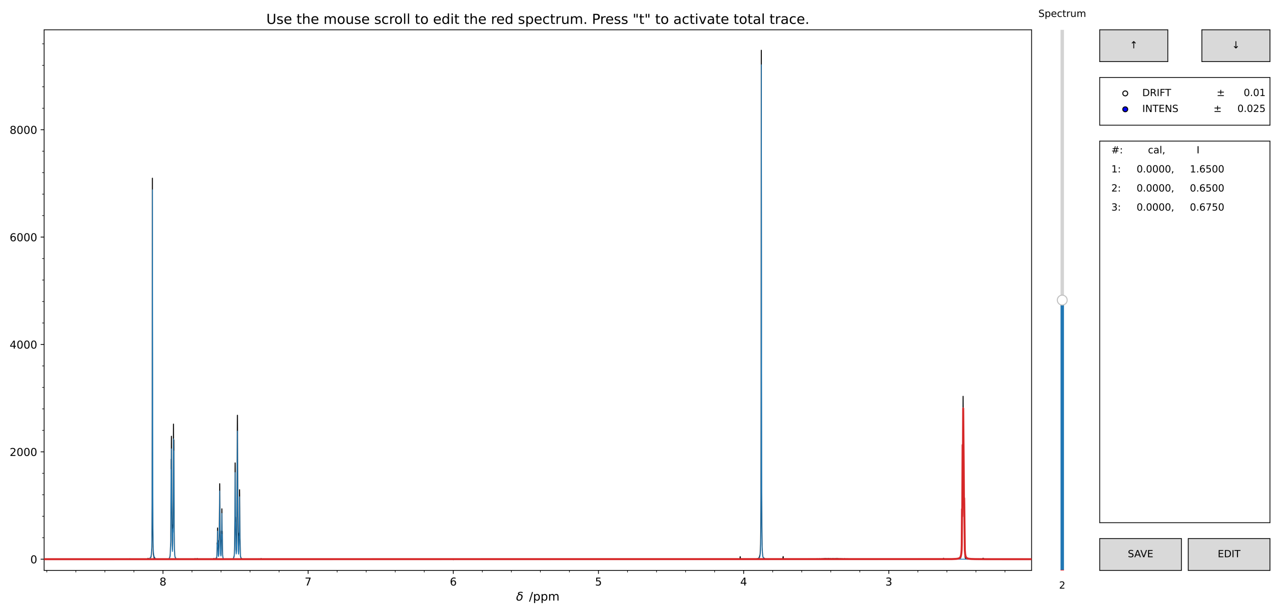
A further degree of refinement can be performed together with this interface through another GUI, that allows to change the appearance of the spectrum peak-by-peak (figure 2.1b). To open it, click on the 'EDIT' button of the first GUI. The interface closes and another one opens, that displays the mixture spectrum in black, the spectrum that was active in the previous GUI in green, and all the other components in blue. You have to zoom on the part of the spectrum that you want to edit, and press 'UNLOCK'. The region of the green spectrum is thus unpacked into all its spectral feature. You can edit them using the mouse scroll and the radiobuttons. Once you are done with a particular region, press 'LOCK': the peaks turn again in a solid green line, and you can work on another region of the spectrum. The reason for this 'LOCK/UNLOCK' loop is to not overload the PC RAM with too many graphicals, which would make the interface to crash.

Once you are satisfied with your modifications, pressing the 'SAVE' button brings you again to the first GUI, and you can repeat the whole process for all the components until you press 'SAVE'. At this point, the calibration process is finished, and the corrected spectra of the components are saved in a separate file with the pattern `<original filename>-cal.fvf`. At the beginning of the calibration, `pyIHM` looks for this kind of files: if it finds them, it prompts a message, where you can decide whether to read the original or the already calibrated version. As a consequence, editing the input file is not necessary.

The list of borders for each window have to be read from the input file under the `FIT_LIMITS` key. However, if it is not given, it can be interactively selected by the user through a GUI (example in figure 2.2). Here, you can drag a red span on the figure to mark a region. The active limits are displayed, in red, in the upper-right corner of the interface. Press the 'ADD' button to add the region to the list: the drawn span will become green. Do not worry about overlapping regions, as they are automatically merged afterwards.

Once you are done, press the 'SAVE' button. The interface will close, and `pyIHM` will prompt the section to add to the input file to avoid to do this selection again.

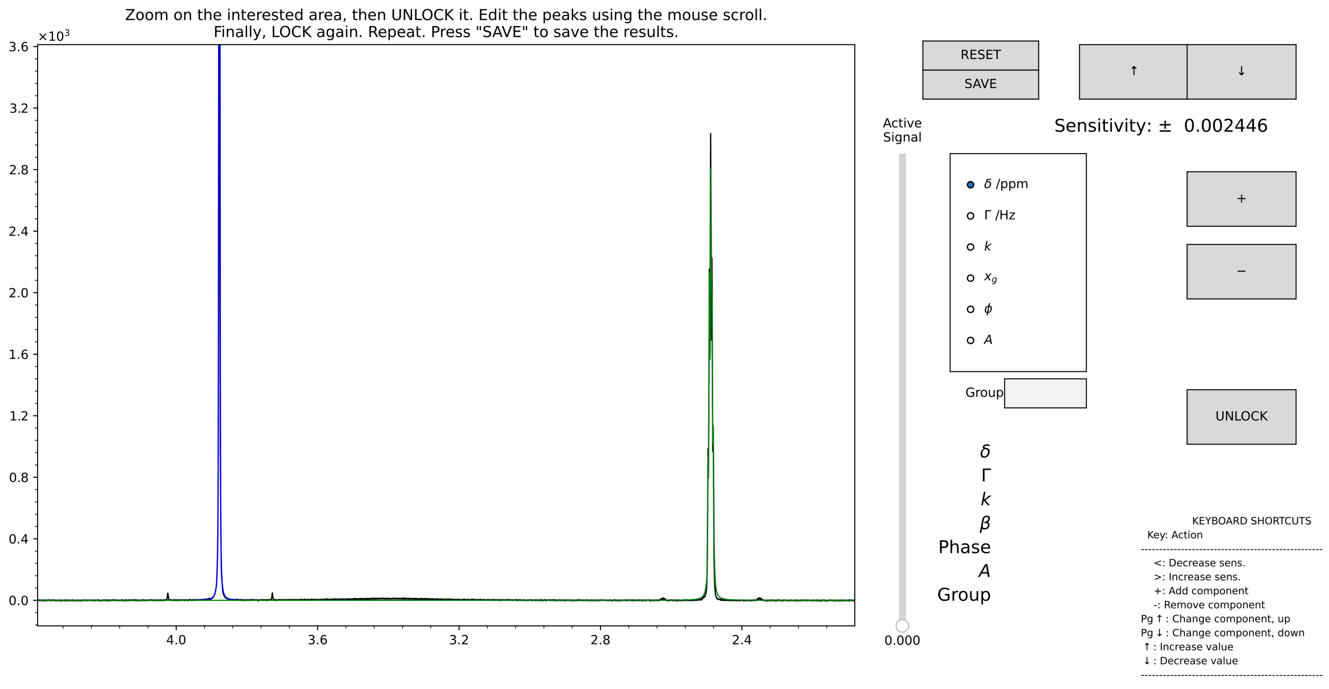
After the selection, the variable `Hs` that contains the intensity of each component is corrected to account only the peaks within the selected region.


Use the mouse scroll to edit the red spectrum. Press "t" to activate total trace.

#:	cal,	I
1:	0.0000,	1.6500
2:	0.0000,	0.6500
3:	0.0000,	0.6750

DRIFT ± 0.01
● INTENS ± 0.025

(a)


Zoom on the interested area, then UNLOCK it. Edit the peaks using the mouse scroll.
Finally, LOCK again. Repeat. Press "SAVE" to save the results.

Sensitivity: ± 0.002446

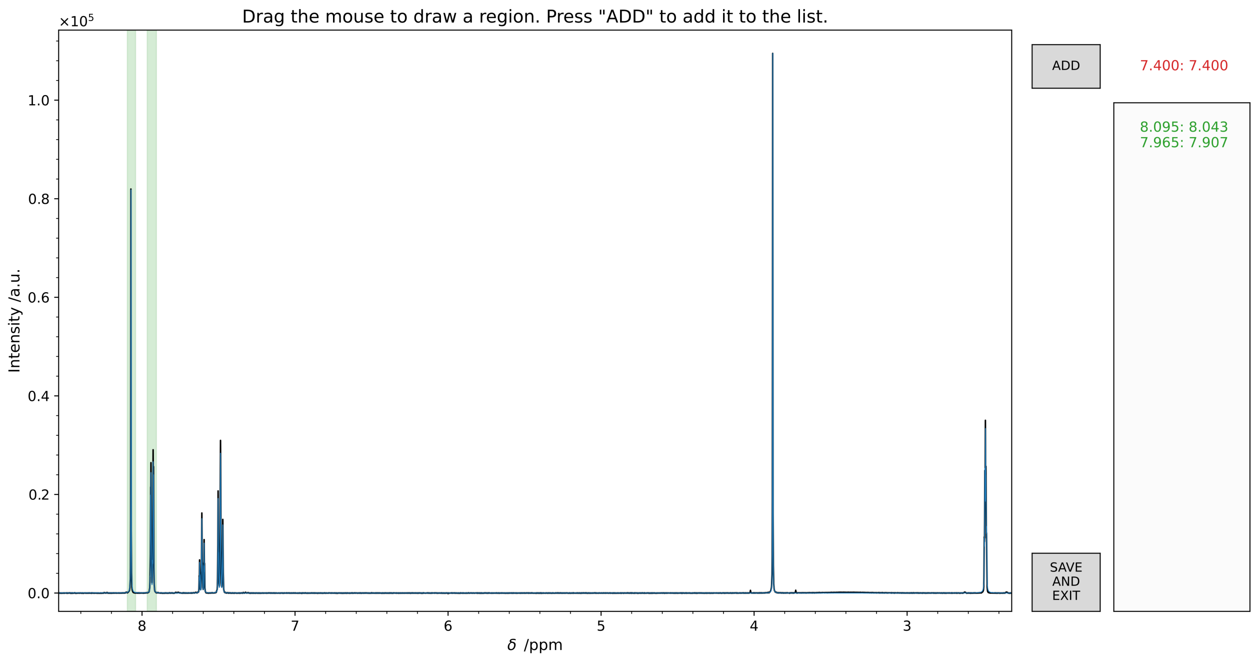
● δ /ppm
Γ /Hz
k
x_g
ϕ
A

KEYBOARD SHORTCUTS

```
Key: Action
---------------------------------
  <: Decrease sens.
  >: Increase sens.
  +: Add component
  -: Remove component
Pg ↑ : Change component, up
Pg ↓ : Change component, down
 ↑ : Increase value
 ↓ : Decrease value
---------------------------------
```

(b)

Figure 2.1: The refinement of the initial guess is performed by two GUIs that work in loop, synergically. The first GUI, in panel **a**, allows for the selection of the concentration of the spectra, and to correct for drifts that affect the component as a whole. The second one, in panel **b**, can refine the component spectrum peakwise.


Drag the mouse to draw a region. Press "ADD" to add it to the list.
×10⁵
1.0
0.8
0.6
0.4
0.2
0.0
Intensity /a.u.
8
7
6
5
4
3
δ /ppm
ADD
SAVE AND EXIT

7.400: 7.400

8.095: 8.043
7.965: 7.907

Figure 2.2: GUI for the selection of the regions to be used in the fit. The user can drag the red span with the mouse until it covers the desired region. Upon clicking on 'ADD', the selection becomes permanent, and the span changes to green. The process can continue until the 'SAVE' button is pressed.

2.6 Results

The following table contains a summary of the files saved by `pyIHM` at the end of each run.

Directory	Filename	Notes
Current working directory	`<filename>.out`	Summary of the fit result, which includes the composition of the mixture and the parameters of all the peaks.
`<filename>-DATA`	`<filename>_iguess.csv`	Initial guess. The columns contain, respectively: the ppm scale, the mixture spectrum, the total fitting function, and the simulated components.
	`<filename>_algn.csv`	Spectra after the alignment fit. The columns contain, respectively: the ppm scale, the mixture spectrum, the total fitting function, and the simulated components.
	`<filename>_result.csv`	Spectra after the final fit. The columns contain, respectively: the ppm scale, the mixture spectrum, the total fitting function, and the simulated components.
	`<filename>.cnvg`	Convergence pathway, i.e. the value of the target function of the main fit as function of the iteration step.
`<filename>-FIGURES`	`<filename>_iguess.<ext>`	Initial guess. The spectra relative to different components are drawn in different colors.
	`<filename>-algn_total.<ext>`	Spectra after the alignment fit, only total trace, with residuals.
	`<filename>-algn_wcomp.<ext>`	Spectra after the alignment fit, highlighted with different colors, without residuals.
	`<filename>-algn_rhist.<ext>`	Histogram of the residuals of the alignment fit, compared to a Gaussian curve computed with the mean and standard deviation of the residuals themselves.
	`<filename>_total.<ext>`	Spectra after the final fit, only total trace, with residuals.
	`<filename>_wcomp.<ext>`	Spectra after the final fit, highlighted with different colors, without residuals.
	`<filename>_rhist.<ext>`	Histogram of the residuals of the final fit, compared to a Gaussian curve computed with the mean and standard deviation of the residuals themselves.
	`<filename>_cnvg.<ext>`	Convergence path, i.e. $\log_{10}$ of the value of the target function as function of the iteration step.

3. How does PYIHM work

3.1 Details about peak simulation

`pyIHM` represent a spectrum as a collection of `kz.Peak` objects. This means that they are simulated in the time domain with the Voigt model, a mixed Lorentzian-Gaussian lineshape:

$$s^{\text{Voigt}}(t) = I \exp\left[\mathrm{i}\omega t\right] \exp\left[-(1-\beta)\Gamma t/2\right] \exp\left[-\beta\sigma^2 t^2/2\right] \qquad \sigma = \frac{\Gamma}{2\sqrt{2\ln 2}}$$

Each peak is described by four parameters:

- the position of the peak, ω, which translates in the chemical shift;
- the linewidth, or better, the full-width at half-maximum, Γ;
- the intensity, I;
- the fraction of gaussianity, β, which defines the lineshape.

The fifth parameter, the phase of the peak, is not included in this implementation, as the mixture spectrum is supposed to be phased before to start the fit. Figure 3.1 shows the impact of the aforementioned parameters on the final appearence of the peak, after the Fourier transform.

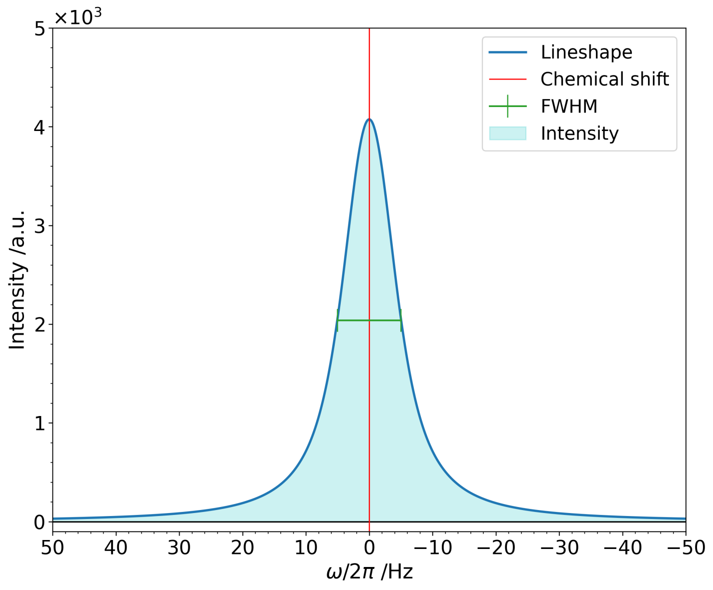


Figure 3.1: Simulated signal with a Voigt model after Fourier transform. The feature of the signal associated with the Voigt parameters are highlighted with different colors.

When we simulate an entire spectrum as sum of peaks, in order to make the model as general as possible, we factorize it as:

$$s^{\text{mixture}}(t) = I^{\text{tot}} \sum_{k=1}^{\#\text{ peaks}} K_k \, s^{\text{Voigt}}(t \,|\, \omega_k, \Gamma_k, \beta_k)$$

where K_k is the relative intensity of the k-th peak, such that $\sum_{k=1}^{\#\text{ peaks}} K_k = 1$.

3.2 Why the fit is in two steps?

The biggest problem one encounters in a simulation of this kind is the alignment of the chemical shifts.

Let us suppose that we know the structure of the multiplet and the relative intensities of the splitted features. If we try to find the correct 'central' chemical shift that describes the multiplet with a grid-search algorithm, we can draw the error surface associated to this parameter. Figure 3.2 depicts the problem very clearly: a little misalignment between the experimental and calculated signal translates in a local minimum of the error surface, resulting in a sudden arrest of the searching algorithm. Depending on the fine structure of the signal, the actual shape of the error surface may change, but its roughness remains an ubiquitous characteristic.

In order to find the correct alignment, it is therefore needed to employ a target function that gives rise to a smoother error surface. Figure 3.3 shows the error surface $\varphi^{\text{algn}}(\delta)$ computed as the difference between the integral (i.e. the cumulative sum) of the experimental and calculated spectrum.

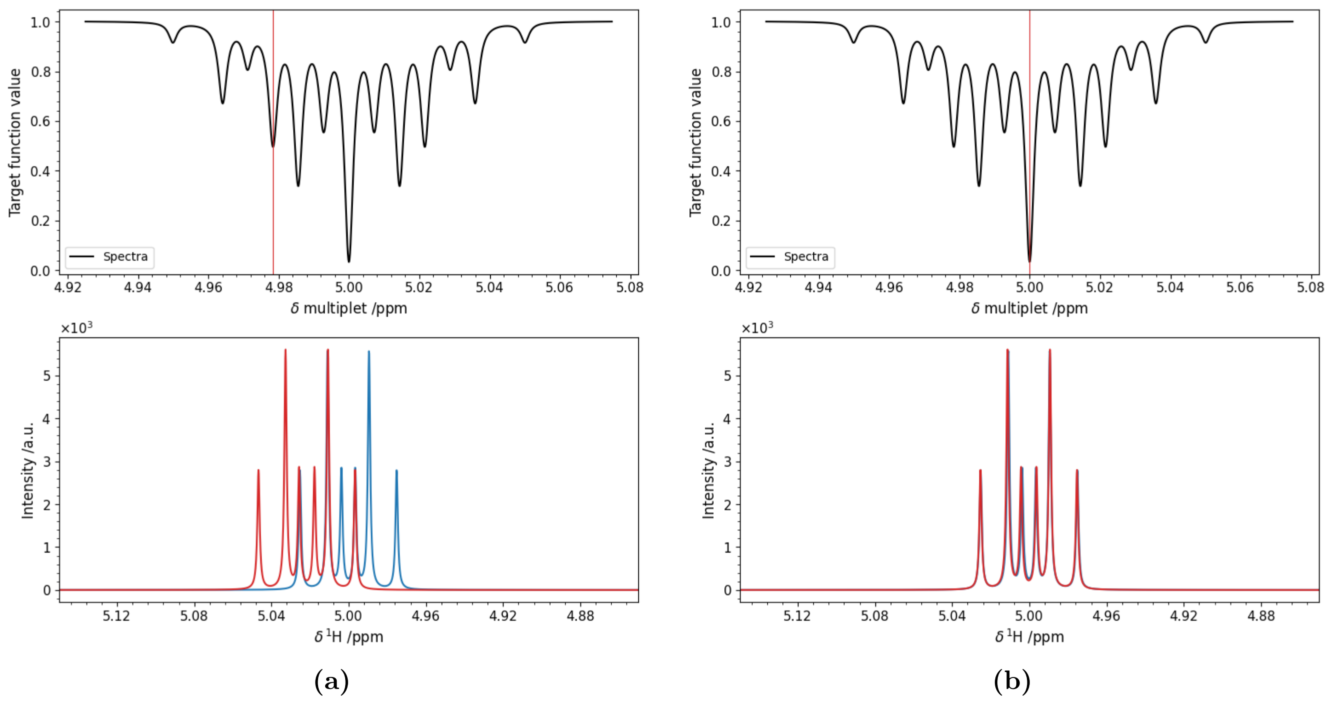
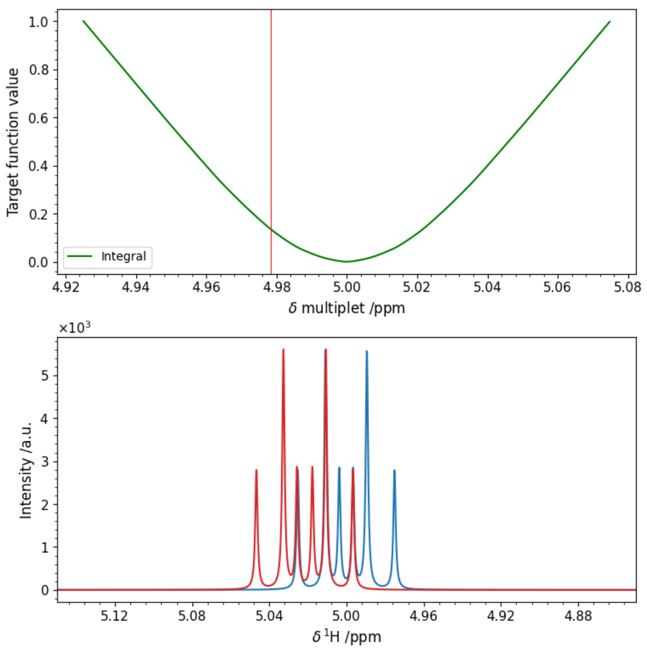
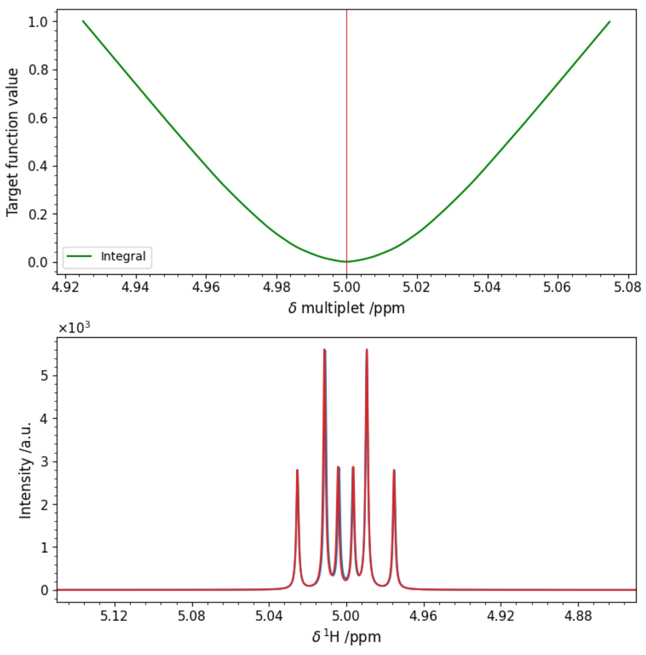


Figure 3.2: Simulated multiplet at 700 MHz ^{1}H Larmor frequency, centered at 5.00 ppm, with doublet of triplets fine structure ($J_d = 15\,\text{Hz}, J_t = 10\,\text{Hz}$). In the bottom panels, the experimental multiplet is drawn in blue, and the calculated one in red. The target function, calculated as the sum of the squared difference between the red and the blue trace, is sketched in black in the top panels, with a red line that marks the current value of the investigated parameter.

Of note, the alignment fit with this latter target function is insensitive to linewidth overestimates. This means that a model with broader signals than the experimental spectrum will be aligned correctly at the end of the fit. The same cannot be said for sure for a model with narrower signals.



(a)



(b)

Figure 3.3: Simulated multiplet at 700 MHz ^{1}H Larmor frequency, centered at 5.00 ppm, with doublet of triplets fine structure ($J_d = 15\,\mathrm{Hz}, J_t = 10\,\mathrm{Hz}$). In the bottom panels, the experimental multiplet is drawn in blue, and the calculated one in red. The target function, calculated as the sum of the squared difference between the integrals of the red and the blue trace, is sketched in green in the top panels, with a red line that marks the current value of the investigated parameter.

Technically speaking, the alignment fit is done by blocking all the parameters except for the chemical shifts. Taken the experimental spectrum y, the model spectrum computed with only chemical shifts, $y_c = M(\vec{\omega})$, and their cumulative sum Y and Y_c respectively, the target function is computed as:

$$\varphi(\vec{\omega}) = \sum_{\text{points}} \left[A\left(Y - Y_c\right) + q\right]^2$$

where A and q are the factors that satisfy the least-squares conditions:

$$A = \frac{\langle Y_c\, Y\rangle - \langle Y_c\rangle\,\langle Y\rangle}{\langle Y_c^2\rangle - \langle Y_c\rangle^2} \qquad q = \frac{\langle Y_c\rangle^2\,\langle Y\rangle - \langle Y_c\rangle\,\langle Y_c\, Y\rangle}{\langle Y_c^2\rangle - \langle Y_c\rangle^2}$$

Once the chemical shifts are aligned, the optimization of the classic target function (i.e. $\varphi = \sum(y - y_c)^2$) is straightforward. The only feature that needs a little bit of explanation is how the chemical shifts of multiplets are treated. We decided not to hard-model a fine structure with the proper splitting given by multiplication of the FID times $\cos(2\pi J)$, hence optimizing the J constants during the fit, because of the possible imperfections that might be present in the experimental dataset, especially at low fields. Therefore, peaks in the same spectrum that have the same 'group' attribute are recognized as part of the same multiplet.

Let us suppose that we have a multiplet composed of M features. Then, in the dictionary of parameters, the key 'U' identifies the central chemical shift of the multiplet, whereas the keys o_j for $j = 1, \ldots, M$ are the *offset* of the j-th feature of the multiplet with respect to U. The parameters u_j, i.e. the actual chemical shifts of the signals, are not optimized, but computed on the fly during the fitting as $u_j = U + o_j$.

In this framework, the U parameters are treated as singlets, and therefore obey the `utol` boundaries. By contrast, the o_j parameters are more or less fixed with respect to their U, hence they move according to the `utol_sg` $\leq$ `utol` boundaries.

4. List of modules and functions

4.1 MODULE input_reading

4.1.1 input_reading.read_input(filename)

Reads the input file to get all the information to perform the fit. The values read from the file are double-checked, and the missing entries are replaced with default values, so not to leave space to stupid mistakes.

Parameters:

- filename: *str*
  Path to the input file

Returns:

- base_filename: *str*
  Root of the name of all the files that the program will save
- mix_path: *str*
  Path to the mixture spectrum
- mix_kws: *dict of keyworded arguments*
  Additional instructions to be passed to kz.Spectrum_1D.__init__
- mix_spectrum_txt: *str or None*
  Path to a .txt file that contains a replacement spectrum for the mixture dic['proc_opt'],
- comp_path: *list*
  Path to the .fvf files to be used for building the spectra of the components
- fit_lims: *tuple*
  Limits of the fitting region, in ppm
- fit_bds: *dict*
  Boundaries for the fitting parameters. The keywords are:
  - utol = allowed displacement for singlets and whole multiplets, in ppm
  - utol_sg = allowed displacement for the peaks that are part of the same multiplet relatively to the center, in ppm
  - stol = allowed variation for the linewidth, in Hz
  - ktol = allowed variation for the relative intensities within the same spectrum
- fit_kws: *list of dic*
  Keyworded arguments for each run of the fit in custom mode (see lmfit)

- plt_opt: *dic*
  Format and resolution of the figures: `{'format'='tiff', 'dpi'=300}`
- Hs: *list*
  Nominal integrals for the components
- I0: *list or None*
  Initial guess for the concentrations

4.1.2 input_reading.read_input_file(filename)

Runs over the input file, looks for specific keywords, and interpret them accordingly.

Parameters:

- filename: *str*
  Path to the input file

Returns:

- dic: *dict*
  Read values, organized

4.1.3 input_reading.select_regions(ppm_scale, spectrum, full_calc)

Interactively select the slices that will be used in the fitting routine.

Parameters:

- ppm_scale: *1darray*
  ppm scale of the spectrum
- spectrum: *1darray*
  Spectrum of the mixture
- full_calc: *1darray*
  Spectrum of the initial guess, with all the peaks in total

Returns:

- regions: *list of tuple*
  Limits, in ppm

4.2 MODULE spectra_reading

4.2.1 spectra_reading.Multiplet `class`

Class that represent a multiplet as a collection of peaks.

Attributes:

- acqus: *dict*
  Dictionary of acquisition parameters
- peaks: *dict*
  Dictionary of kz.fit.Peak objects
- U: *float*
  Mean chemical shift of the multiplet
- u_off: *dict*
  Chemical shift of the components of the multiplet, expressed as offset from self.U

Methods:

__init__(self, acqus, *peaks)

Initialize the class.

Parameters:

- acqus: *dict*
  Dictionary of acquisition parameters
- peaks: *kz.fit.Peak objects*
  Peaks that are part of the multiplet. They must have an attribute 'idx' which serves as label

---

__call__(self)

Compute the trace correspondant to the multiplet.

Returns:

- trace: *1darray*
  Sum of the components

---

par(self)

Computes a summary dictioanary of all the parameters of the multiplet.

Returns:

- dic: *dict of dict*
  The keys of the inner dictionary are the parameters of each single peak, the outer keys are the labels of the single components

---

4.2.2 spectra_reading.Spectr `class`

Class that represents a spectrum as a collection of peaks and multiplets.

Attributes:

- acqus: *dict*
  Acquisition parameters
- peaks: *dict*
  Dictionary of peaks object, labelled according to the 'idx' attribute of each single peak
- unique_groups: *list*
  Identifier labels for the multiplets, without duplicates
- p_collections: *dict*
  Dictionary of kz.fit.Peak and Multiplet objects, labelled according to the group they belong to. In particular, self.p_collections[0] is a list of kz.fit.Peak objects, whereas all the remaining entries consist of a single Multiplet object.
- total: *1darray*
  Placeholder for the trace of the spectrum, as sum of all the peaks.

Methods:

__init__(self, acqus, *peaks)

Initialize the class.

Parameters:

- acqus: *dict*
  Dictionary of acquisition parameters
- peaks: *kz.fit.Peak objects*
  Peaks that are part of the multiplet. They must have an attribute 'idx' which serves as label

---

__call__(self, I=1)

Compute the total spectrum, multiplied by I.

Parameters:

- I: *float*
  Intensity value that multiplies the spectrum

Returns:

- total: *1darray*
  Computed spectrum

---

calc_total(self)

Computes the sum of all the peaks to make the spectrum

Returns:

- total: *1darray*
  Computed spectrum

4.2.3 spectra_reading.get_component_spectrum(filename, acqus, return_dict N=None, norm=True)

Reads a .ivf or .fvf file, and builds a dictionary of kz.fit.Peak objects with the information therein. Then, depending on return_dict, this function either returns it as is, or sums them up together to obtain the spectrum as array.

Parameters:

- filename: *str*
  Path to the .ivf or .fvf file
- acqus: *dict*
  Dictionary of acquisition parameters.
- return_dict: *bool*
  If True, returns the dictionary of kz.fit.Peak objects. If False, returns the computed spectrum as 1darray.
- N: *int or None*
  Total number of points of the spectrum. If None, the length of the acqusition timescale is used.
- norm: *bool or float*
  If True, all the peak intensities sum up to 1. If it is a number, the intensities sum up to that number. If False, they sum up to the intensity written in the .vf file

Returns:

if return_dict is True:

- dic_Peaks: *dict*
  Dictionary of kz.fit.Peak parameters
- I: *float*
  Total intensity

else:

- total: *1darray*
  Computed spectrum
- I: *float*
  Total intensity

4.2.4 spectra_reading.main(M, spectra_dir, Hs, lims=None, I0=None, cal_flag=False, rav_flag=False)

Reads the .fvf files, containing the fitted parameters of the peaks of a series of spectra. Then, computes a list of Spectr objects with those parameters, and returns it. The relative intensities are referred to the total intensity of the whole spectrum, not to the ones of the fitted regions. Employs kz.fit.read_vf to read the .fvf files and generate the parameters.

Parameters:

- M: *kz.Spectrum_1D object*
  Mixture spectrum. Used to get the spectral parameters for the kz.fit.Peak objects
- spectra_dir: *list of str*
  Sequence of the locations of the .fvf files to be read
- Hs: *list*
  Number of protons each spectrum integrates for
- lims: *list of tuple*
  Borders of the fitting windows, in ppm (left, right)
- I0: *list*
  Initial guess for the concentrations
- cal_flag: *bool*
  If True, opens the optimization of the initial guess through GUI
- rav_flag: *bool*
  If True, the GUIs will be drawn in colorblind palette

Returns:

- components: *list of Spectr*
  Spectra to be used as model in the fit. Contain only the peaks within the fittin regions.
- Hs: *list*
  Corrected integrals to be used for quantification
- I0: *list*
  Initial concentrations of the components in the mixture
- lims: *list of tuple*
  Fitting windows, in ppm
- c_idx: *list*
  Indices of the components to be used
- I: *float*
  Theoretical integral of the mixture spectrum, normalized

4.3 MODULE gen_param

4.3.1 gen_param.as_par(name, value, lims=0, rel=True, minthresh=None)

Creates a lmfit.Parameter object using the given parameters.

Parameters:

- name: *str*
  Label of the parameter
- value: *float or str*
  If it is float, it is the value of the parameter. If it is a str, it is put in the 'expr' attribute of the lmfit.Parameter object.
- lims: *float or tuple*
  Determines the boundaries. If it is a tuple, the boundaries are min(lims) and max(lims). If it is a single float, the boundaries are (value-lims, value+lims). Not read if value is str
- rel: *bool*
  Relative boundaries. If it is True and lims is a float, the boundaries are set to value-lims*value, value+lims*value.
- minthresh: *float*
  If given, overwrite the minimum threshold with this value, if the calculated one is lower than it.

Returns:

- p: *lmfit.Parameter object*
  Object created according to the given parameter

4.3.2 gen_param.main(M, components, bds, lims, Hs, c_idx, I0=None)

Create the lmfit.Parameters objects needed for the fitting procedure.

Parameters:

- M: *kz.Spectrum_1D object*
  Mixture spectrum
- components: *list*
  List of Spectra objects
- bds: *dict*
  Boundaries for the fitting parameters.
- lims: *list of tuple*
  Borders of the fitting windows, in ppm (left, right)
- Hs: *list*
  Number of protons each spectrum integrates for
- c_idx: *list*
  index of the components that are actually used in the fit
- I0: *list*
  Initial guess for the concentrations. If None, 1 is used for each spectrum

Returns:

- param: *lmfit.Parameters object*
  Actual parameters for the fit

4.3.3 gen_param.multiplet2par(item, spect, group, bds)

Converts a Multiplet object into a list of lmfit.Parameter objects. The keys are of the form 'S#_p?' where # is spect and ? is the index of the peak. p = U is the mean chemical shift p = o is the offset from U p = u is the absolute chemical shift, computed as U + o, set as expression.

Parameters:

- item: *fit.Peak object*
  Peak to convert into Parameter. Make sure the .idx attribute is set!
- spect: *int*
  Label of the spectrum to which the peak belongs to
- group: *int*
  Label of the multiplet group
- bds: *dict*
  Contains the parameters' boundaries

Returns:

- p: *list*
  List of lmfit.Parameter objects

4.3.4 gen_param.singlet2par(item, spect, bds)

Converts a fit.Peak object into a list of lmfit.Parameter objects: the chemical shift (u), the linewidth (s), and intensity (k). The keys are of the form 'S#_p?' where # is spect and ? is the index of the peak.

Parameters:

- item: *kz.fit.Peak object*
  Peak to convert into Parameter. Make sure the .idx attribute is set!

- spect: *int*
  Label of the spectrum to which the peak belongs to

- bds: *dict*
  Contains the parameters' boundaries

Returns:

- p: *list*
  List of lmfit.Parameter objects

4.4 MODULE fit_mixture

4.4.1 fit_mixture.calc_spectra(param, N_spectra, acqus, N)

Computes the spectra to be used as components for the fitting procedure, in form of lists of 1darrays. Each array is the sum of all the peaks. This function is called at each iteration of the fit.

Parameters:

- param: *lmfit.Parameters object*
  Actual parameters

- N_spectra: *int*
  Number of spectra to be used as components

- acqus: *dict*
  Dictionary of acquisition parameters

- N: *int*
  Number of points for zero-filling, i.e. final dimension of the arrays

Returns:

- spectra: *list of 1darray*
  Computed components of the mixture, weighted for their relative intensity

---

4.4.2 fit_mixture.calc_spectra_obj(param, N_spectra, acqus, N)

Computes the spectra to be used as components for the fitting procedure, in form of lists of kz.fit.Peak objects.

Parameters:

- param: *lmfit.Parameters object*
  Actual parameters
- N_spectra: *int*
  Number of spectra to be used as components
- acqus: *dict*
  Dictionary of acquisition parameters
- N: *int*
  Number of points for zero-filling, i.e. final dimension of the arrays

Returns:

- spectra: *list of kz.fit.Peak objects*
  Computed components of the mixture, weighted for their relative intensity

---

4.4.3 fit_mixture.f2min(param, N_spectra, acqus, N, exp, I, plims, cnvg_path, debug=False)

Function to compute the quantity to be minimized by the fit.

Parameters:

- param: *lmfit.Parameters object*
  actual parameters
- N_spectra: *int*
  Number of spectra to be used as components
- acqus: *dict*
  Dictionary of acquisition parameters
- N: *int*
  Number of points for zero-filling, i.e. final dimension of the arrays
- exp: *1darray*
  Experimental spectrum
- I: *float*
  Intensity correction for the calculated spectrum. Used to maintain the relative intensity small.
- plims: *slice*
  Delimiters for the fitting region. The residuals are computed only in this regio. They must be given as point indices
- cnvg_path: *str*
  Path for the file where to save the convergence path
- debug: *bool*
  If True, saves a figurte of the ongoing fit in the current working directory every 20 iterations

Returns:

- t_residual: *1darray*
  exp/I - calc

4.4.4 fit_mixture.f2min_align(param, N_spectra, acqus, N, exp, plims, debug=False)

Function to compute the quantity to be minimized by the fit.

Parameters:

- param: *lmfit.Parameters object*
  actual parameters
- N_spectra: *int*
  Number of spectra to be used as components
- acqus: *dict*
  Dictionary of acquisition parameters
- N: *int*
  Number of points for zero-filling, i.e. final dimension of the arrays
- exp: *1darray*
  Experimental spectrum
- plims: *slice*
  Delimiters for the fitting region. The residuals are computed only in this regio. They must be given as point indices
- debug: *bool*
  True for saving a figure of the ongoing fit every 20 iterations

Returns:

- t_residual: *1darray*
  exp/I - calc

4.4.5 fit_mixture.main(M, components, bds, lims, Hs, c_idx, I0=None)

Create the lmfit.Parameters objects needed for the fitting procedure.

Parameters:

- M: *kz.Spectrum_1D object*
  Mixture spectrum
- components: *list*
  List of Spectra objects
- bds: *dict*
  Boundaries for the fitting parameters.
- lims: *list of tuple*
  Borders of the fitting windows, in ppm (left, right)
- Hs: *list*
  Number of protons each spectrum integrates for
- c_idx: *list*
  index of the components that are actually used in the fit
- I0: *list*
  Initial guess for the concentrations. If None, 1 is used for each spectrum

Returns:

- param: *lmfit.Parameters object*
  Actual parameters for the fit

4.4.6 fit_mixture.main(M, N_spectra, Hs, param, I, lims=None, fit_kws={}, filename='fit', NOALGN_FLAG=False, DEBUG_FLAG=False, METHO ext='tiff', dpi=600)

Core of the fitting procedure. It computes the initial guess, save the figure, then starts the fit. After the fit, writes the output file and saves the figures of the result. Summary of saved files:

- '<filename>.out': fit report
- '<filename>_iguess.<ext>': figure of the initial guess
- '<filename>_total.<ext>': figure that contains the experimental spectrum, the total fitting function, and the residuals
- '<filename>_wcomp.<ext>': figure that contains the experimental spectrum, the total fitting function, and the components in different colors. The residuals are not shown
- '<filename>_rhist.<ext>': histogram of the residual, with a gaussian function drawn on top according to its statistical parameters.

Parameters:

- M: *kz.Spectrum_1D object*
  Mixture spectrum
- N_spectra: *int*
  Number of spectra to be used as fitting components
- Hs: *list*
  Number of protons each spectrum integrates for
- param: *lmfit.Parameters object*
  Actual parameters
- lims: *list of tuple or None*
  Delimiters of the fitting region, in ppm. If None, the whole spectrum is used.
- fit_kws: *dict of keyworded arguments*
  Additional parameters for the lmfit.Minimizer.minimize function
- filename: *str*
  Root of the names for the names of the files that will be saved.
- NOALGN_FLAG: *bool*
  If True, skips the alignment fit
- DEBUG_FLAG: *bool*
  True for saving a figure of the ongoing fit every 20 iterations
- METHOD_FLAG: *str*
  Method to be used for the fit. Can be 'fast', 'tight', 'custom'
- ext: *str*
  Format of the figures
- dpi: *int*
  Resolution of the figures, in dots per inches

4.4.7 fit_mixture.pre_alignment(exp, acqus, N_spectra, N, plims, param, DEBUG_FLAG=False)

Makes a fit with all the parameters blocked, except for the chemical shifts, on the target function of the integral. Used to improve the initial guess in case of misplacements of the signals.

Parameters:

- exp: *1darray*
  Experimental spectrum
- acqus: *dict*
  Dictionary of acquisition parameters
- N_spectra: *int*
  Number of spectra to be used as components
- N: *int*
  Number of points for zero-filling, i.e. final dimension of the arrays
- plims: *list of slice*
  Delimiters for the fitting region. The residuals are computed only in these regions. They must be given as point indices
- param: *lmfit.Parameters object*
  actual parameters
- DEBUG_FLAG: *bool*
  True for saving a figure of the ongoing fit every 20 iterations

Returns:

- popt: *lmfit.Parameters object*
  Parameters with optimal chemical shifts

4.4.8 fit_mixture.save_data(filename, ppm_scale, exp, *opt_spectra)

Saves the ppm scale, the experimental spectrum, the total trace and the components in .csv files, to be opened with excel, origin, or whatever.

Parameters:

- filename: *str*
  Location of the filename to be saved, without the .csv extension.
- ppm_scale: *1darray*
  PPM scale of the experimental spectrum
- exp: *1darray*
  Experimental spectrum, real part
- opt_spectra: *sequence of 1darray*
  Spectra of the components

4.4.9 fit_mixture.write_output(M, I, K, spectra, n_comp, lims, filename='fit.r

Write a report of the performed fit in a file. The parameters of the single peaks are saved using the kz.fit.write_vf function.

Parameters:

- M: *kz.Spectrum_1D object*
  Mixture spectrum
- I: *float*
  Absolute intensity for the calculated spectrum
- K: *sequence*
  Relative concentrations of the components spectra in the mixture
- spectra: *list of kz.fit.Peak objects*
  Computed components of the mixture, weighted for their relative intensity
- n_comp: *list*
  Indices of the components of the mixture
- lims: *tuple*
  Upper and lower boundaries of the fit region
- filename: *str*
  Name of the file where to write the files.

4.5 MODULE plots

4.5.1 plots.convergence_path(conv_path, filename='conv', ext='tiff', dpi=600

Makes the figures of the final fitted spectrum and saves them. Three figures are made: look at the fitting.main function documentation for details.

Parameters:

- conv_path: *str*
  Path to the file of the convergence path

- filename: *str*
  Filename of the final figure

- ext: *str*
  Format of the figure

- dpi: *int*
  Resolution of the figure, in dots per inches

4.5.2 plots.plot_iguess(ppm_scale, exp, total, components, lims=None, plims=None, X_label='$ delta$ /ppm', filename='fit', ext='tiff', dpi=600)

Makes the figure of the initial guess and saves it.

Parameters:

- ppm_scale: *1darray*
  PPM scale of the spectrum
- exp: *1darray*
  Mixture spectrum, real part
- total: *1darray*
  Fitting function
- components: *list of 1darray*
  Spectra used as components, real part
- lims: *tuple or None*
  Delimiters of the fitting region, in ppm. If None, the whole spectrum is used.
- plims: *list of slice or None*
  Delimiters of the fitting windows, to cut the residuals
- X_label: *str*
  Label for the X_axis
- filename: *str*
  The name of the figure will be <filename>_iguess.<ext>
- ext: *str*
  Format of the figures
- dpi: *int*
  Resolution of the figures, in dots per inches

4.5.3 plots.plot_output(ppm_scale, exp, total, components, lims=None, plims=None, X_label='$ delta$ /ppm', filename='fit', ext='tiff', dpi=600, windows=False)

Makes the figures of the final fitted spectrum and saves them. Three figures are made: look at the fitting.main function documentation for details.

Parameters:

- ppm_scale: *1darray*
  PPM scale of the spectrum
- exp: *1darray*
  Mixture spectrum, real part
- total: *1darray*
  Fitting function
- components: *list of 1darray*
  Spectra used as components, real part
- lims: *tuple or None*
  Delimiters of the fitting region, in ppm. If None, the whole spectrum is used.
- plims: *list of slice or None*
  Delimiters of the fitting windows, to cut the residuals
- X_label: *str*
  Label for the X_axis
- filename: *str*
  Root filename for the figures
- ext: *str*
  Format of the figures
- dpi: *int*
  Resolution of the figures, in dots per inches
- windows: *bool*
  If True, saves a separate figure for each fitting window

4.6 MODULE GUIs

Module that contains graphical user interfaces.

4.6.1 GUIs.cal_gui(exp, ppm_scale, components, I, prev_Icorr=None, init_active=1, rav_flag=False)

Corrects the chemical shifts and the intensities of the spectra to be employed during the fit. Works together with edit_gui, that allows to break up a spectrum in single components in order to adjust them.

Parameters:

- exp: *1darray*
  Experimental spectrum
- ppm_scale: *1darray*
  Chemical shift scale of the spectrum
- components: *list of 1darray*
  Spectra to calibrate
- I: *float*
  Intensity correction for the experimental spectrum
- prev_Icorr: *list*
  Starting concentration to be further edited
- init_active: *int*
  Initial spectrum to be activated
- rav_flag: *bool*
  Activates colorblind paelette

Returns:

- exit_code: *int*
  If 0, there is nothing to edit further. If not, it is the number of the component to be edited with edit_gui. The python-ish index is therefore (exit_code - 1)
- drifts: *1darray*
  Correction for the chemical shift correction of each spectrum, in ppm
- Icorr Correction for the intensity of each spectrum

4.6.2 GUIs.calc_spectra(param, N_spectra, acqus, N)

Computes the spectra to be used as components for the fitting procedure, in form of lists of 1darrays. Each array is the sum of all the peaks. This function is called at each iteration of the fit.

Parameters:

- param: *lmfit.Parameters object*
  Actual parameters
- N_spectra: *int*
  Number of spectra to be used as components
- acqus: *dict*
  Dictionary of acquisition parameters
- N: *int*
  Number of points for zero-filling, i.e. final dimension of the arrays

Returns:

- spectra: *list of 1darray*
  Computed components of the mixture, weighted for their relative intensity

---

4.6.3 GUIs.edit_gui(exp, ppm_scale, peaks, t_AQ, SFO1, o1p, offset=None, I=1, A0=None, rav_flag=False)

Opens a GUI for the interactive edit of a spectrum, peak-by-peak. The usage resembles the one for the manual computation of the initial guess for the fit in KLASSEZ. At the end, all the relative intensities of the peaks sum up to 1. You need to restore the total intensity outside this function. For this reason, it is not possible to add new peaks, or remove already existing ones. For this latter option, consider to move them in an empty region of the spectrum, and exclude them with select_regions.

Parameters:

- exp: *1darray*
  Experimental spectrum

- ppm_scale: *1darray*
  PPM scale of the spectrum

- peaks: *dict*
  Dictionary of kz.fit.Peak objects to edit with the GUI

- t_AQ: *1darray*
  Acquisition timescale

- SFO1: *float*
  Nucleus Larmor frequency /MHz

- o1p: *float*
  Carrier frequency /ppm

- offset: *1darray or None*
  External contribution to be added to the total trace. If None, an array of zeros is used

- I: *float*
  Intensity correction for the experimental spectrum

- A0: *float*
  Starting total intensity for the spectrum to edit (use concentration from cal_gui)

- rav_flag: *bool*
  Uses colorblind palette

Returns:

- peaks: *dict*
  Modified kz.fit.Peak dictionary

- Acorr: *float*
  Correction for the intensity

4.6.4 GUIs.select_regions(ppm_scale, spectrum, full_calc)

Interactively select the slices that will be used in the fitting routine.

Parameters:

- ppm_scale: *1darray*
  ppm scale of the spectrum
- spectrum: *1darray*
  Spectrum of the mixture
- full_calc: *1darray*
  Spectrum of the initial guess, with all the peaks in total

Returns:

- regions: *list of tuple*
  Limits, in ppm
