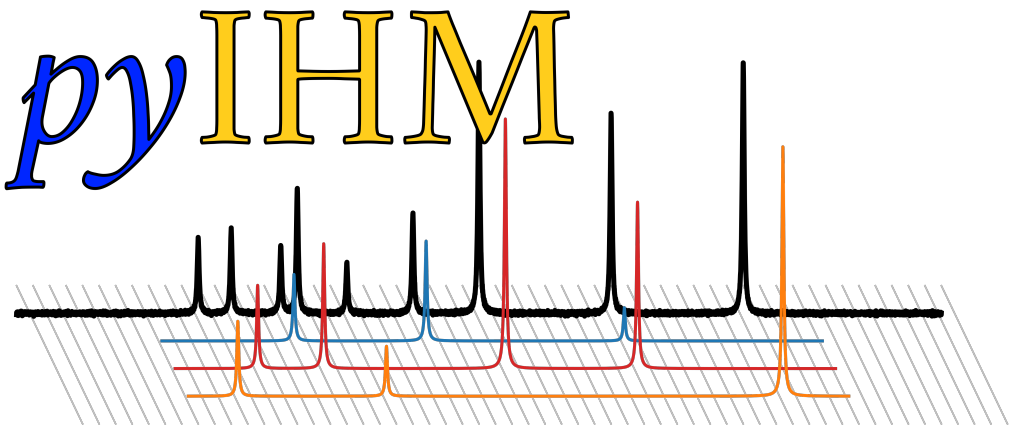

Indirect Hard Modelling, in Python

Author:
Francesco Bruno

Version: 0.1.0

Documentation release date:
October 9, 2023

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
```

2.2 Write the input file

Once installed, the software can be operated from the command line via typing:

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

where `<input_file>` is the path to the input file that contains the parameters for the run. Multiple input files can be given at once, writing their paths in sequence without punctuation signs between them.

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

The input file must be written in a plain text file. It consists in a series of keywords, followed by their arguments in the following line. The sections of the file, one per keyword, are separated by an empty line.

A template for the input file is shown in table 2.1. A detailed explanation of the keyword meanings and the syntax of the related parameters follows.

- `BASE_FILENAME`
  Root of the name of all files that the program will save.

- `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.

- `MIX_SPECTRUM_TXT`−optional
  Path to a plain text file that contains the intensity values of the real part of the spectrum. Useful to be set if the mixture spectrum was processed with anexternal software.

Table 2.1: Example of the input file, used to run the test located in the `test/` folder of the GitHub repository.

```
BASE_FILENAME
output/test

MIX_PATH
M.acqus, isexp=False

MIX_SPECTRUM_TXT
M.r

COMP_PATH
comp/C_1.fvf
comp/C_2.fvf
comp/C_3.fvf

FIT_LIMITS
10, 0

FIT_BDS
utol=0.2
utol_sg=0.01
stol=0.01
ktol=0.001

PLT_OPTS
ext=tiff, dpi=300
```

- `COMP_PATH`
  List of the `.fvf` files that contain the parameters of the signals of the components, generated by `KLASSEZ`. See section 2.3 for details. Write one file per row.

- `FIT_LIMITS`
  Limits of the fitting region, in ppm, separated by a comma.

- `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 $[\delta - \texttt{utol}, \delta + \texttt{utol}]$.
  - `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. They will vary in the interval $[\delta - \texttt{utol}, \delta + \texttt{utol}]$. Given the starting linewitdh Γ, in Hz, they will vary in the interval $[\Gamma - \texttt{stol} \cdot \Gamma, \Gamma + \texttt{stol} \cdot \Gamma]$.
  - `ktol`: tolerance for the relative intensities of the signals in the same spectrum. Given the starting intensity k, they will vary in the interval $[k - \texttt{ktol}, k + \texttt{ktol}]$.

- `FIT_KWS`−optional
  Additional parameters for `lmfit.Minimizer.minimize`, except the 'method' key, which is always set as 'nelder'.

- PLT_OPTS—optional
  Set specific resolution (dpi) and format (ext) for the figures that will be saved. The default values are ext='tiff', dpi=600.

2.3 Deconvolution of the spectral components

A script for the deconvolution of the spectra is provided here. It uses the KLASSEZ package to read, process and deconvolve the spectra. The script must be edited in order to fit the specific user's need.

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

import sys
import klassez as kz

filename = sys.argv[1] # Spectrum
spect = sys.argv[2]   # Format

# 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()
# Perform the fit...
S.F.dofit( # ...with the following options:
       u_tol=0.2,        # variation on chemical shift /ppm
       f_tol=2,          # variation of FWHM /Hz
       vary_phase=False, # Phase correction on the peaks
       vary_xg=False,    # Fraction of gaussianity
       )
# Save the figures for the fit
S.F.plot('result', show_res=True, res_offset=0.1)
```

At the end of the run, the .fvf file will be saved in the folder of the spectrum.

2.4 Reading the results

During the execution of pyIHM, a series of files will be generated. The .out file contains a summary of the result of the fit, i.e. the concentrations of the species of the mixture and the parameters of the individual signals.

Regarding the figures, the '_iguess' one diplays the initial guess for the fit, whereas the '_total' one shows the result. The '_wcomp' figure highlights the components with different colors. Finally, the '_rhist' figure is the histogram of the residuals, with a gaussian curve overlayed in red. The statistical parameters of the residuals (i.e. the mean and the standard deviation) are reported in the legend of this figure.

3. List of modules and functions

3.1 MODULE input_reading

3.1.1 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
- 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 (absolute)
  - utol_sg = allowed displacement for the peaks that are part of the same multiplet relatively to the center, in ppm (absolute)
  - stol = allowed variation for the linewidth, in Hz (relative)
  - ktol = allowed variation for the relative intensities within the same spectrum(relative)

3.1.2 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

3.2 MODULE spectra_reading

3.2.1 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

---

3.2.2 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

3.2.3 spectra_reading.main(M, spectra_dir)

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

Returns:

- collections: *list of Spectr objects*
  Spectra of pure components, treated as collections of peaks.

3.3 MODULE gen_param

3.3.1 gen_param.L2P(L, Xmin, Xmax)

Convert a normalized parameter into its absolute counterpart.

Parameters:

- L: *float*
  Normalized parameter value
- Xmin: *float*
  Lower bound of the 'original' patameter
- Xmax: *float*
  Upper bound of the 'original' patameter

Returns:

- name: *str*
  Label of the parameter
- value: *float*
  Correspondant value

3.3.2 gen_param.P2L(P)

Normalize a lmfit.Parameter object according to its boundaries. Works only if expr is not set! In this case, in fact, it returns None. The boundaries of the new parameter are set to be (0,1), where 0 corresponds to P.min and 1 to P.max.

Parameters:

- P: *lmfit.Parameter object*
  Not normalized parameter

Returns:

- L: *lmfit.Parameter object*
  Normalized parameter. If P.expr is set, this is None.

3.3.3 gen_param.as_par(name, value, lims=0, rel=True)

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.

Returns:

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

3.3.4 gen_param.main(M, components, bds)

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.

Returns:

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

3.3.5 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

3.3.6 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

3.4 MODULE fit_mixture

3.4.1 fit_mixture.L2P(L, Xmin, Xmax)

Convert a normalized parameter into its absolute counterpart.

Parameters:

- L: *float*
  Normalized parameter value
- Xmin: *float*
  Lower bound of the 'original' patameter
- Xmax: *float*
  Upper bound of the 'original' patameter

Returns:

- name: *str*
  Label of the parameter
- value: *float*
  Correspondant value

3.4.2 fit_mixture.calc_spectra(Lparam, 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:

- Lparam: *lmfit.Parameters object*
  Normalized 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

---

3.4.3 fit_mixture.calc_spectra_obj(Lparam, 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:

- Lparam: *lmfit.Parameters object*
  Normalized 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

---

3.4.4 fit_mixture.f2min(Lparam, param, N_spectra, acqus, N, exp, I, plims)

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

Parameters:

- Lparam: *lmfit.Parameters object*
  Normalized 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

Returns:

- target: *float*
  Given the experimental data y and the calculated data y_c: $\sum\left[(y - I * y_c)^2\right]$

3.4.5 fit_mixture.main(M, components, bds)

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.

Returns:

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

3.4.6 fit_mixture.main(M, N_spectra, Lparam, param, lims=None, filename='fit', 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
- Lparam: *lmfit.Parameters object*
  Normalized parameters
- param: *lmfit.Parameters object*
  Actual parameters
- lims: *tuple or None*
  Delimiters of the fitting region, in ppm. If None, the whole spectrum is used.
- filename: *str*
  Root of the names for the names of the files that will be saved.
- ext: *str*
  Format of the figures
- dpi: *int*
  Resolution of the figures, in dots per inches

3.4.7 fit_mixture.write_output(M, I, K, spectra, lims, filename='fit.report')

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 intensities of the spectra in the mixture
- lims: *tuple*
  Boundaries of the fit region
- filename: *str*
  Name of the file where to write the files.

3.5 MODULE plots

3.5.1 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

3.5.2 plots.plot_iguess(ppm_scale, exp, total, components, lims=None, X_label='δ /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.
- 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

3.5.3 plots.plot_output(ppm_scale, exp, total, components, lims=None, X_label='δ /ppm', filename='fit', 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:

- 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.
- 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
