UNIONE EUROPEA
Fondo Sociale Europeo

PON RICERCA E INNOVAZIONE

UNIVERSITÀ DEGLI STUDI FIRENZE
Da un secolo, oltre.

DOTTORATO DI RICERCA IN
International Doctorate in Structural Biology

CICLO XXXVII - PON ex DM1061

Automatized Quantitative NMR for Industrial Applications

Settore Scientifico Disciplinare
CHEM-03/a

Dottorando/a
Dott. Bruno Francesco

Supervisore
Prof. Ravera Enrico

Coordinatore
Prof.ssa Pierattelli Roberta

This thesis has been approved by the University of Florence, the University of Frankfurt and the Utrecht University

Universiteit Utrecht

"There's no shame in admitting what you don't know.
The only shame is pretending you know all the answers."

– Neil deGrasse Tyson

A Letizia

Contents

List of Figures

List of Tables

List of Listings

1. Introduction

Nuclear Magnetic Resonance (NMR) spectroscopy is currently a wide-spread technology both in academic and industrial environments. The most invaluable feature of NMR spectroscopy is the atomistic view that it gives on the system under investigation. The peak positions and their appearances are a consequence of the local magnetic field generated by the electron distribution near the nucleus associated with the signal. For this reason, NMR has an exquisite qualitative power, as each substance gives rise to a unique spectrum. Furthermore, this comes along with an intrinsic quantitativeness, as the intensity of each peak is only proportional to the number of equivalent nuclei that contributes to it. It is thus not a surprise that the applications of NMR spectroscopy as analytical tool, in particular for the industry, are primarily towards quantification of mixtures.

NMR observes nuclear spin transitions in a magnetic field. Since the introduction of FT-NMR, the experiment relies upon the manipulation of nuclear populations and coherences by the application of radio-frequency pulses with appropriate frequency, which are used to push the population of the spin states off-equilibrium, up to the actual detection, where an oscillating current is induced in the receiver coil. This induced current, the Free-Induction Decay (FID), carries all structural and dynamical information on the system.

However, the outstanding features in terms of qualitativeness and quantitativeness of the analysis comes at a very high price. Going back to the physical backgrounds of the technique, the energy of the nuclear spin states is determined by the Zeeman interaction (equation 1.1), and they are populated according to the Boltzmann distribution (equation 1.2):

$$\hat{\mathcal{H}}^{\text{Zeeman}} = \hbar \gamma_I B_0 \tag{1.1}$$

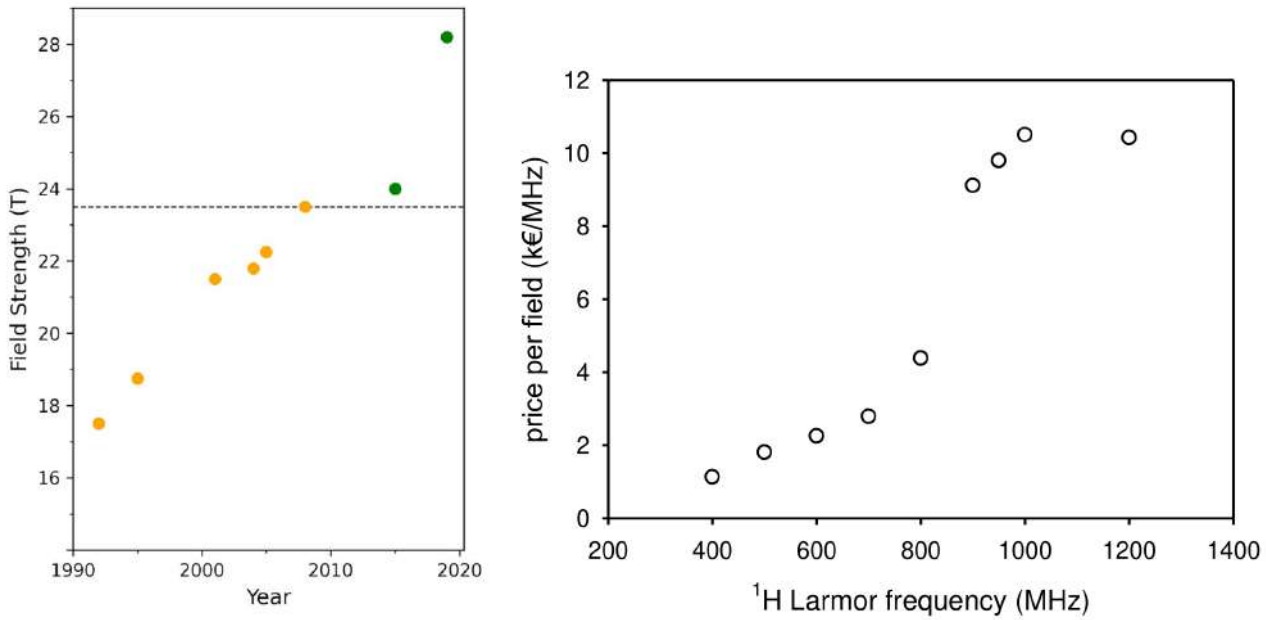
$$P_j = \frac{e^{E_j/k_B T}}{\sum_j e^{E_j/k_B T}} \tag{1.2}$$

Considering the example of ^{1}H, which has the highest gyromagnetic ratio among the common magnetically-active nuclei ($\gamma_{^1\text{H}} = 42.5\,\text{MHzT}^{-1}$), at room temperature only one transition out of approximately 10000 is actually observed. This is the reason of the infamously poor sensitivity of NMR spectroscopy.

There are two straightforward solutions to the sensitivity problem. One is to increase the static magnetic field B_0, so that to increase the energy separation of the nuclear spin states. This lead to the development of increasingly powerful superconducting magnets, from the nowadays common low-temperature type II superconductors (e.g. niobium-titanium alloys (Nb−Ti) or niobium-tin (Nb_3Sn)), up to the newest generation dual high-temperature/low-temperature superconducting magnets that can operate at even 28.2 T (that correspond to 1.2 GHz ^{1}H Larmor frequency). Unsurprisingly, the higher the magnetic field and the more expensive is the spectrometer (see figure 1.1).

The other approach would be to lower the temperature of the electronics used for the acquisition, by using dedicated cryo-probes and dedicated hardware. Unfortunately, this kind of instrumentations is more expensive (about three times a regular system), and it is also much more delicate.

Finally one can also cool the sample down to (e.g.) liquid nitrogen or liquid helium temperatures, which increases the population of the nuclear spin states, and thus the signal intensity. However, this also comes with an attached non-negligible price tag: for one day of measurements at 80 K, about 200 dm^3 of liquid nitrogen are needed, at an estimated cost of 800 €. To work at 5 K, an even larger amount of liquid helium is necessary, for an estimated cost that can easily exceed 5000 € per day. The cryogenics consumption is not the only issue in this case: the probes must have a very good cryostatization, and the sample must be properly prepared to avoid the formation of ice crystals, which can cause sizeable field inhomogeneities in the sample. These outlays come on top of the set-up of a dedicated space for the spectrometer, which is often not trivial, and of routine cryogenics consumption to preserve the superconductivity of the magnet. Such an investment is often not economically compatible for low- and medium-level academic and industrial settings. Although there exist several initiatives at national and international level to allow user-access to large-scale academic facilities also for industrial users, such as Instruct-ERIC[1] (a pan-European distributed research infrastructure that makes cutting-edge structural biology tools accessible to researchers), an easier and more approachable alternative would be beneficial.



(a) **(b)**

Figure 1.1: **(a)** Advancements in magnetic field strengths for NMR over the last few decades. Yellow indicates low temperature magnets, whereas the hybrid high-temperature/low-temperature magnets are marked in green. A dashed line is drawn in correspondence of 1 GHz ^{1}H Larmor frequency (23.5 T). **(b)** Trend of the price of commercial NMR spectrometers using superconducting magnet systems normalized by the field, at the time of market introduction of the 1.2 GHz instrument[2].

Such a more economical alternative came in the last decade in the form of benchtop NMR spectrometers. These instruments are more compact, at the point that they can be installed on a normal lab desk, and they employ permanent magnet up to about 2 T (80−120 MHz ^{1}H Larmor frequency) for the build-up of the magnetization. The presence of an external lock system and dedicated add-ons for in-flow measurements make them useful for routine applications, despite

the low resolution of the acquired spectra.

The resolution of a spectrum can actually represent a problem also for high-field NMR. Generally speaking, spectroscopic resolution of an NMR spectrometer is the minimum separation in frequency that allows for distinguishing two neighboring peaks, and is estimated as twice the full-width at half maximum (FWHM) of the signal. This concept is graphically presented in figure 1.2. In case of complicated sample (e.g. proteins) or complex mixtures (e.g. biological fluids), the peak overlap can be so severe to prevent standard analyses to be performed. In such dramatic cases, the physical bases of FT-NMR come to a rescue. The spectroscopist can select pulse sequences that can act as a filter on the huge amount of information encoded in the FID, thus making the interpretation of the results much easier. Examples of such experiments include the 1D-filtered-NOESY or CPMG, or multidimensional NMR experiments. The latter option allows to distribute the signals on an additional frequency dimension, thus providing the desired resolution increase and a further source of information on the system[3]. However, this comes at the cost of an increased acquisition time: as the whole time-domain information in each transient of an nD experiment depends on a single indirect time evolution, the indirect dimensions grow at a much slower pace than the former. This means that it is not uncommon that it takes several hours, or even days, to record a single 2D NMR spectrum.

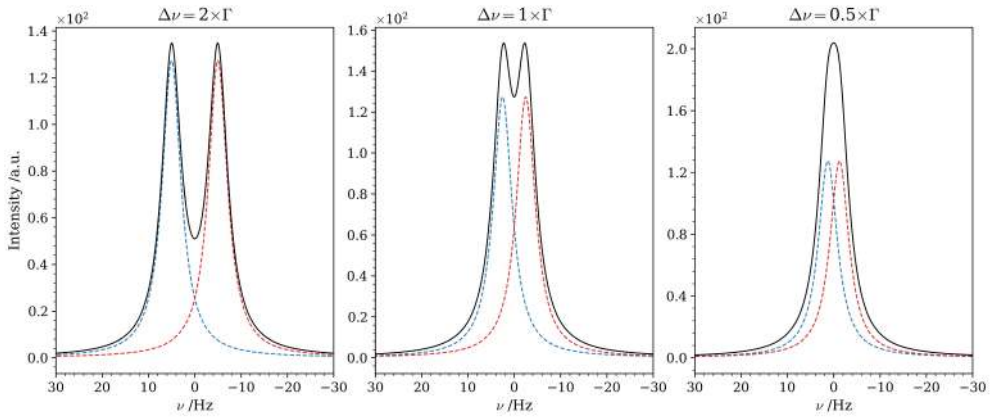


Figure 1.2: Two Lorentzian signals were simulated with same intensity, equal to 1, and same full-width at half maximum $\Gamma = 5\,\text{Hz}$. The resulting peak that would appear in the spectrum is depicted as black trace. The frequency difference between the resonances of the two peaks was set to twice the linewidth (left panel), equal to the linewidth (central panel), and half the linewidth (right panel). The trend shows that the two perfectly distinguishable features start to coalesce, until they appear as a single, broader peak.

However, there exist acquisition methods that allow for speeding up the construction of the indirect dimension. One of the ideas is to use shaped pulses to excite only a restricted frequency range, namely the one that contains the interested resonances (Band-selective Excitation Short Transient, BEST), which reduces the relaxation time of the selected resonances because of cross-relaxation with the non-excited resonances. This is more efficient the larger the molecule[4]. A further improvement can be obtained by adding a careful tuning of the excitation pulse, to maximize the balance between the flip angle and the required recovery delay (Band-Selective Optimized Flip-Angle Short-Transient, SOFAST)[5]. It is also possible to shorten the measurement time by acquiring only a part of the indirect evolution timescale, and reconstruct the missing points through the use of dedicated algorithm in the post-acquisition phase (non-uniform sampling, NUS[6]). More exotic solutions, that deviate from the standard acquisition strategies, include the use of gradients to record the entire indirect dimension in a single scan (Ultrafast NMR[7]). This latter solution comes at the price of sizeably reduced sensitivity because the sample is sliced to obtain the indirect dimension.

Regardless the actual method employed to record the NMR data, the fact that the information is encoded in a time-domain series makes the interpretation of the results a rather challenging and unpractical task. As the structural and dynamical information on the system are physically reflected in the energy of spin transitions, and considering that "energy" means "frequency" in spectroscopy, it is clear that trying to find a specific frequency from the observation of a complicated medley of time-decaying plane waves is not very convenient.

In fact, unlike with most analytical techniques, the path from acquisition to analysis in NMR is not direct nor straightforward. It might as well be regarded as the most critical outstanding obstacle to an even wider diffusion of the technique.

This thesis work is focused on the post-acquisition treatment of the data. The sequence of operations carried out to transform the FID in an interpretable spectrum is commonly referred to as processing. A necessity-aware processing is of utmost importance to increase the quality of the data. A routine processing pipeline usually involves the following steps:

1. *Choice of window function*
   The FID is multiplied by a function that acts like a weight for specific regions, thus giving more emphasis on specific features of the spectrum.

2. *Zero-filling*
   The end of the FID is padded with zeroes until a specific number of points is reached, with the aim of increasing the digital resolution of the spectrum and fulfil the power-of-two points that the Fast Fourier Transform algorithm requires[8].

3. *Fourier Transform*
   This is the step in which the time-domain FID is converted in the frequency-domain spectrum.

4. *Phase Correction*
   The complex NMR signal is the linear combination of an absorption-mode Lorentzian and a dispersion-mode Lorentzian. The aim of the phase correction is to confine the absorption-mode Lorentzian in the real part of the spectrum (i.e. the one that is usually displayed).

In addition to standard protocols, either the FID or the spectrum can be treated to increase the signal-to-noise ratio (SNR) of the spectrum. This approach is known as denoising. A denoising technique is considered good if it increases the SNR while preserving the position, the relative intensity, and possibly the peak shape of each signal. Among the available options for mono-dimensional datasets, there are the Cadzow filter[9], wavelet transform[10], Savitzky-Golay filters[11], and random QR denoising[12]. Although not being a denoising method *per se*, Reference Deconvolution[13] aims to correct distorted lineshapes of the signals, thus increasing the quality of the spectrum in a lateral sense. For two-dimensional datasets, which can be either true 2D experiments or series of 1D spectra, denoising approaches based on low-rank decompositions[14], such as SVD or Multivariate Curve Resolutions (MCR)[15], become viable options. This topic is addressed in detail in chapter 4.

The processing of the data is normally carried out using the routines encoded in the spectrometer software. The advantage of a closed-box program is the ease of use: as it is the same software used for the acquisition, it knows exactly how to handle the different data format. The problem arises when the spectroscopist wants to perform some kind of processing that is not already implemented in the software. In this case, there are open-source alternatives that implement the processing routines[16,17,18]. To use these packages, the spectroscopist must have a detailed knowledge of how the processing exactly works, and deal with several technical issues

in the conversion of the datasets. Furthermore, the learning curve of such programs is quite steep, often because of the lack of very detailed user manuals and examples.

In addition to the challenges posed by the processing, also the analysis of the data is a difficult task. As previously mentioned, the intensity of an NMR signal is proportional to the number of equivalent nuclei that contribute to it. Therefore, quantitative analyses in NMR are traditionally performed by peak integration of signature peaks of the interested chemical species, by means of dedicated GUI. By adding an internal standard to the sample, i.e. an external substance in a given amount, it is possible to get the absolute concentrations of all the components of the mixture by means of a simple proportion. Such an intuitive and straightforward approach has its own limitations. The NMR signal in the frequency domain is described by a Lorentzian profile: the integral of the normalized Lorentzian function is 1 only if considered in the $(-\infty, +\infty)$ range, which of course is not possible in practical applications. To account for the 99% of the intensity, the integration range should be chosen $64\times$ the FWHM of the peak[19]. Fulfilling this requirement is a nontrivial task when dealing with complex mixtures, and it becomes even more difficult in low-field NMR, where the low resolution is the main disadvantage. Furthermore, baseline distortions and incorrect phasing of the spectrum can result either in unwanted offsets or negative portions of the signal, which further jeopardize the computation of the integral.

A rather obvious workaround to these complications would be to fit the signals of interest with an appropriate model, and extract the intensities from the fit. This approach is less sensitive to the aforementioned problems, and can be applied in cases with low SNR or significant peak overlap. However, in the case of mixture quantification, the optimization problem escalates rather quickly if one tries to fit the spectrum as a whole. The task can be simplified if one considers the spectrum of the mixture as a linear combination of the spectrum of the components, where the coefficients are the relative concentrations. This procedure is commonly known as Indirect Hard Modelling. Several implementations have been proposed and applied with quite good results[20,21].

The Indirect Hard Modelling approach can be also useful when dealing with non-static samples. In fact, NMR can be also used to study the variation of a property of the system due to the use of a particular pulse sequence, or to chemical changes occurring. The evolution of the system can be followed by the acquisition of series of spectra, which when assembled in a single FID give rise to a so-called pseudo-2D experiment. The word "pseudo" is referred to the fact that only one of the two temporal dimensions of the 2D FID encodes for a frequency, whereas the other one is the evolution time, and should not be Fourier-transformed. As such a dataset is composed by hundreds of spectra, the integration approach can be quite cumbersome, and when some of the signals drift, or their intensity falls under the detection limit, it might also fail completely. Hence, the deconvolution of the peaks represents the only valuable technique to address the analysis for an accurate evaluation.

All this is to say that data analysis in NMR spectroscopy is overall not an easy task. For most applications, a qualitative, visual inspection of the spectrum is not enough, and it would be very limiting with respect to the possibilities that the technique offers. Most basic analysis routines, such as the peak integration, can be easily performed with the same software used for the acquisition. However, more advanced and detailed analyses require dedicated softwares. Being specifically designed to perform only few tasks, these softwares lack the necessary flexibility to adapt exactly to the user's needs. Actually, it is often quite the contrary: it is the user that has to bend to the software, which is a paradigm that goes against the principle of science. This is the main reason why NMR spectroscopists develop their own programs and scripts. As a consequence, they have to face a series of compatibility issues, first of all how to feed the raw data to the script, and how to prepare it for the analysis. For this reason, we developed `KLASSEZ`

(see section 8.1.1), an open-source python package that supports the analyst with reading and processing the FID. In order to make it universal, it supports data formats coming from several spectrometers brands, such as Bruker, Jeol and Varian, as well as benchtop spectrometers like Magritek and Oxford Instruments.

Throughout this thesis work, we will present the main features of `KLASSEZ` , and several examples of their application. Each chapter of the document puts its focus on a particular category of the data-handling. In chapter 2, the mathematical structure of the NMR signal and noise are discussed, in order to provide methods to simulate these features. Chapter 3 illustrates the steps to perform in order to process the data, to get an interpretable spectrum. In particular, chapter 4 presents several strategies to increase the quality of the data in the post-acquisition stage. Then, specific methods on how to perform data analysis follow. Chapter 5 deals about the fitting of the spectra in order to gather information on the spectral parameters. Finally, in chapter 6 a few approaches to exploit the intrinsic quantitativeness of NMR spectroscopy are addressed. Each section comes with example scripts, that illustrate how to employ `KLASSEZ` to perform the described tasks.

2. Models for the simulation of NMR data

Simulating the spectra is an extremely valuable tool for testing implemented routines against reference data. This is important, in particular, to ensure the reliability of NMR data processing, and to compare the outcomes with other softwares. For this reason, a section of `KLASSEZ` is dedicated to the simulation of datasets starting from user-defined values. I implemented the methods described herein in `KLASSEZ` as illustrated in the enclosed listings.

2.1 1D Spectra

The hard model dictated by the physics of the FT-NMR experiment is a complex, oscillating and exponentially decaying signal in the time domain, which is then Fourier transformed to obtain a Lorentzian lineshape. An FID encoding for N signals is the sum of the FIDs of the individual signals:

$$s(t) = \sum_{k=0}^{N-1} s_k(t) \tag{2.1}$$

However, experimental practice teaches us that the NMR signal is never a pure Lorentzian. The reason for such a deviation comes from a number of factors. The assumption that the magnetic field is perfectly homogeneous throughout the whole sample is often not fulfilled. Each nucleus would hence experience a slightly different Larmor frequency, which induces a variability in the chemical shift. The resulting effect is the so-called heterogeneous broadening, which in the easiest case introduces a Gaussian component to the lineshape due to the random distribution of the field values. Intuitively, this fraction becomes more and more important as the field inhomogeneity increase, up to the point that the NMR peak can be modelled with a pure Gaussian function, as it often happens in solid-state NMR. In this case, the field inhomogeneity is often due to the size distribution of the particles that compose the sample, which reflects in a distribution of the magnetic susceptibility values and shapes for the different particles. The other main source of deviation is a poor shimming of the instrument. This leads the magnetic field profile to deviate from a Gaussian distribution, with the effect of very oddly-shaped signals (see figure 2.1). Other sources of lineshape distortion can arise from the non ideality of the other spectrometer components.

Several models have been proposed to describe spectral lineshapes beyond the pure Lorentzian, the most popular of which are the Voigt (equation 2.2) and the pseudo-Voigt (equation 2.3), which are the convolution and the linear combination of a Lorentzian and a Gaussian profile, respectively. The syntax to call the `KLASSEZ` implementations of equation 2.3 and 2.2 is reported in listing 2.1. These two models compensate from slight deviations from the ideal Lorentzian lineshape by assigning a different fraction of gaussianity for each signal, without attributing any physical meaning to the actual source of the distortion.

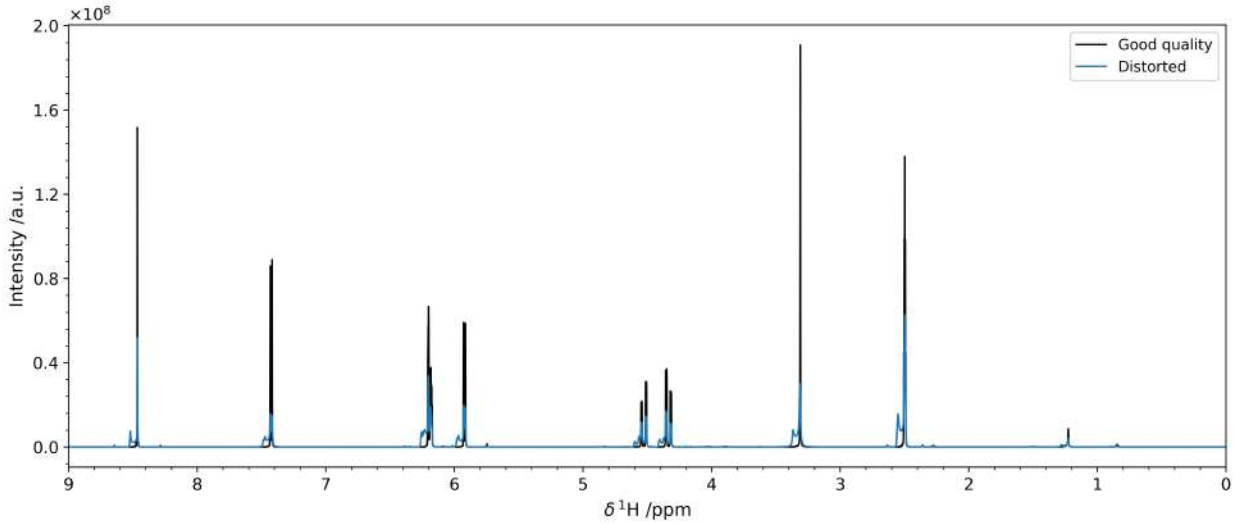


Figure 2.1: The figure shows a ^{1}H-NMR spectrum of Patulin (4-hydroxy-4H-furo[3,2-c]pyran-2(6H)-one) at 500 MHz, acquired with a poor shimming of the magnet (blue trace). As it is apparent from the comparison with the properly-shimmed spectrum, the signals do not only show an asymmetric and quite strange distortion, but also the SNR is dramatically decreased.

$$s^{\text{Voigt}}(t) = K\ \exp\{\mathrm{i}\omega t\}\ \exp\{-(1-\beta)\Gamma t/2\}\ \exp\left\{-\beta\sigma^2 t^2/2\right\} \qquad \sigma = \frac{\Gamma}{2\sqrt{2\ln 2}} \tag{2.2}$$

$$s^{\text{pseudo-Voigt}}(t) = K\ \exp\{\mathrm{i}\omega t\}\left[(1-\beta)\exp\{-\Gamma t/2\} + \beta\exp\left\{-\sigma^2 t^2/2\right\}\right] \qquad \sigma = \frac{\Gamma}{2\sqrt{2\ln 2}} \tag{2.3}$$

In both models, $K = I\exp\{\mathrm{i}\phi\}$ contains information on the intensity and the phase of the peak, Γ is the full-width at half-maximum (FWHM) in $\mathrm{rad\,s^{-1}}$, and $\beta \in [0,1]$ represents the fraction of gaussianity that contribute to the lineshape of the signal. In this manner, they generate pure Lorentzian peaks for $\beta = 0$ and pure Gaussian peaks for $\beta = 1$. The frequency $\omega = 2\pi\nu$ is to be intended as the angular precession of the spin in the rotating frame, and therefore is the relative frequency with respect to the carrier. In this way, given the reference chemical shift δ_0 (i.e. the carrier position, in ppm, in a real NMR experiment), the conversion of this frequency to the chemical shift is straightforward:

$$\delta = \frac{\omega}{2\pi\gamma_I B_0} + \delta_0 \tag{2.4}$$

A visual interpretation of the parameters is given in figure 2.2.

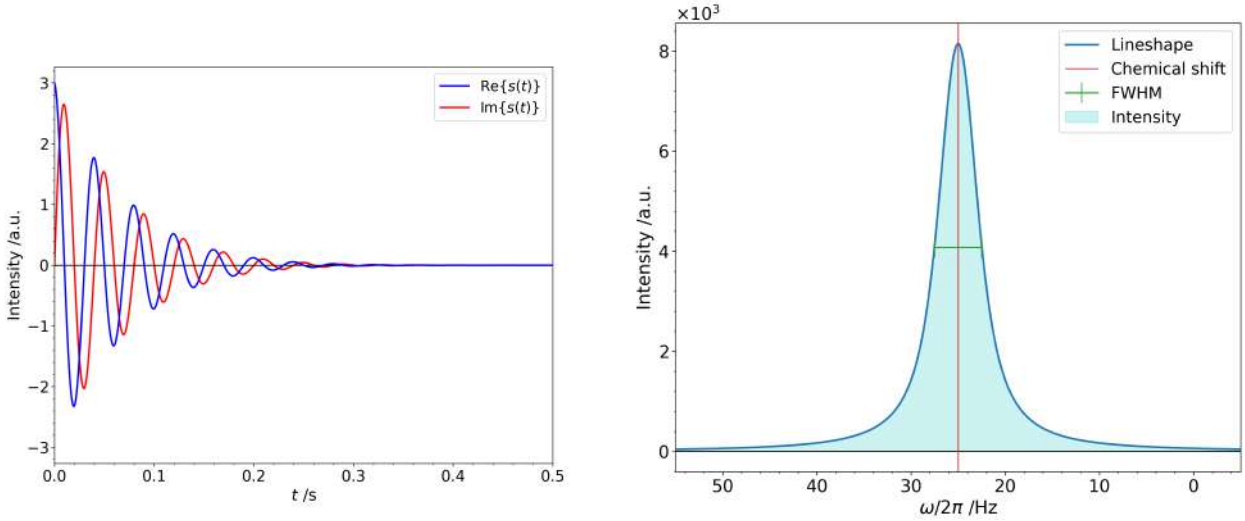
For our purposes, it is convenient to compute these functions in the time domain. This choice allows us to mimic the behavior of the NMR spectroscopy, i.e. we generate the FID of the signals, not directly the spectrum in the frequency domain. Furthermore, this approach delegates the ominous task of performing the convolution in the Voigt model to the Fourier transform, by exploiting the Convolution Theorem.

The generation of the simulated FID is implemented in the function `sim.sim_1D` of `KLASSEZ` . The user has to write a simulation input file structured in a key-value fashion. The complete list of information to provide is reported in table 2.1. An example of such input file is shown in listing 2.2. The simulated spectrum with this input, after Fourier transform, is shown in figure 2.3.

Listing 2.1: Pseudo-Voigt (function `t_pvoigt`) and Voigt models (function`t_voigt`), from the `sim` module of KLASSEZ .

```
def t_pvoigt(t, u, fwhm, A=1, b=0, phi=0):
    """
    Pseudo-Voigt function in the time domain:
    --------
    Parameters:
    - t: 1darray
        Independent variable
    - u: float
        Peak position, in Hz
    - fwhm: float
        Full-width at half-maximum, in rad/s
    - A: float
        Intensity
    - b: float
        Fraction of gaussianity
    - phi: float
        Phase, in radians
    --------
    Returns:
    - S: 1darray
        Pseudo-Voigt function.
    """
```

```
def t_voigt(t, u, fwhm, A=1, b=0, phi=0):
    """
    Voigt function in the time domain. The parameter b affects the linewidth of the lorentzian and gaussian
         contributions.
    --------
    Parameters:
    - t: 1darray
        Independent variable
    - u: float
        Peak position, in Hz
    - fwhm: float
        Full-width at half-maximum, in rad/s
    - A: float
        Intensity
    - b: float
        Fraction of gaussianity
    - phi: float
        Phase, in radians
    --------
    Returns:
    - S: 1darray
        Voigt function.
    """
```



(a) FID of the signal.

(b) Fourier-transformed FID.

Figure 2.2: Voigt model generated according to equation 2.2, (a) in the time domain and (b) in the frequency domain (only real part displayed). The used parameters were: $K = 3$, $\nu = 25\,\mathrm{Hz}$, $\Gamma/2\pi = 5\,\mathrm{Hz}$, $\beta = 0.2$.

Listing 2.2: Input file for the simulation of the NMR spectrum of α-glucose.

```
# Alpha-glucose
B0      14.12
nuc     1H
SWp     20
o1p     4.7
TD      2**16

shifts     5.214, 3.516, 3.696, 3.393, 3.817, 3.823, 3.745
amplitudes [1 for w in range(7)]
fwhm       [1 for w in range(7)]
b          [0 for w in range(7)]
mult       d, dd, dd, dd, ddd, dd, dd
Jconst     3.80, [3.80, 9.84], [9.84, 9.17], [9.17, 9.99], [9.99, 2.31, 5.40], [2.31, -12.37], [5.4, -12.37]
```

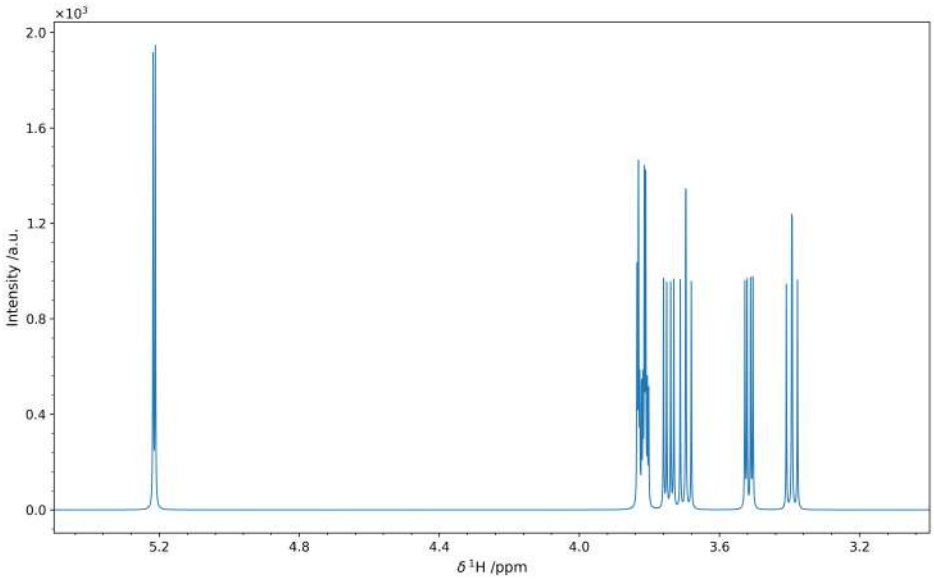


Figure 2.3: Simulated spectrum of α-glucose, using the input file in listing 2.2.

Table 2.1: List of keys and values that the input file for simulation of 1D spectra must contain. Optional parameters are marked with a *.

Key	Description
`B0`	Magnetic field strength, in T.
`nuc`	Observed nucleus.
`o1p`	Carrier frequency i.e. center of the spectrum, in ppm.
`SWp`	Spectral width, in ppm.
`TD`	Number of sampled complex points.
`shifts`	List of the chemical shifts at which the peaks have to appear.
`fwhm`	List of linewidths for each peak, in Hz.
`amplitudes`	Intensities of the signals.
`b`	Fraction of gaussianity for each peak $\in [0, 1]$.
`phases`*	Phase of each signal, in degrees. If not given, all signals are assumed to have phase equal to zero.
`mult`*	Multiplicity for each signal (s, d, dd, etc.). If not given, all signals are considered as singlets.
`Jconst`*	Scalar coupling constants for each peak and for each splitting, in Hz.

2.2 2D Spectra

NMR spectroscopy offers the possibility to acquire multidimensional experiments in order to improve the resolution, to focus on particular couplings occurring in the sample, and to select specific signals, with the aim of reducing spectral complexity. As in the mono-dimensional case, our model is designed to be computed in the time domain, as it resembles the way experimental spectra are actually recorded, and allows for the disentanglement of the information on the frequency and on the linewidth.

Nevertheless, a straightforward operation like multiplying two monodimensional Voigt functions does not work, because the Fourier transform in the indirect dimension would make the final peak to appear twisted and doubled with respect to the zero-frequency line in the indirect dimension. The reason for this behavior is the way how the evolution of the indirect dimension is actually recorded. Bidimensional NMR spectra are acquired using two time dimensions: during t_1 (indirect dimension), a certain property of the magnetization is evolved, whereas the actual FID is sampled during t_2 (direct dimension), and contains the modulated information. This means that 2D NMR datasets essentially are series of 1D spectra, acquired each one with a different value of t_1, stacked together. The quadrature detection is responsible for the frequency discrimination of the signals in the direct dimension, but in the indirect dimension this cannot be done. As a consequence, the signal will be only cosine-modulated in the indirect dimension, which will make its Fourier transform mirrored with respect to the zero-frequency axis. In order to achieve frequency discrimination also in the indirect dimension, it is necessary to implement a sort of "sampling scheme".

The methods encoded in 2D pulse sequences can be classified as phase-encoding (States-TPPI, Echo-Antiecho) and non-phase-encoding (QF, States, TPPI)[22]. The Echo-Antiecho method uses gradients to achieve the frequency discrimination in the indirect dimension, which is a situation that cannot be replicated easily in a simulation environment. Therefore we decided to implement the States-TPPI method. It must be noted, however, that it is possible to convert an Echo-Antiecho FID in the States-TPPI format (see section 3.7.3).

According to the States-TPPI scheme, the signal must be cosine-modulated in the indirect dimension. To achieve the correct frequency discrimination, the indirect timescale is modified so to acquire two subsequent transient with the same t_1 value, obtaining the effect of modulating the frequency both in cosine and in sine. Then, the receiver phase should be advanced of 90° for each scan. However, since in the simulation there is not an actual receiver that samples the data, the phase shift must be described from the signal perspective: to replicate the same situation encountered in acquisition, the signal would see the receiver to phase-shift counterclockwise. The comparison between the States-TPPI method used for the acquisition and for the simulation are summarized in table 2.2.

Table 2.2: t_1 sampling and phase shifts to be set in the States-TPPI scheme for the frequency discrimination in the indirect dimension. This four-step cycle must be repeated for all t_1 increments.

States-TPPI			**Simulation**		
t_1	ϕ_s	ϕ_{rec}	t_1	ϕ_s	ϕ_{rec}
t	0°	0°	t	0°	0°
t	0°	90°	t	270°	0°
$t + \Delta t$	0°	180°	$t + \Delta t$	180°	0°
$t + \Delta t$	0°	270°	$t + \Delta t$	90°	0°

Being $\mathbb{t}_1$ and $\mathbb{t}_2$ the arrays of the indirect evolution and acquisition timescale, respectively:

$$\mathbb{t}_1 = (0, \Delta t_1, 2\Delta t_1, \ldots, (N_1 - 1)\Delta t_1) \tag{2.5}$$
$$\mathbb{t}_2 = (0, \Delta t_2, 2\Delta t_2, \ldots, (N_2 - 1)\Delta t_2) \tag{2.6}$$
$$\tag{2.7}$$

we define the States timescale $\mathbb{t}_1^S$ as:

$$\mathbb{t}_1^S = (0, 0, 2\Delta t_1, 2\Delta t_1, \ldots, (N_1 - 2)\Delta t_1) \tag{2.8}$$

Considering a pure 2D Lorentzian signal for simplicity, the calculation of two arrays is required. For the direct dimension:

$$\mathbb{s}_2[k] = \exp\{\mathrm{i}\omega_2 \mathbb{t}_2[k]\} \exp\left\{-\frac{\Gamma_2 \mathbb{t}_2[k]}{2}\right\} \qquad \text{for } k = 0, \ldots, N_2 - 1 \tag{2.9}$$

For the indirect dimension:

$$\mathbb{s}_1[k] = \cos\left\{\omega_1 \mathbb{t}_1^S[k] - (k\%4)\frac{\pi}{2}\right\} \exp\left\{-\frac{\Gamma_1 \mathbb{t}_1^S[k]}{2}\right\} \qquad \text{for } k = 0, \ldots, N_1 - 1 \tag{2.10}$$

The whole 2D FID, $\mathbb{S}$, is obtained by multiplying the two arrays:

$$\mathbb{S} = \mathbb{s}_1 \, \mathbb{s}_2^T \tag{2.11}$$

As the lineshape of the resulting signal depends uniquely on the real exponential, to obtain the Voigt profile one has to replace

$$\exp\left\{-\frac{\Gamma \mathbb{t}[k]}{2}\right\}$$

with

$$\exp\left\{-(1-\beta)\frac{\Gamma \mathbb{t}}{2}\right\} \exp\left\{-\beta\frac{\sigma^2 \mathbb{t}^2}{2}\right\} \qquad \sigma = \frac{\Gamma}{2\sqrt{2\ln 2}} \tag{2.12}$$

with $\Gamma = \Gamma_1, \Gamma_2$ and $\mathbb{t} = \mathbb{t}_1^S, \mathbb{t}_2$ for the indirect and direct dimension respectively. The 2D Voigt signal is implemented in the function `t_2Dvoigt` (listing 2.3)

Following the same structure of the 1D datasets case, the simulation of 2D NMR FIDs is handled by the function `sim.sim_2D`. The rationale behind it is similar: the only difference is that information on two dimensions has to be provided. The complete list of information to provide is reported in table 2.3. An example of such input file is shown in listing 2.4. The simulated spectrum with this input, after Fourier transform, is shown in figure 2.4.

Listing 2.3: Function that implements the 2D Voigt model, from the `sim` module of `KLASSEZ` .

```
def t_2Dvoigt(t1, t2, v1, v2, fwhm1, fwhm2, A=1, b=0, states=True, alt=True):
    """
    Generates a 2D Voigt signal in the time domain.
    b states for the fraction of gaussianity, whereas A defines the overall amplitude of the total peak.
    Indexes '1' and '2' on the variables stand for 'F1' and 'F2', respectively.
    --------
    Parameters:
    - t1: 1darray
        Indirect evolution timescale
    - t2: 1darray
        Timescale of the direct dimension
    - v1: float
        Peak position in the indirect dimension, in Hz
    - v2: float
        Peak position in the direct dimension, in Hz
    - fwhm1: float
        Full-width at half maximum in the indirect dimension, in rad/s
    - fwhm2: float
        Full-width at half maximum in the direct dimension, in rad/s
    - A: float
        Intensity
    - b: float
        Fraction of gaussianity
    - states: bool
        Set to True for "FnMODE":"States-TPPI
    - alt: bool
        Set to True for "FnMODE":"States-TPPI
    --------
    Returns:
    - S: 2darray
        Voigt function.
    """
```

Listing 2.4: Input file for the simulation of a ^{1}H-^{15}N spectrum of five signals.

```
B0      16.4
nuc1    15N
nuc2    1H
o1p     115
o2p     5
SW1p    40
SW2p    20
TD1     512
TD2     2048

shifts_f1  100, 106, 120, 125, 130
shifts_f2  0, 2.5, 5, 7.5, 10
fwhm_f1    [50 for w in range(5)]
fwhm_f2    [50 for w in range(5)]
amplitudes [1 for w in range(5)]
b          0, 0.2, 0.4, 0.6, 0.8, 1
```

Table 2.3: List of keys and values that the input file for simulation of 2D spectra must contain.

Key	Description
`B0`	Magnetic field strength, in T.
`nuc1`	Observed nucleus in the indirect dimension.
`nuc2`	Observed nucleus in the direct dimension.
`o1p`	Carrier frequency i.e. center of the spectrum, in ppm, of the indirect dimension.
`o2p`	Carrier frequency i.e. center of the spectrum, in ppm, of the direct dimension.
`SW1p`	Spectral width, in ppm, of the indirect dimension.
`SW2p`	Spectral width, in ppm, of the indirect dimension.
`TD1`	Number of time evolution points sampled in the indirect dimension.
`TD2`	Number of sampled complex points in the direct dimension.
`shifts_f1`	List of the chemical shifts at which the peaks have to appear in the indirect dimension.
`shifts_f2`	List of the chemical shifts at which the peaks have to appear in the direct dimension.
`fwhm_f1`	List of linewidths for each peak, in Hz, in the indirect dimension.
`fwhm_f2`	List of linewidths for each peak, in Hz, in the direct dimension.
`amplitudes`	Intensities of the signals.
`b`	Fraction of gaussianity for each peak $\in [0, 1]$.

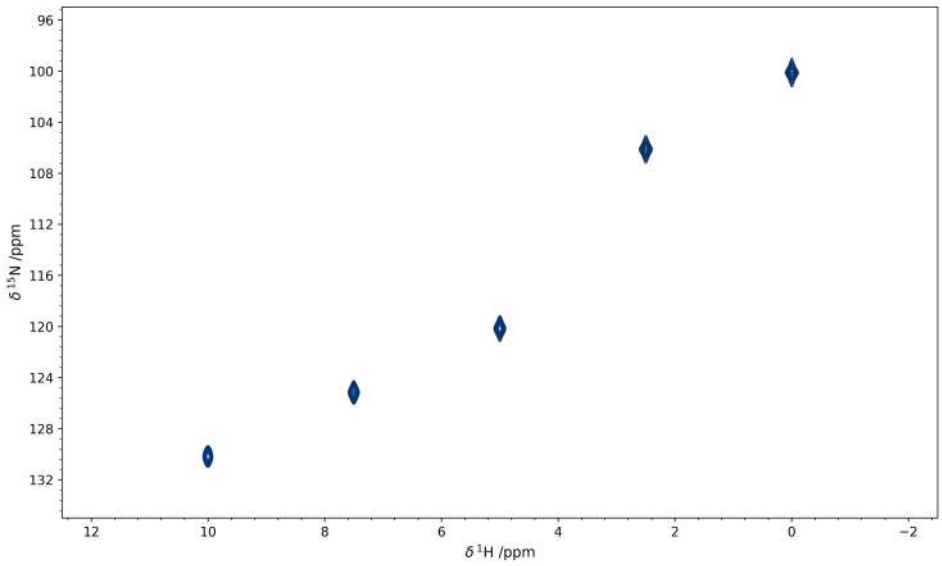


Figure 2.4: Simulated 2D spectrum composed of five signals of equal intensities and linewidths. The lineshape progressively goes from pure Lorentzian to pure Gaussian. The contour levels start from the 5% of the highest peak.

2.3 Noise in NMR spectra

In NMR, there are several noise sources, and each one generates noise with peculiar characteristics. The models presented herein are quite useful to understand the structure of the noise, and can be used to simulate noise in order to test (e.g.) denoising algorithms on simulated spectra.

2.3.1 Additive noise

The so-called additive noise is generally attributed to the electronics of the instrument, and therefore is ubiquitous in all NMR experiments. It can be modelled as a white noise (i.e. a set of random numbers whose mean is zero and gaussianly distributed with variance σ_N^2), modulated by a plane wave with frequency equal to the carrier frequency of the pulse[23]. We considered the additive noise as comprised of two parts: a correlated part, related to the fact that the acquired current travels the same cables until the ADC, and a non-correlated one, that accounts for the splitting of the signal into real and imaginary channel. Therefore:

$$\xi_+(t) = w_{\text{corr}} \exp\{\mathrm{i}\omega_p t\} + [w_{\text{Re}} \cos(\omega_p t) + \mathrm{i} w_{\text{Im}} \sin(\omega_p t)] \tag{2.13}$$

where ω_p is the carrier frequency of the pulse, and the ws are random samples extracted from a normal distribution centered at 0 with variance σ_N^2. Intuitively, the noisy FID can be obtained by summing this contribution to the noiseless FID.

$$\tilde{s}(t) = s(t) + \xi_+(t) \tag{2.14}$$

The `KLASSEZ` implementation is reported in listing 2.5.

Listing 2.5: Function that simulates additive noise, from the `sim` module of `KLASSEZ` . An example of application on an FID `fid` is also shown.

```
def noisegen(size, o2, t2, s_n=1):
    """
    Simulates additive noise in the time domain.
    --------
    Parameters:
    - size: int or tuple
        Dimension of the noise matrix
    - o2: float
        Carrier frequency, in Hz.
    - t2: 1darray
        Time scale of the last temporal dimension.
    - s_n: float
        Standard deviation of the noise.
    --------
    Returns:
    - noise: 2darray
        Noise matrix, of dimensions size.
    """
```

```
    fid += kz.sim.noisegen(fid.shape, carr_freq, t_aq, s_n=0.5)
```

2.3.2 Multiplicative noise

Multiplicative noise is often observed in 2D spectra as a consequence of the little field drifts between the acquisition of subsequent transients: for this reason, it is commonly known also as

t_1-noise. Other possible sources of t_1-noise can be variations of flip-angle and phase of the pulse over the course of the experiment, as well as the non-ideality of the receiver and fluctuations of the receiver-gain[24]. Its peculiarities are that it is proportional to the signal intensity and it does not reduce its impact by increasing the number of scans. According to the existing literature[25], the t_1-noise behaves like a modulation of the intensity of all signals in each transient, and therefore can be modelled as a diagonal matrix $\mathbb{X}$ which multiplies the noiseless data matrix, and whose diagonal entries are a function of a white noise vector $\mathbb{w}$ as stated in equation 2.15. Given a 2D FID $\mathbb{D}$ constituted by M transients of N points each, the correspondent "noisy" matrix $\tilde{\mathbb{D}}$ is given by:

$$\tilde{\mathbb{D}} = \mathbb{X}\mathbb{D}, \quad \mathbb{X}[i,j] := (1 - \mathbb{w}[i])\delta_{ij} \qquad \text{for } i,j = 0,\dots,M-1 \quad (2.15)$$

where $\mathbb{w}$ is an array of random variables as stated in the previous subsection. The `KLASSEZ` implementation is reported in listing 2.6.

Listing 2.6: Function that simulates multiplicative noise, from the `sim` module of `KLASSEZ` . An example of application on an FID `fid` is also shown.

```
def mult_noise(data_size, mean=0, s_n=0.1):
    """
    Multiplicative noise model.
    ---------
    Parameters:
    - data_size: tuple
        Dimension of the FID array
    - mean: float
        Mean of the random array distribution
    - s_n: float
        Standard deviation of the random array distribution
    --------
    Returns:
    - noisemat: 2darray
        Multiplicative noise array matrix
    """
```

```
    X = kz.sim.mult_noise(fid.shape, 0, 0.02)
    fid = X @ fid
```

2.3.3 Water artifacts

In 2D experiments, often the solvent signal (usually, water) appears as a ridge that spans all frequencies of the indirect dimension. The origin of this peculiarity is attributed to the residual off-axis magnetization, which is a consequence of the non-ideality of the spectrometer components and of the field inhomogeneity during the acquisition of the indirect dimension. Since the solvent normally gives rise to the most intense signal in the spectrum, the ridge can in principle hide relevant information that could be inferred by the interpretation of the spectrum. Theoretically, water does not give a cross-peak in ^{1}H-detected heteronuclear spectra, therefore the ridge can be modelled as a series of 1D FIDs expressed as in equation 2.2. The field inhomogeneity can be introduced by considering the solvent resonation frequency to vary around its theoretical value according to a Gaussian distribution, whose variance determines the width of the ridge. Finally, observing the experimental spectra, the signals constituting the ridge appear to be on-phase in even transients and 90°-dephased in the odd ones: this can be introduced in the model through the variable ϕ of equation 2.2. The resulting matrix $\mathbb{r}$, of the same dimensions of the data matrix $\mathbb{D}$ (i.e. $M \times N$), is given by the following expression:

$$\mathbb{r} = (\mathbb{r}_0| \dots |\mathbb{r}_{M-1})^T \quad (2.16)$$

Each of the $\mathbb{r}_k$ vectors is a Gaussian signal centered at the frequency ω_S (i.e. the relative frequency of the solvent signal) perturbed by a Gaussian random sample $\mathbb{w}[k]$ extracted from a distribution with mean equal to 1 and standard deviation σ_R, which represents the width of the ridge.

$$\mathbb{r}_k = \exp\left\{(k\%2)\mathrm{i}\frac{\pi}{2}\right\}\exp\left\{\mathrm{i}\mathbb{w}[k]\omega_S\mathbb{t}_2\right\}\exp\left\{-\frac{\sigma_S^2\mathbb{t}_2^2}{2}\right\} \qquad \text{for } k = 0,\ldots,M-1 \quad (2.17)$$

The `KLASSEZ` implementation is reported in listing 2.7. An example of the application of both multiplicative noise and ridge-like signals is shown in the left panel of figure 2.5.

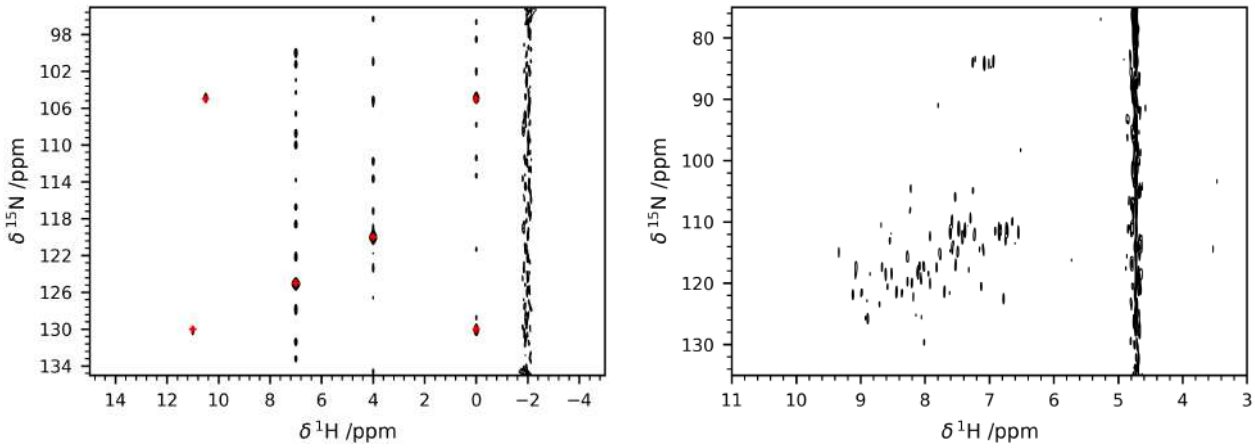


Figure 2.5: (**a**) Simulated HSQC-like spectrum with added multiplicative noise. The native peaks position are marked with red crosshairs on the figure. The signal-dependent nature of the simulated t_1-noise can be appreciated from the fact that the most intense peak (7 ppm ^{1}H, 125 ppm ^{15}N) is the most altered, whereas the least intense (11 ppm ^{1}H, 130 ppm ^{15}N) is left almost unperturbed. A simulated signal-ridge was added at −2 ppm, which can be compared with the one present at 4.7 ppm in an experimental ^{1}H-^{15}N HSQC spectrum of lysozyme at 1.2 GHz ^{1}H Larmor frequency(**b**).

Listing 2.7: Function that implements the solvent ridge signal, from the `sim` module of `KLASSEZ` .

```
def water7(N, t2, vW, fwhm=300, A=1, spread=701.125):
    """
    Simulates a feature like the water ridge in HSQC spectra, in the time domain.
    --------
    Parameters:
    - N: int
        Number of transients
    - t2: 1darray
        Time scale of the last temporal dimension.
    - vW: float
        Nominal peak position, in Hz.
    - fwhm: float
        Nominal full-width at half maximum of the peak, in rad/s.
    - A: float
        Signal intensity.
    - spread: float
        Standard deviation of the peak position distribution, in Hz.
    --------
    Returns:
    - ridge: 2darray
        Matrix of the ridge.
    """
```

3. Processing

As anticipated in the Introduction section, the path to transform the FID into an interpretable spectrum is not straightforward. A series of manipulation of both the time-domain and frequency-domain data are responsible for the final appearance of the spectrum, and might be a crucial factor to the achievement of an accurate analysis. As the theoretical backgrounds of these techniques are solidly grounded in NMR routine application, in this section I will present just a brief description, focusing on the actual implementation of such methods. I implemented the functions in `KLASSEZ` under the module `processing`.

3.1 Orthogonalization of the FID channels

The quadrature detection is obtained unshuffling the signal coming out of the coil and send it through two independent RF channels. If the channels are slightly unbalanced, the intensities of the two signals might be different, which results in non-orthogonality of the real and imaginary channels of the FID. Visually speaking, the FID shows an offset, or, in the most severe cases, a slowly-oscillating component. In the spectrum, this offset translates in the arise of the so-called DC component of the Fourier transform, i.e. an artifact that appears at zero-frequency (which corresponds with the position of the carrier). The intensity and shape of such a signal depends on the deviation from zero-mean of the two channels of the FID. Although mild distortions usually do not represent a problem, as they are hidden by the solvent resonance, more apparent ones can harm the interpretation of the spectrum by affecting the lineshape and intensity of the neighboring signals.

The proposed solutions to this problem aim to subtract a "baseline" from the distorted FID, thus recovering the native one:

$$\mathbb{s} = \tilde{\mathbb{s}} - \mathbb{q} \tag{3.1}$$

The algorithm named *quad*, which is best suited for modelling offsets, computes the baseline from the mean of its last quarter (equation 3.2).

$$\mathbb{q} = \mathbb{1}\left(\frac{4}{N}\sum_{k=\frac{3}{4}N}^{N-1}\tilde{\mathbb{s}}[k]\right) \tag{3.2}$$

When this is not enough, the slow-oscillating components of the FID are fitted with a polynomion of rank r (normally, $r = 5$) in the least-squares sense through the *qpol* algorithm (equation 3.3). For more details about polynomion fitting, see appendix A5.

$$\mathbb{q} = \sum_{i=0}^{r-1}\hat{\mathbb{c}}[i]\mathbb{t}^i \text{ where } \hat{\mathbb{c}} = \min_{\mathbb{c}}\left\|\tilde{\mathbb{s}} - \sum_{i=0}^{r-1}\mathbb{c}[i]\mathbb{t}^i\right\|_2^2 \tag{3.3}$$

The implementations in `KLASSEZ` for these two methods are shown in listing 3.1.

Listing 3.1: Functions that implement the `quad` and *qpol* algorithms, from the `processing` module of `KLASSEZ` .

```
def quad(fid):
    """
    Subtracts from the FID the arithmetic mean of its last quarter. The real and imaginary channels are treated
        separately.
    -------
    Parameters:
    - fid : ndarray
        Self-explanatory.
    -------
    Returns:
    - fid : ndarray
        Processed FID.
    """
```

```
def qpol(fid):
    """
    Fits the FID with a 4-th degree polynomion, then subtracts it from the original FID. The real and imaginary
        channels are treated separately.
    -------
    Parameters:
    - fid : ndarray
        Self-explanatory.
    -------
    Returns:
    - fid_corr : ndarray
        Processed FID
    """
```

3.2 Window functions

The process of modifying the data before the Fourier transform in order to improve the quality of the spectrum is called *apodization*, or *windowing*.

In general, the apodization process consists in the element-wise multiplication of the native FID $\mathbb{s}(t)$ by a smooth, real function $\mathbb{a}(t)$ (equation 3.4). Since the window function $\mathbb{a}$ is real, it leaves the phases and the frequencies unaltered, and only affects the linewidths and the lineshapes. The resulting spectrum, as a consequence of the convolution theorem, is the convolution between the native spectrum and the Fourier transform of the window function itself.

$$\mathbb{s}^{\text{apod}} = \mathbb{a} \odot \mathbb{s} \tag{3.4}$$

As it is not possible to enhance all the features of the FID at once, a huge plethora of window functions have been developed and are nowadays available. Their usage depends from case to case.

3.2.1 Box function

It may happen that the FID is sampled longer than the complete decay of the sharpest signal. In this case, the latter part of the FID contributes to the spectrum only with noise. It is therefore useful to trim the FID up to this point, using a box function. Being $t_{\max}$ the selected breakpoint, the box function to be used for apodization is defined as:

$$a^{\text{box}}(t|t_{\max}) = \begin{cases} 1 & \text{for } t \in [0, t_{\max}] \\ 0 & \text{for } t \in (t_{\max}, t_{\text{aq}}] \end{cases} \tag{3.5}$$

As shown in figure 3.1, the use of this window function can be greatly effective in reducing the noise levels of the spectrum. However, a wrong setting of the box dimension can give rise to truncation artifacts, as well as to remove part of the information. The `KLASSEZ` function for the box apodization is reported in listing 3.2.

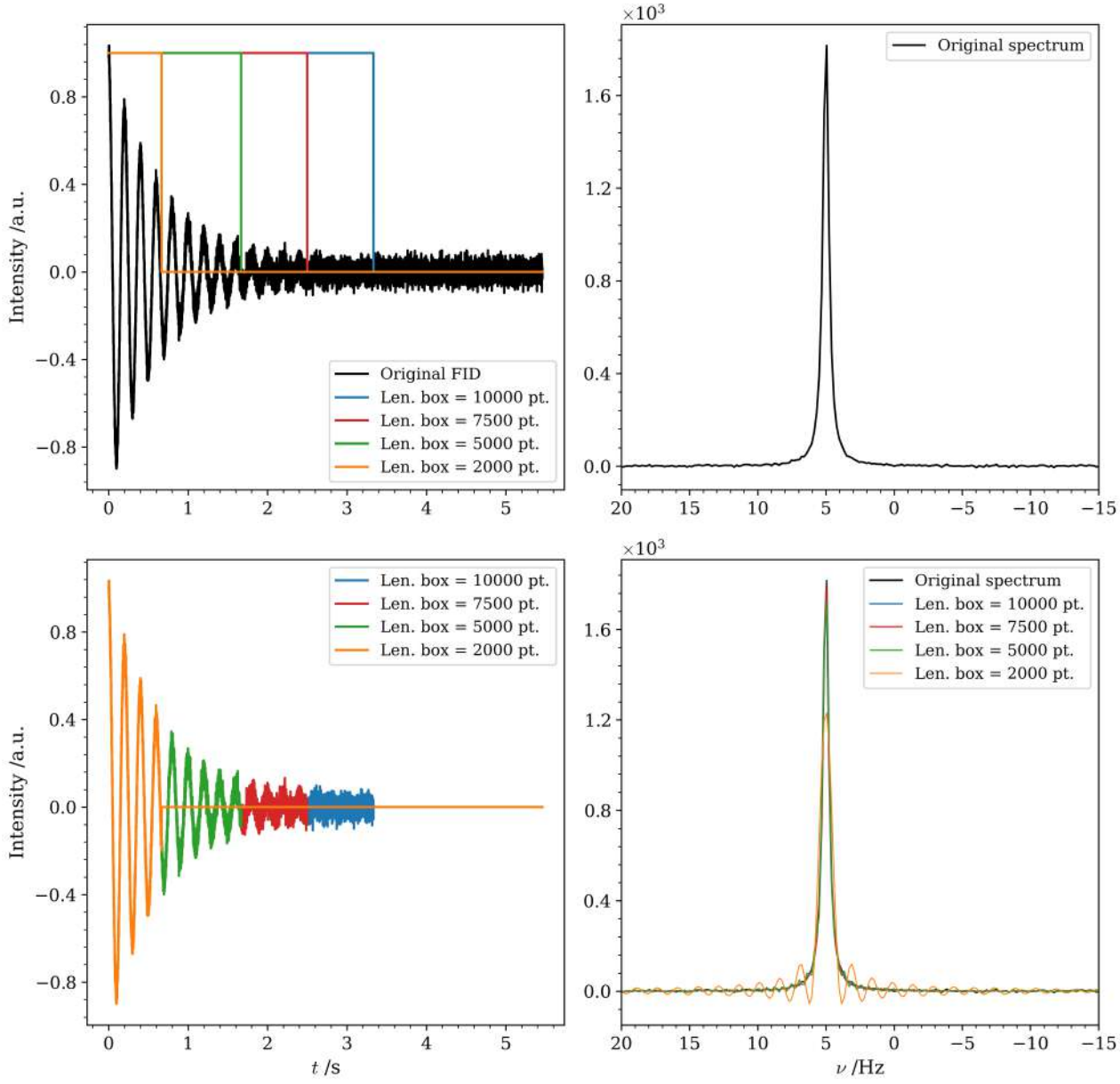


Figure 3.1: The FID of a single Lorentzian peak with a native $\Gamma = 0.5\,\text{Hz}$ and intensity $K = 1$ was simulated, and shown as black trace in the top-left panel. The noise was introduced according to the method described in section 2.3.1, setting the standard deviation $\sigma_N = 0.02$. A series of box window functions of decreasing length was applied on this FID: the results of the apodization is shown in the bottom-left panel. The right panels show the Fourier transformed spectra of the original and processed data.

Listing 3.2: Function that implements the box apodization, from the `processing` module of `KLASSEZ` .

```
def td_eff(data, tdeff):
    """
    Uses only the first tdeff points of data. tdeff must be a list as long as the dimensions:
    tdeff = [F1, F2, ..., Fn]
    --------
    Parameters:
    - data: ndarray
        Data to be trimmed
    - tdeff: list of int
        Number of points to be used in each dimension
    """
```

```
    fid_apod = kz.processing.td_eff(fid, tdeff)
```

3.2.2 Exponential function

An exponential decaying function is known in NMR environment as exponential modulation (em). It is usually implemented as in equation 3.6, where the factor $\texttt{lb} > 0$ can be tuned for sensitivity enhancement.

$$a^{\mathrm{em}}(t|\texttt{lb}) = \exp\{-\pi \texttt{lb} x\}, \quad x = \frac{t}{SW} \tag{3.6}$$

This comes at the price of a loss of resolution, as the signals will appear `lb` broader (figure 3.2). The `KLASSEZ` function for the exponential apodization is reported in listing 3.3.

Listing 3.3: Function that implements the exponential apodization, from the `processing` module of `KLASSEZ` .

```
def em(data, lb, sw):
    """
    Exponential apodization
    ---------
    Parameters:
    - data: ndarray
        Input data
    - lb: float
        Lorentzian broadening. It should be positive.
    - sw: float
        Spectral width /Hz
    """
```

```
    fid_apod = kz.processing.em(fid, lb, sw)
```

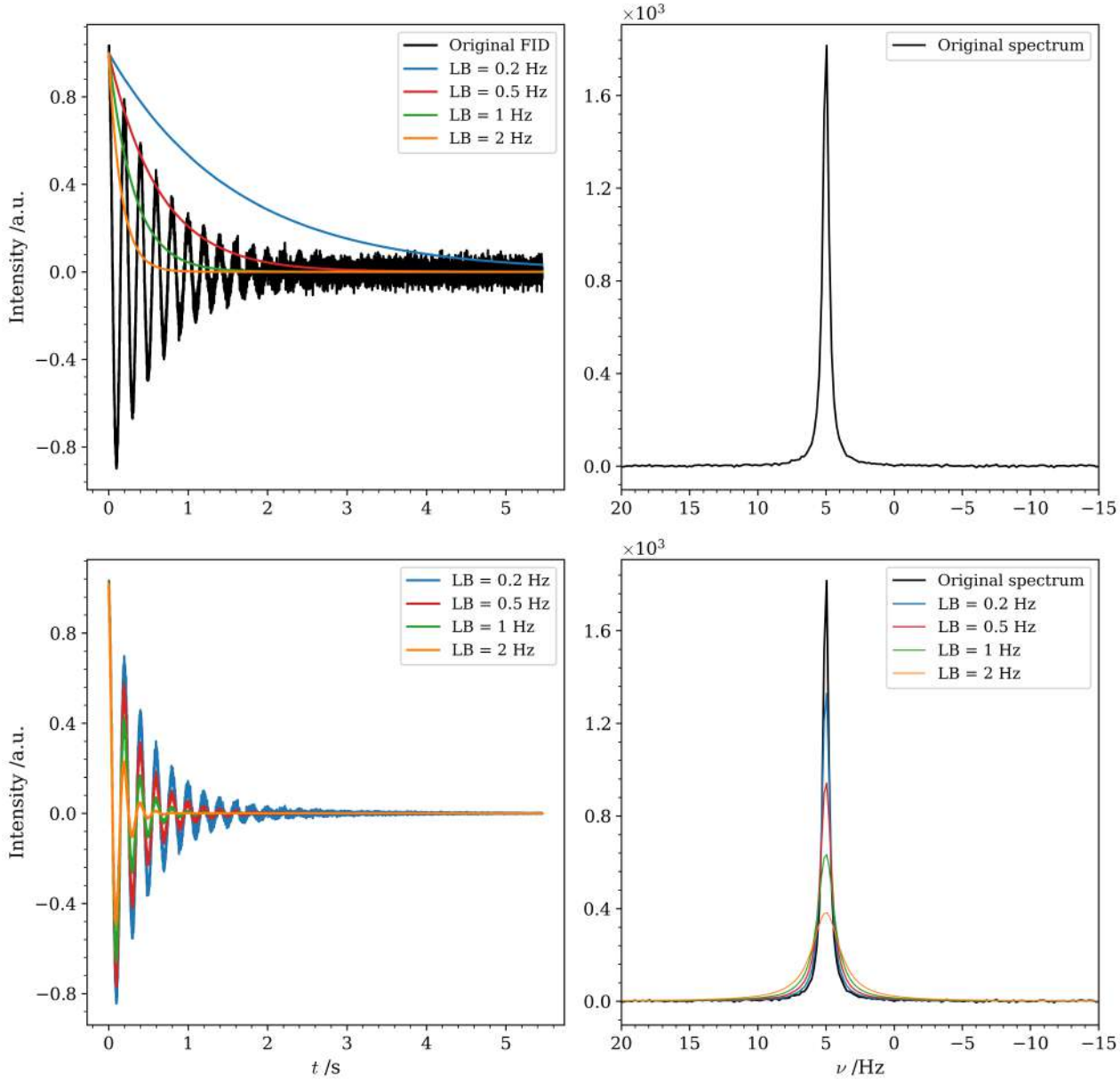


Figure 3.2: The FID of a single Lorentzian peak with a native $\Gamma = 0.5\,\text{Hz}$ and intensity $K = 1$ was simulated, and shown as black trace in the top-left panel. The noise was introduced according to the method described in section 2.3.1, setting the standard deviation $\sigma_N = 0.02$. A series of exponential window functions of increasing dampening factor `lb` was applied on this FID: the results of the apodization is shown in the bottom-left panel. The right panels show the Fourier transformed spectra of the original and processed data.

3.2.3 Sine bells

A very useful window function that can be used for several purposes is the sine bell (equation 3.8) and its squared variant (equation 3.9). It goes smoothly to zero, thus being very effective in removing truncation artifacts, and the position of the maximum can be adjusted by tuning the `ssb` parameter for sensitivity or resolution enhancement.

Being the FID $\mathbb{s}$ an array of N points, a normalized scale $\mathbb{x}$ is defined as:

$$\mathbb{x}[k] = \frac{k}{N} \qquad k = 0, \ldots, N-1 \quad (3.7)$$

Using this scale, the sine-bell (*sin*) and squared sine-bell window functions (*qsin*) are defined as:

$$a^{\sin}(x|\mathtt{ssb}) = \sin\left[\pi\frac{1}{\mathtt{ssb}} + \pi\left(1 - \frac{1}{\mathtt{ssb}}\right)x\right] \quad (3.8)$$

$$a^{\mathrm{qsin}}(x|\mathtt{ssb}) = \sin^2\left[\pi\frac{1}{\mathtt{ssb}} + \pi\left(1 - \frac{1}{\mathtt{ssb}}\right)x\right] \quad (3.9)$$

These window functions with $\mathtt{ssb} = 1$ are useful with sine-modulated data, such as COSY spectra. Alternatively, $\mathtt{ssb} = 2$ is the best choice to remove truncation artifacts from the indirect dimension of 2D datasets. Examples of applications of sine and squared-sine windowing are presented in figure 3.3 and 3.4, respectively. The `KLASSEZ` functions for the sine and squared-sine apodization are reported in listing 3.4.

Listing 3.4: Function that implements the sine and squared-sine apodization, from the `processing` module of `KLASSEZ` .

```
def sin(data, ssb):
    """
    Sine apodization.
    --------
    Parameters:
    - ssb: int
        Sine bell shift.
    """
```

```
    """
    Sine-squared apodization.
    --------
    Parameters:
    - ssb: int
        Sine bell shift.
    """
```

```
    fid_apod = kz.processing.sin(fid, ssb)
    fid_apod = kz.processing.qsin(fid, ssb)
```

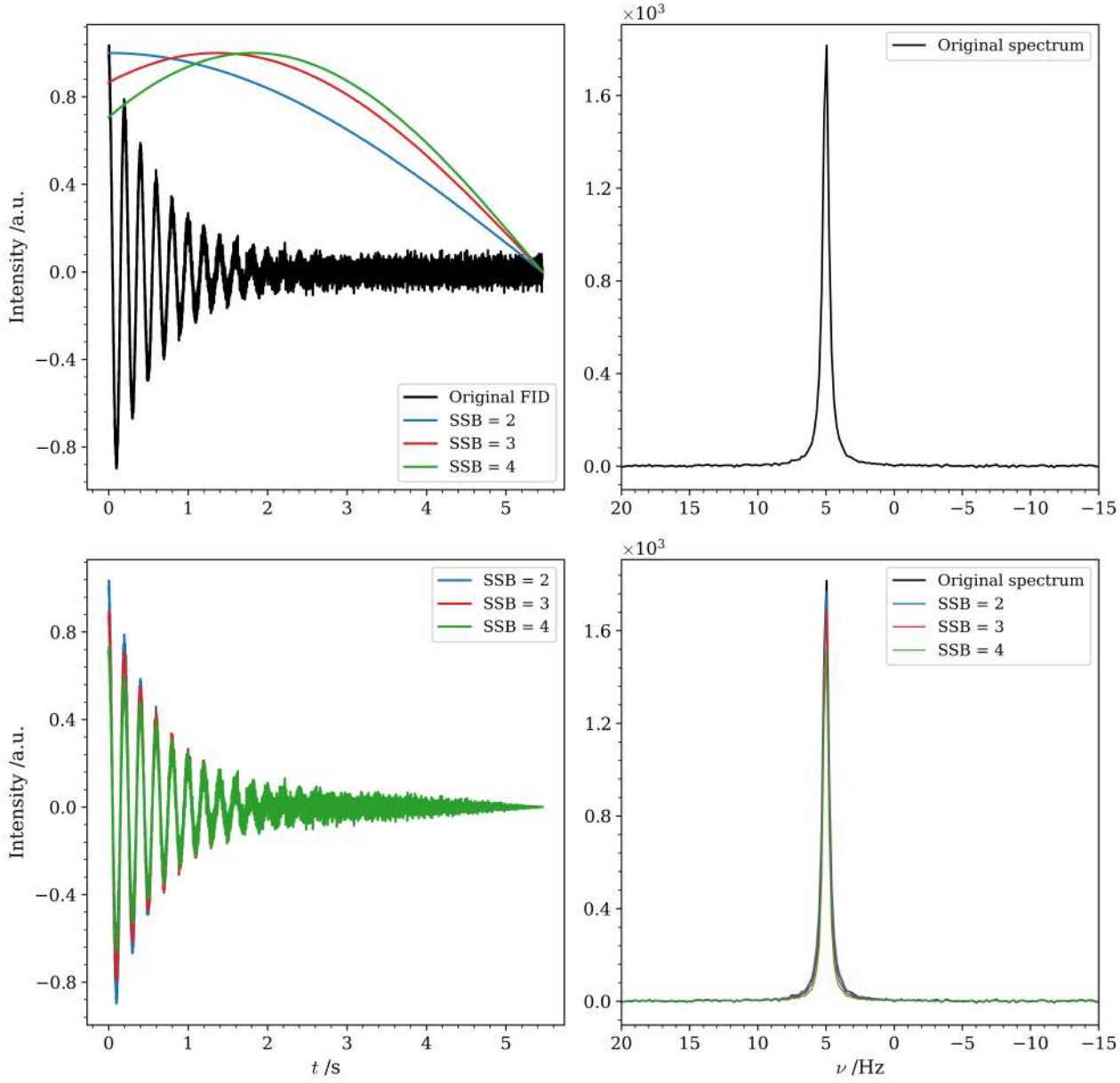


Figure 3.3: The FID of a single Lorentzian peak with a native $\Gamma = 0.5\,\mathrm{Hz}$ and intensity $K = 1$ was simulated, and shown as black trace in the top-left panel. The noise was introduced according to the method described in section 2.3.1, setting the standard deviation $\sigma_N = 0.02$. A series of sine window functions with different values of `ssb` was applied on this FID: the results of the apodization is shown in the bottom-left panel. The right panels show the Fourier transformed spectra of the original and processed data.

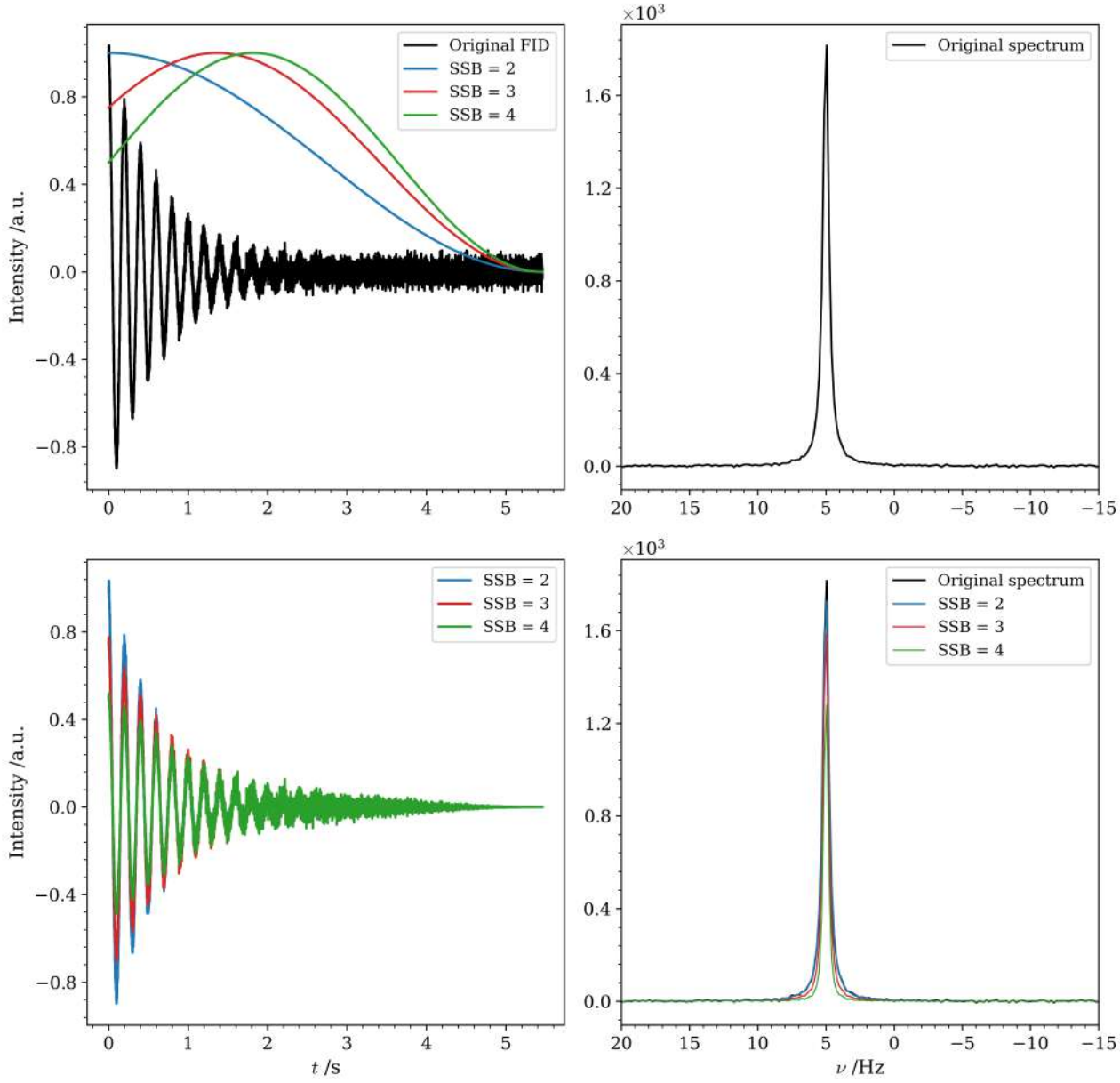


Figure 3.4: The FID of a single Lorentzian peak with a native $\Gamma = 0.5\,\mathrm{Hz}$ and intensity $K = 1$ was simulated, and shown as black trace in the top-left panel. The noise was introduced according to the method described in section 2.3.1, setting the standard deviation $\sigma_N = 0.02$. A series of squared-sine window functions with different values of `ssb` was applied on this FID: the results of the apodization is shown in the bottom-left panel. The right panels show the Fourier transformed spectra of the original and processed data.

3.2.4 Lorentzian to Gaussian transformation

An interesting class of window functions are the ones that transform the lineshape from the theoretical Lorentzian profile to Gaussian. As the tails of a Gaussian function fall off faster than the corresponding Lorentzian with the same linewidth, this transformation is very useful for resolution enhancement.

In principle, one could use an exponential window function as in equation 3.6 with a negative `lb` factor, thus obtaining a rising exponential function, to achieve a resolution enhancement. However, this choice would have a dramatical impact on the noise level. An elegant solution is to use this function in a combination with a Gaussian decay, which has the effect of modifying the lineshape. This is the so-called Bruker-style Gaussian apodization (*gmb*, equation 3.10).

$$a^{\mathrm{gmb}}(x|\mathtt{lb},\mathtt{gb}) = \exp\left\{-\frac{\pi}{SW}\left(\mathtt{lb}x - \frac{\mathtt{lb}}{2\mathtt{gb}}x^2\right)\right\}, \quad \mathbb{x}[k] = k \,\text{for}\, k = 0,\ldots,N-1 \tag{3.10}$$

In this function, the parameter `lb` must be negative, in order to generate a rising exponential function. The `gb` parameter controls the position of the maximum of the Gaussian function, and must be set between 0 (start of the FID) and 1 (end of the FID).

There is also an alternative implementation, where the parameters that control the linear and the quadratic exponentials are disentangled. This "general purpose Gaussian apodization" (*gm*, equation 3.11) depends on three parameters. α controls the terms $\propto \exp\{-x\}$, in the same way as the *em* function does. Instead, the β parameter is proportional to the Gaussian decay, whose maximum is defined by the parameter $\gamma \in [0,1]$, in a similar fashion as `gb` for the *gmb* function.

$$\begin{aligned} &a^{\mathrm{gmb}}(x|\alpha,\beta,\gamma) = \exp\left\{A - B^2\right\} \\ &A(x|\alpha) = -\frac{\pi\alpha}{SW}Nx, \quad B(x|\beta,\gamma) = \frac{\pi\beta}{SW}[\gamma(N-1) - Nx] \end{aligned} \tag{3.11}$$

It is extremely difficult to assign the correct values to all the parameters of these two functions, because of the need to balance the rising and the decaying contribution with respect to the data. The best course of action is to employ an interactive apodization correction, where the effect of the apodization as a function of their values can be observed in real time. An example of application for these two transformations is given in figure 3.5. The `KLASSEZ` functions for the *gm* and *gmb* apodization are reported in listing 3.5.

Listing 3.5: Functions that implement the *gm* and *gmb* apodization, from the `processing` module of `KLASSEZ` .

```
def gm(data, lb, gb, gc, sw):
    """
    Gaussian apodization.
    The parameter 'lb' controls the sharpening factor of a rising exponential, and behaves exactly as in
         processing.em.
    In contrast, 'gb' controls the gaussian decay factor.
    Apply this function VERY CAREFULLY. Choose the right values through the interactive processing.
    -------
    Parameters:
    - data: ndarray
        Input data
    - lb: float
        Lorentzian sharpening /Hz. It should be negative.
    - gb: float
        Gaussian broadening. It should be positive.
    - gc: float
        Gaussian center, relatively to the FID length: 0 <= gc <= 1
    - sw: float
        Spectral width /Hz
    -------
    Returns:
    - pdata: ndarray
        Processed data
    """
```

```
def gmb(data, lb, gb, sw):
    """
    Bruker-style Gaussian apodization.
    Apply this function VERY CAREFULLY. Choose the right values through the interactive processing.
    --------
    Parameters:
    - data: ndarray
        Input data
    - lb: float
        Lorentzian sharpening /Hz. It should be negative.
    - gb: float
        Gaussian broadening. It should be positive.
    - sw: float
        Spectral width /Hz
    -------
    Returns:
    - pdata: ndarray
        Processed data
    """
```

```
    fid_apod = kz.processing.gm(fid, lb, gb, gc, sw)
    fid_apod = kz.processing.gmb(fid, lb, gb, sw)
```

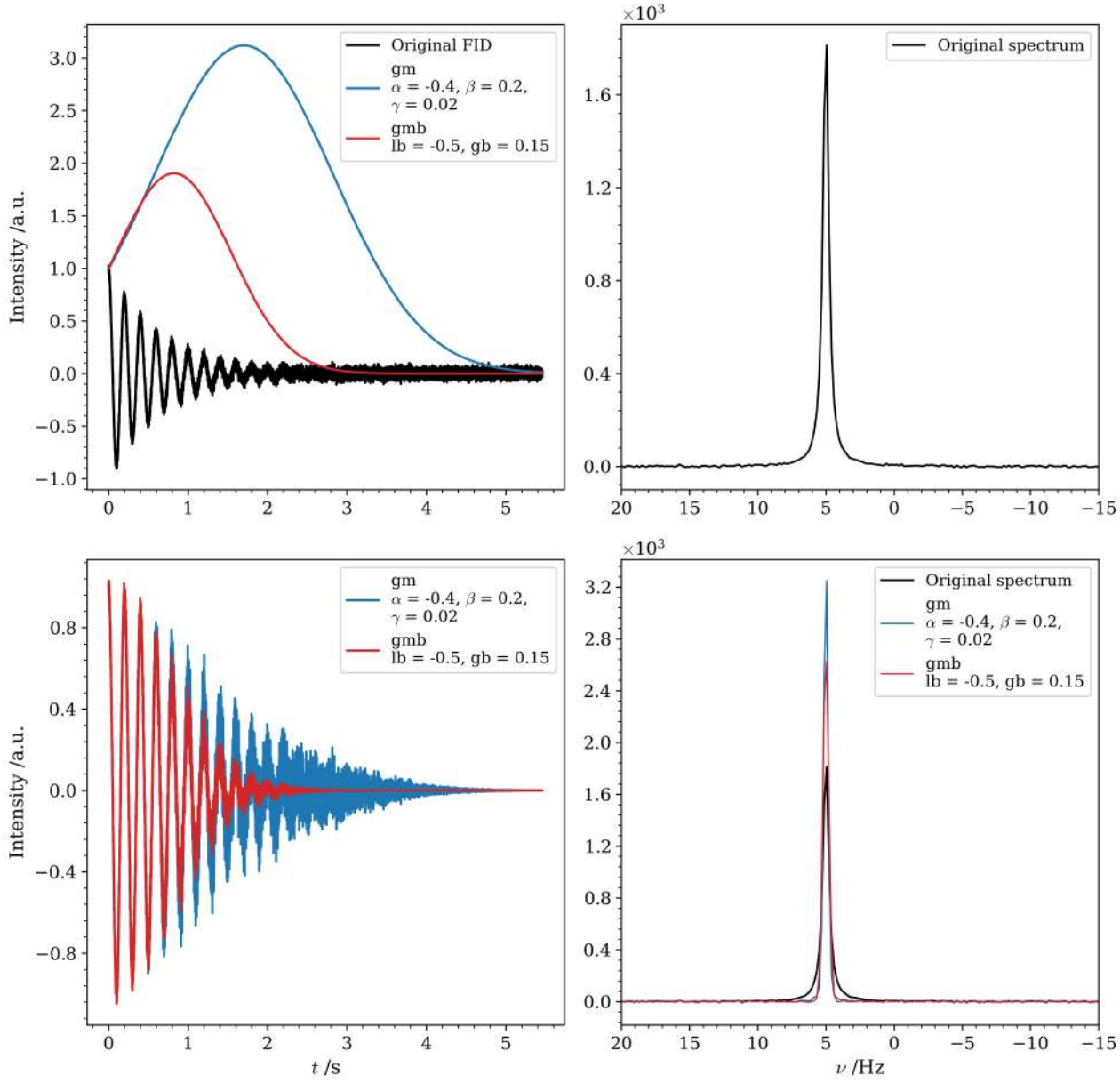


Figure 3.5: The FID of a single Lorentzian peak with a native $\Gamma = 0.5\,\text{Hz}$ and intensity $K = 1$ was simulated, and shown as black trace in the top-left panel. The noise was introduced according to the method described in section 2.3.1, setting the standard deviation $\sigma_N = 0.02$. The FID was multiplied for a *gm* function (parameters: $\alpha = -0.4$, $\beta = 0.2$, $\gamma = 0.02$) and for a *gmb* function (parameters: `lb` $= -0.5$, `gb` $= 0.15$): the results of the apodization is shown in the bottom-left panel. The right panels show the Fourier transformed spectra of the original and processed data.

3.3 Zero-filling

The game-changer in the history of FT-NMR was the development of the Fast Fourier Transform (FFT) algorithm, which is the algorithm nowadays employed for every Fourier-transform application. The beauty of the FFT is that its computational cost scales with $\mathcal{O}(n \log n)$, with respect to the theoretical FT calculation, that scales with $\mathcal{O}(n^2)$, which is a huge improvement of computation speed. One of the prerequisites in order to apply FFT on an array of data (e.g. the FID) is that the number of points the array is composed of is a power of 2. This is the reason why numbers in NMR terminology are expressed as a power of 2: $1k = 2^{10} = 1024$, $2k = 2^{11} = 2048$, $16k = 2^{14} = 16384$, etc. However, there is a number of reason for which the FID is not sampled with a power of 2 time increments. In order to fulfil the requirement, the end of the FID is padded with zeros in order to reach the desired length. Such operation is known as *zero-filling* or *zero-padding*.

Let us suppose that the experimental FID $\mathbb{s}$ is an array of N points, and that we have to zero-fill it to $N' > N$, which is a power of two. The zero-filled FID $\mathbb{s}^{\text{zf}}$ therefore is:

$$\mathbb{s}^{\text{zf}}[k] = \begin{cases} \mathbb{s}[k] & \text{for } k = 0, \dots, N-1 \\ 0 & \text{for } k = N, \dots, N'-1 \end{cases} \tag{3.12}$$

In the frequency domain, the zero-filling operation behaves as an extension by interpolation, i.e. the NMR signal is described by more points, hence it appears smoother. However, it must be noted that this operation does not add information to the data. On the contrary, it might introduce truncation artifacts if the experimental FID is not fully decayed. The `KLASSEZ` function to perform zero-filling is reported in listing 3.6.

Listing 3.6: Function that implements the zero-filling operation, from the `processing` module of `KLASSEZ` .

```
def zf(data, size):
    """
    Zero-filling of data up to size in its last dimension.
    -------
    Parameters:
    - data: ndarray
        Array to be zero-filled
    - size: int
        Number of points of the last dimension after zero-filling
    -------
    Returns:
    - datazf: ndarray
        Zero-filled data
    """
```

```
    fid_zf = kz.processing.zf(fid, newsize)
```

3.4 Linear prediction

There exists an alternative processing method to zero-filling, that allows to reconstruct the points of the FID truncated by the too short acquisition. The idea behind it is the causality effect: each point of the FID is thought to be dependent from the previous ones according to a certain model. It is reasonable to assume that this model is linear. For this reason, the method is called *linear prediction* (LP).

The LP theory, presented in equation 3.13, states that it is possible to write each point of the FID $\mathbb{s}$ as a linear combination of m LP coefficients, $\mathbb{b}$. The number of LP coefficient is the order of the prediction.

$$\mathbb{s}[k] = \sum_{j=0}^{m-1} \mathbb{b}[j]\, \mathbb{s}[k-j-1] \tag{3.13}$$

The usage of equation 3.13 in a predictive manner follows immediately. It is although clear that the prediction error grows larger as further away we push from the original, experimental FID. Another, less trivial cause of failure is a non-linear dependence between subsequent points of the FID. This dependence is very difficult to be estimated *a priori*, hence the only way to see if it works or not for a given dataset is to test several combination of predicted points and order of prediction in a trial-and-error fashion. Nevertheless, in several cases it is a good approximation.

A direct consequence of the LP theory is that it can be used not only to predict "future" points of the FID, but also for "past" points. The first points of the FID can either be corrupted for a series of reason, among which figure clipping and non-linearity of the receiver, or be not sampled at all, for instance due to the presence of the dead time between the end of the last pulse and the start of the acquisition. These kind of errors appear in the spectrum as odd twists of the peaks, or as first-order phase distortions (*vide infra*, section 3.5). Equation 3.13 is commonly known as *forward* linear prediction. The equation for *backward* linear prediction can be derived from equation 3.13 by means of simple algebraic manipulations:

$$\mathbb{s}[k] = \sum_{j=0}^{m-1} \mathbb{b}[j]\, \mathbb{s}[k+j+1] \tag{3.14}$$

I here present an algorithm for the computation of the linear prediction coefficients. Let us suppose that we have an FID of N points, and we want to forward-predict n points with m LP coefficients. We compute a Hankel matrix $\mathbb{D}$, that has $\mathbb{s}[0 : m-1]$ as first row and $\mathbb{s}[m-1 : N-2]$ as last column, and an array $\mathbb{d}$ as:

$$\mathbb{D} = \begin{pmatrix} \mathbb{s}[0] & \mathbb{s}[1] & \dots & \mathbb{s}[m-1] \\ \mathbb{s}[1] & \mathbb{s}[2] & \dots & \mathbb{s}[m] \\ \mathbb{s}[2] & \dots & \dots & \mathbb{s}[m+1] \\ \ddots & \ddots & \ddots & \vdots \\ \ddots & \ddots & \ddots & \mathbb{s}[N-3] \\ \dots & \dots & \dots & \mathbb{s}[N-2] \end{pmatrix}, \quad \mathbb{d} = \mathbb{s}[m : N-1] = \begin{pmatrix} \mathbb{s}[m] \\ \mathbb{s}[m+1] \\ \mathbb{s}[m+2] \\ \dots \\ \mathbb{s}[N-2] \\ \mathbb{s}[N-1] \end{pmatrix} \tag{3.15}$$

The task of determining the LP coefficients $\mathbb{b} = \begin{pmatrix} \mathbb{b}[0] & \dots & \mathbb{b}[m-1] \end{pmatrix}$ is thus to solve the linear system

$$\mathbb{D}\mathbb{b} = \mathbb{d} \implies \begin{pmatrix} \mathbb{s}[0] & \mathbb{s}[1] & \dots & \mathbb{s}[m-1] \\ \mathbb{s}[1] & \mathbb{s}[2] & \dots & \mathbb{s}[m] \\ \mathbb{s}[2] & \dots & \dots & \mathbb{s}[m+1] \\ \ddots & \ddots & \ddots & \vdots \\ \ddots & \ddots & \ddots & \mathbb{s}[N-3] \\ \dots & \dots & \dots & \mathbb{s}[N-2] \end{pmatrix} \begin{pmatrix} \mathbb{b}[0] \\ \mathbb{b}[1] \\ \mathbb{b}[2] \\ \vdots \\ \mathbb{b}[m-2] \\ \mathbb{b}[m-1] \end{pmatrix} = \begin{pmatrix} \mathbb{s}[m] \\ \mathbb{s}[m+1] \\ \mathbb{s}[m+2] \\ \vdots \\ \mathbb{s}[N-2] \\ \mathbb{s}[N-1] \end{pmatrix} \tag{3.16}$$

which is exactly the matrix representation of equation 3.13. The solution to the linear system is given by

$$\mathbb{b} = \mathbb{D}^{+}\mathbb{d} \tag{3.17}$$

The actual algorithms for linear prediction take their name from the method they use for the computation of the pseudo-inverse: LP-SVD (most used), LP-QRD, LP-Cholesky[26]. The N-th and following points of the FID can be readily predicted by:

$$\mathbb{s}[N+k] = \mathbb{s}_k^T \, \mathbb{b}, \quad \mathbb{s}_k = \mathbb{s}[N-m+k-1 : N+k-1] \qquad \text{for } k = 0, \ldots, n-1 \quad (3.18)$$

For the backward LP, the same solution strategy holds, but the matrix $\mathbb{D}$ and the vector $\mathbb{d}$ are built differently:

$$\mathbb{D} = \begin{pmatrix} \mathbb{s}[1] & \mathbb{s}[2] & \ldots & \mathbb{s}[m-1] \\ \mathbb{s}[2] & \mathbb{s}[3] & \ldots & \mathbb{s}[m] \\ \mathbb{s}[3] & \ldots & \ldots & \mathbb{s}[m+1] \\ \ddots & \ddots & \ddots & \vdots \\ \ddots & \ddots & \ddots & \mathbb{s}[N-2] \\ \ldots & \ldots & \ldots & \mathbb{s}[N-1] \end{pmatrix}, \quad \mathbb{d} = \mathbb{s}[0 : N-m-1] = \begin{pmatrix} \mathbb{s}[0] \\ \mathbb{s}[1] \\ \mathbb{s}[2] \\ \ldots \\ \mathbb{s}[N-m-2] \\ \mathbb{s}[N-m-1] \end{pmatrix} \quad (3.19)$$

With $\mathbb{D}$ and $\mathbb{d}$ from equation 3.19, equation 3.16 becomes equivalent to equation 3.14. The n previous points to the start of the acquisition are thus predicted by:

$$\mathbb{s}[-k-1] = \mathbb{s}_k^T \, \mathbb{b}, \quad \mathbb{s}_k = \mathbb{s}[-k : m-k] \qquad \text{for } k = 0, \ldots, n-1 \quad (3.20)$$

The linear prediction is implemented in `KLASSEZ` as the function `lp` of the `processing` module (see listing 3.7).

Listing 3.7: Function that implements the linear prediction, from the `processing` module of `KLASSEZ` .

```
            u_cal[i] = int(u)
            u_cal_ppm[i] = u * misc.calcres(ppm_scale)

    # Apply the correction
    data_roll = []    # Initialize output variable
    for i, experiment in enumerate(data_in):    # Loop over the experiments
        # Roll the spectra of the appropriate amount and append them to the list
        data_roll.append(np.roll(experiment, int(u_cal[i])))
    # Transform into array
    data_roll = np.array(data_roll)

    return data_roll, u_cal, u_cal_ppm

def lp(data, pred=1, order=8, mode='b'):
    """
    Apply linear prediction on the dataset.
    This method solves the linear system
        D a = d
    where 'a' is the array of lp coefficients.
    -----------
```

```
    fid_lp = kz.processing.lp(fid, pred, order, mode)
```

3.5 Phase correction

In the physics of NMR experiments, each signal has its own phase. This is actively taken into account in the models proposed in section 2.1 through the parameter ϕ. However, in the vast majority of datasets, it is possible to approximate the phases of all signals with only two

parameters: ϕ_0 and ϕ_1. A correction function that depends on these two parameters, stated in equation 3.21, can be applied on the Fourier-transformed spectrum in order to make the signals to have phase equal to zero. In this condition, the spectrum is said to be "in phase".

$$p_c(\omega|\phi_0, \phi_1) = \exp\{\mathrm{i}(\phi_0 + \omega\phi_1)\} \tag{3.21}$$

Due to how they appear in equation 3.21, the terms ϕ_0 and ϕ_1 are named zero- and first-order phase correction angles, respectively.

The phase offset ϕ_0 accounts for the fact that the signal arrives to the detector with an arbitrary phase. In fact, when programming phase cycles in NMR pulse sequences, the only important feature is the difference between the phases at each pulse, as the final phase can be easily corrected by this factor.

The frequency-dependent term is a little bit more complicated. From the properties of Fourier transform, it can be proved that frequency-dependent phase shifts are correlated to time-shifts of the FID. One of the possible reasons for a time-shift of the FID is the presence of the dead time between the end of the last pulse of the sequence and the start of the acquisition. This delay, which is of the order of tens of µs, is needed to avoid the ringing of the pulse to enter in the receiver and to switch the coil operation mode from "pulse" to "acquire". If a substantial part of the information is encoded in the first points of the FID, such as with paramagnetic samples, the effect is precisely a time-shift of the FID. Another cause is the infamous group delay, an artifact introduced by the digital filter, which appears as n zeroes at the beginning of the FID. In this latter case, the group delay can be removed by applying a first-order phase correction with $\phi_1 = 2\pi n$.

The phase correction angles are traditionally computed interactively by dedicated GUIs. However, when one has to deal with a dataset that contains tens or hundreds of spectra, this approach can become cumbersome. For this reason, there exist a number of algorithms for the automatic phase correction, which are based on the iterative fitting of ϕ_0 and ϕ_1 until certain conditions are satisfied. These conditions can include symmetry of the peaks[27], minimum integral of the imaginary part[28], dispersion/absorption plots[29], minimization of entropy[30], and many others. However, these algorithms are usually very sensitive to baseline distortions.

In `KLASSEZ` , phase correction is applied either directly with known values of phase angles, or interactively through a dedicated user interfaces, as shown in listing 3.8.

Listing 3.8: Phase correction functions from the from the `processing` module of `KLASSEZ` . The application on a non-phased spectrum `S` is shown.

```
def ps(data, ppmscale=None, p0=None, p1=None, pivot=None, interactive=False):
    """
    Applies phase correction on the last dimension of data.
    The pivot is set at the center of the spectrum by default.
    Missing parameters will be inserted interactively.
    -------
    Parameters:
    - data: ndarray
        Input data
    - ppmscale: 1darray or None
        PPM scale of the spectrum. Required for pivot and interactive phase correction
    - p0: float
        Zero-order phase correction angle /degrees
    - p1: float
        First-order phase correction angle /degrees
    - pivot: float or None.
        First-order phase correction pivot /ppm. If None, it is the center of the spectrum.
    - interactive: bool
        If True, all the parameters will be ignored and the interactive phase correction panel will be opened.
    --------
    Returns:
    - datap: ndarray
        Phased data
    - final_values: tuple
        Employed values of the phase correction. (p0, p1, pivot)
    """
```

```
    # With known values of p0, p1 and pv
    S_phased, *_ = kz.processing.ps(ppm_scale, S, p0, p1, pv)
    # Interactively
    S_phased, phase_angles = kz.processing.ps(ppm_scale, S)
```

3.6 Baseline correction

Computing the baseline for the imaginary part of the spectrum is not a common procedure in NMR processing. Therefore, how to actually do it is still an open issue. In principle, the real and the imaginary part of an NMR spectrum should have the same baseline. However, there might be cases in which they differ, e.g. if it is due to interactions with a macromolecule[31].

The most straightforward way to correct a distorted baseline is to manually adjust the parameters of a given model function on a restricted portion of the spectrum. There are several functions that can be employed for this purpose: the ones usually implemented in commercial processing softwares are polynomia, exponentials, sine bells and sinc functions. Although this method is the one that gives the best results, it can be very tricky to be applied. Besides the optimization times, that can easily escalate to tens of hours even in not very complicated spectrum, a critical point is encountered in the junctions between adjacent regions, which normally feature "baseline spikes" and cannot be easily corrected with this method.

Another option is to compute an interpolating smooth function in the parts of the spectrum that do not contain peaks, using for instance Savitzky-Golay filters[11], Whittaker-like smoothers[32], or splines[33]. This is not a solution to the problem, but a simple redirection to another task: determine the peaks positions. Therefore, the accuracy of this class of methods relies on the accuracy of peak modelling. A further problem arises from the fact that phase and baseline correction are mutually-dependent tasks: the algorithms for automatic phase correction require flat baseline, while the algorithm for baseline computation require the spectrum to be in phase, otherwise the peak-picking fails. The SINC algorithm[34] was proposed to address the two tasks in a synergic fashion, by fitting the phase angles while the baseline is computed using a smoother at each iteration. However, its major drawback is that it still requires a prior knowledge about the positions of the signals that one wants to exclude from the baseline computation.

Fitting approaches using slow-varying functions as models (e.g. polynomia) are deemed to fail without prior peak subtraction, as the peaks make a too big contribution in the calculation of the residuals for a least-squares approach. The solution to this problem is to actually employ a non-quadratic cost functions, in order to decrease the impact of the peaks on the residuals[35].

Given an array of residuals $\mathbb{r}$, the target value to be minimized in a fitting routine, R, is computed from the target function φ as:

$$R = \sum_k \varphi(\mathbb{r}[k]) \tag{3.22}$$

In this formalism, the target function for the ordinary least-squares approach is:

$$\varphi(r) = r^2 \tag{3.23}$$

The non-quadratic approach consists in defining a threshold value s in order to modify the way the residuals are handled by the target function when they lie above that threshold. There are two functions worth to be mentioned: the Huber function (equation 3.24), which behaves linearly above s, and the truncated-quadratic function (equation 3.25), which behaves as a constant above s.

$$\varphi(r) = \begin{cases} r^2 & \text{if } |r| \leq s \\ 2s|r| - s^2 & \text{otherwise} \end{cases} \tag{3.24}$$

$$\varphi(r) = \begin{cases} r^2 & \text{if } |r| \leq s \\ s^2 & \text{otherwise} \end{cases} \tag{3.25}$$

A visual comparison between these three cost functions is shown in figure 3.6. These functions are known to be used for optimization problems in general, when the data present outliers. Following this rationale, these cost functions work only in situations where the peaks can be considered as outliers, and the "real" data to be modelled is the baseline.

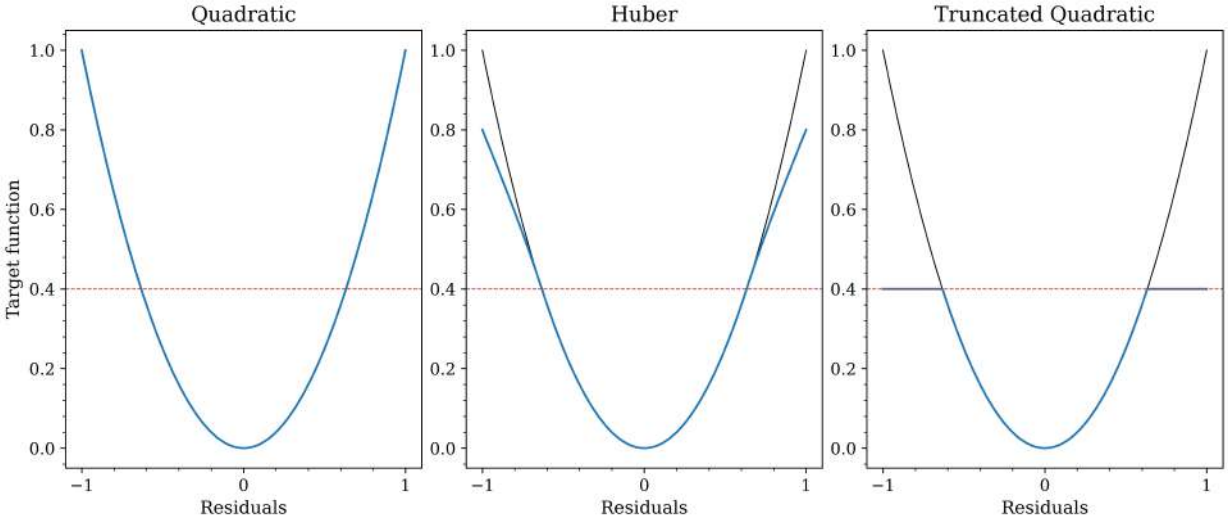


Figure 3.6: Plot of the behavior of three possible options for target functions: quadratic (equation 3.23, left), Huber (equation 3.24, center), truncated quadratic (equation 3.25, right). The threshold value s was set to 0.4. In all panels, the trend of the ordinary quadratic function is shown as black trace to better appreciate the differences.

With these assumptions, the baseline correction problem is now to find the set of coefficients $\mathbb{c}$ which minimize the difference between the experimental spectrum $\mathbb{y}$ and the polynomial $p(\mathbb{x}|\mathbb{c})$. The initial guess for this fit is set to be the ordinary least-squares solution, computed as shown in appendix A5. At this point, the optimized set of coefficient is calculated iteratively using the trust-region reflective least-squares minimization algorithm (implemented in `lmfit` with the *least-squares* method keyword), employing the non-quadratic cost function.

The model-based approach for the description of a baseline offers several advantages with respect to its computation with a smoother. First of all, the computed baseline becomes independent from processing operations that alter the number of points of the spectrum, such as zero-filling, linear prediction, or trimming the end of the FID. As a result, the baseline coefficients can be computed just once and then saved in a file, to be read when needed. As the actual polynomion scale is normalized, the coefficients are the only information required to rebuild the baseline at any time.

The non-quadratic fitting procedure also has a very interesting feature: it is independent from the phase of the peaks. This allows to compute the baseline *before* performing the phase correction. In order to do this, it is necessary to compute the baseline also for the imaginary part of the spectrum, because if the baseline has an imaginary part it can enter into phase-correction algorithms without any problems.

The key (and only) parameter to set in order to generate a suitable baseline is the order of the polynomion to be computed: generally 4 is a good choice. Once the baseline is computed and subtracted from the original spectrum, the key requirement for the phase correction algorithm is fulfilled; hence, the choice of the method to phase the spectrum is only determined by the user's taste. In particular, we found the one implemented in the SINC algorithm quite useful in this framework.

The algorithm is implemented in `KLASSEZ` under the function `rpbc` of the `processing` module, and can be invoked as in listing 3.9. An example of a run of the whole program, i.e.

baseline computation and then phase correction, is shown in figure 3.7.

Listing 3.9: Function that implements the simultaneous phase and baseline correction, from the module `processing` of `KLASSEZ` . An example of application on the spectrum `S` is also shown.

```
    W = sps.lil_matrix((m, m)) # Sparse weights matrix
    W.setdiag(w)
    W.tocsr()      # Conversion to csr

    A = sps.csr_matrix(W + s_f * D.T @ D) # Sparse criterion
    z = spsolve(A, w*y)   # Find solutions using LU factorization

    return z
```

```
def rpbc(data, split_imag=False, n=5, basl_method='huber', basl_thresh=0.2, basl_itermax=2000, **phase_kws):
    """
    Reversed Phase and Baseline Correction.
    Allows for the automatic phase correction and baseline subtraction of NMR spectra.
    It is called "reversed" because the baseline is actually computed and subtracted before to perform the phase
        correction.

    The baseline is computed using a low-order polynomion, built on a scale that goes from -1 to 1, whose
        coefficients are obtained minimizing a non-quadratic cost function. It is recommended to use either "tq"
        (truncated quadratic, much faster) or "huber" (Huber function, slower but sometimes more accurate). The
        user is requested to choose between separating the real and imaginary channel in this step. The order of
        the polynomion and the threshold value are the key parameters for obtaining a good baseline. The used
        function is processing.polyn_basl

    The phase correction is computed on the baseline-subtracted complex data as described in the SINC algorithm (
        ref.). The default parameters are generally fine, but in case of data with poor SNR (approximately SNR <
        10) better results can be obtained by increasing the value of the e1 parameter. The employed function is
        processing.SINC_phase
    -----------
    Parameters:
    - data: 1darray
        Data to be processed, complex-valued
    - split_imag: bool
        If True, computes the baseline on the real and imaginary part separately; else, the set of polynomion
            coefficients are forced to be the same for both
    - n: int
        Number of coefficients of the polynomion, i.e. it will be of degree n-1
    - basl_method: str
        Cost function to be minimized for the baseline computation. Look for fit.CostFunc, "method" attribute
    - basl_thresh: float
        Relative threshold value for the non-quadratic behaviour of the cost function. Look for fit.CostFunc, "s"
            attribute
    - basl_itermax: int
```

```
    S_proc, p0, p1, basl_coeff = kz.processing.rpbc(S)
    x = np.linspace(-1, 1, S.shape[-1])
    baseline = kz.misc.polyn(x, basl_coeff)
```

We faced algorithm failures when processing spectra acquired at benchtop spectrometers: as the resonating frequencies of the investigated nuclei are low, the excitation profile of the pulses cover a much wider spectral window with respect to the chemical shift dispersion of the signals. For instance, a standard 90° pulse of 20 μs covers a spectral window of 12.5 kHz, that at 42.5 MHz ^{1}H Larmor frequency (1 T) correspond to about 300 ppm. Considering that the ^{1}H-NMR signals are normally concentrated in the $0-10$ ppm range, the 96% of the contribution to the cost functions would come from flat baseline-only regions.

We are currently working on an algorithm for baseline correction based on Fast Iterative Filtering (FIF)[36,37], that is a more computationally-efficient method of Empirical Mode De-

composition (EMD)[38]. The idea is to partition the spectrum into well-behaved amplitude and frequency modulation components called Intrinsic Mode Functions (IMFs), which represent the oscillatory component of the spectrum itself, sorted for increasing frequency. This method suits very well for NMR, as the signals represent a much faster intensity variation than the baseline: in this framework, the baseline goes in the first one or two IMFs.

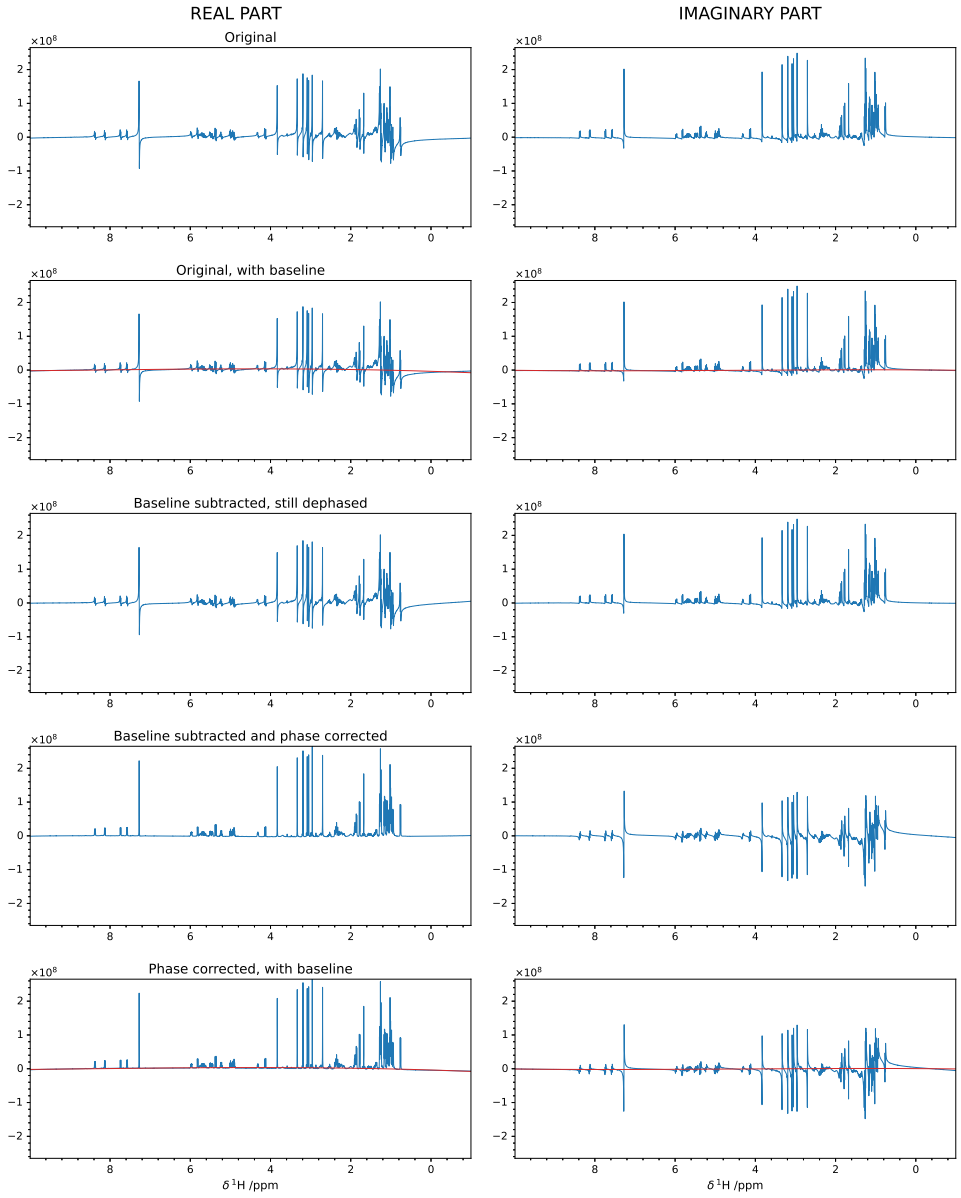


Figure 3.7: Example of application of the baseline and phase correction on the ^{1}H-NMR experimental spectrum of cholesteryl acetate, from Bruker TopSpin's test data. The baseline was computed using a 4-th degree polynomion, using separate coefficients for real and imaginary part. Phase correction was computed using the algorithm proposed in SINC[34], with $\gamma_1 = 10, \gamma_2 = 0.01, \gamma_3 = 0, \varepsilon_1 = 0, \varepsilon_2 = 0$.

3.7 Processing of a 2D spectrum

The processing of a 1D spectrum consists of applying the operations listed in the previous sections. In `KLASSEZ` , this is conveniently coded in the function `fp` of the `processing` module, as shown in appendix A2.1. Although there are details to take into account for each of these steps, conceptually the task is quite easy to be rationalized. The same cannot be told when dealing with 2D spectra. The 2D spectrum cannot be computed by an actual 2D Fourier-transform, because of the way the indirect dimension is acquired. Instead, the 1D-like Fourier transform is applied on the rows of the FID, then the resulting interferogram is transposed, and the Fourier transform is applied again. Finally, the spectrum is transposed again, to give its correct appearance with respect to the frequency axes.

3.7.1 Dealing with hypercomplex data

What we mean with "transpose" is actually a delicate topic, as the correct transposition method depends on the way the frequency discrimination is achieved. In the "QF" method, the FID is made of normal complex numbers, therefore the usual transposition method $\mathbb{A}^T[i,j] = \mathbb{A}[j,i]$ is sufficient. On the contrary, the States-TPPI FID has a hypercomplex number as building block[39]. To adapt the concept of a four-dimensional mathematical entity to the binary representation of complex number, the real and imaginary parts of each point are saved in two subsequent rows of the FID. In other words, a hypercomplex number $h = r + a\mathrm{i} + b\mathrm{j} + c\mathrm{k}$ can be saved in a 2×1 complex matrix block as:

$$\mathbb{H} = \begin{pmatrix} r + a\mathrm{i} \\ b + c\mathrm{i} \end{pmatrix} \tag{3.26}$$

This formalism holds for each entry of a States-TPPI FID.

At this point, it is intuitive that a normal transposition would have the effect of scrambling the various components of nearby entries. A dedicated transpose method must be designed in order to preserve the structure of the hypercomplex number inside the matrix. The *hypercomplex transpose* is thus defined as follows. Let $\mathbb{A}$ a $m \times n$ matrix with entries arranged as in equation 3.26. The hypercomplex transpose $\mathbb{A}^{HT}$ of the matrix $\mathbb{A}$ is the $(2n) \times (m//2)$ matrix with entries

$$\begin{aligned} \mathbb{A}^{HT}[2i, j] &= \mathrm{Re}\left\{\mathbb{A}[2j, i]\right\} + \mathrm{i}\,\mathrm{Re}\left\{\mathbb{A}[2j+1, i]\right\} \\ \mathbb{A}^{HT}[2i+1, j] &= \mathrm{Im}\left\{\mathbb{A}[2j, i]\right\} + \mathrm{i}\,\mathrm{Im}\left\{\mathbb{A}[2j+1, i]\right\} \\ &\text{for } i = 0, \ldots, n \text{ and } j = 0, \ldots, m//2 \end{aligned} \tag{3.27}$$

Visually speaking, given an example 6×4 matrix $\mathbb{A}$:

$$\mathbb{A} = \begin{pmatrix} x_{1,1} + \mathrm{i}y_{1,1} & x_{1,2} + \mathrm{i}y_{1,2} & x_{1,3} + \mathrm{i}y_{1,3} & x_{1,4} + \mathrm{i}y_{1,4} \\ x_{2,1} + \mathrm{i}y_{2,1} & x_{2,2} + \mathrm{i}y_{2,2} & x_{2,3} + \mathrm{i}y_{2,3} & x_{2,4} + \mathrm{i}y_{2,4} \\ x_{3,1} + \mathrm{i}y_{3,1} & x_{3,2} + \mathrm{i}y_{3,2} & x_{3,3} + \mathrm{i}y_{3,3} & x_{3,4} + \mathrm{i}y_{3,4} \\ x_{4,1} + \mathrm{i}y_{4,1} & x_{4,2} + \mathrm{i}y_{4,2} & x_{4,3} + \mathrm{i}y_{4,3} & x_{4,4} + \mathrm{i}y_{4,4} \\ x_{5,1} + \mathrm{i}y_{5,1} & x_{5,2} + \mathrm{i}y_{5,2} & x_{5,3} + \mathrm{i}y_{5,3} & x_{5,4} + \mathrm{i}y_{5,4} \\ x_{6,1} + \mathrm{i}y_{6,1} & x_{6,2} + \mathrm{i}y_{6,2} & x_{6,3} + \mathrm{i}y_{6,3} & x_{6,4} + \mathrm{i}y_{6,4} \end{pmatrix} \tag{3.28}$$

its hypercomplex transpose $\mathbb{A}^{HT}$ is the 8×3 matrix:

$$\mathbb{A}^{HT} = \begin{pmatrix} x_{1,1} + \mathrm{i}x_{2,1} & x_{3,1} + \mathrm{i}x_{4,1} & x_{5,1} + \mathrm{i}x_{6,1} \\ y_{1,1} + \mathrm{i}y_{2,1} & y_{3,1} + \mathrm{i}y_{4,1} & y_{5,1} + \mathrm{i}y_{6,1} \\ x_{1,2} + \mathrm{i}x_{2,2} & x_{3,2} + \mathrm{i}x_{4,2} & x_{5,2} + \mathrm{i}x_{6,2} \\ y_{1,2} + \mathrm{i}y_{2,2} & y_{3,2} + \mathrm{i}y_{4,2} & y_{5,2} + \mathrm{i}y_{6,2} \\ x_{1,3} + \mathrm{i}x_{2,3} & x_{3,3} + \mathrm{i}x_{4,3} & x_{5,3} + \mathrm{i}x_{6,3} \\ y_{1,3} + \mathrm{i}y_{2,3} & y_{3,3} + \mathrm{i}y_{4,3} & y_{5,3} + \mathrm{i}y_{6,3} \\ x_{1,4} + \mathrm{i}x_{2,4} & x_{3,4} + \mathrm{i}x_{4,4} & x_{5,4} + \mathrm{i}x_{6,4} \\ y_{1,4} + \mathrm{i}y_{2,4} & y_{3,4} + \mathrm{i}y_{4,4} & y_{5,4} + \mathrm{i}y_{6,4} \end{pmatrix} \tag{3.29}$$

It must be noted that applying the hypercomplex transposition twice, the original matrix is retrieved.

$$\left(\mathbb{A}^{HT}\right)^{HT} = \mathbb{A} \tag{3.30}$$

The implementation of the hypercomplex transpose in `KLASSEZ` is shown in listing 3.10.

Listing 3.10: Function that implements the hypercomplex transpose, from the `processing` module of `KLASSEZ` .

```
def tp_hyper(data):
    """
    Computes the hypercomplex transpose of data.
    Needed for the processing of data acquired in a phase_sensitive manner
    in the indirect dimension.
    """
```

```
    fid_HT = kz.processing.tp_hyper(fid)
```

3.7.2 States-TPPI processing

The processing workflow for a 2D States-TPPI spectrum thus is represented by the following list. All the operations are intended to be performed on one row of the FID at the time, as stated in the previous sections.

1. Apodization
2. Zero-filling
3. Fourier transform: $\mathbb{S}(t_1, t_2) \rightarrow \mathbb{S}(t_1, \omega_2)$
4. Hypercomplex transposition: $\mathbb{S}(t_1, \omega_2) \rightarrow \mathbb{S}^{HT}(\omega_2, t_1)$
5. Apodization
6. Zero-filling
7. Fourier transform: $\mathbb{S}^{HT}(\omega_2, t_1) \rightarrow \mathbb{S}^{HT}(\omega_2, \omega_1)$
8. Hypercomplex transposition: $\mathbb{S}^{HT}(\omega_2, \omega_1) \rightarrow \mathbb{S}(\omega_1, \omega_2)$

In order to obtain the correct appearance for the spectrum, the transposed interferogram $\mathbb{S}^{HT}(\omega_2, t_1)$ needs to be corrected before the Fourier transform at point 7. The operation to apply is:

$$\mathbb{S}^{HT}[i,j] = (-1)^j \big(\mathbb{S}^{HT}[i,j]\big)^* \tag{3.31}$$

which re-establish the cosine-modulation in the real part and the sine-modulation in the imaginary part of the hypercomplex transposed FID, so that the Fourier transform gives rise to a single peak.

Usually, it is only the real part of the spectrum that is visualized and employed for data interpretation and analysis. Hence, we have to separate each hypercomplex entry of the spectrum array into its four components. These four components, that appear as (r, a, b, c) in equation 3.26, give rise to four arrays, which are conventionally named `rr`, `ir`, `ri` and `ii`. Let us consider a fully processed 2D States-TPPI spectrum $\mathbb{S}$ of dimension $m \times n$. The decomposition of $\mathbb{S}$ into the four component arrays is achieved as follows:

- $r \rightarrow$ `rr`

  $$\mathbb{rr}[i,j] = \mathrm{Re}\{\mathbb{S}[2k,j]\} \qquad \text{for } k = 0, \ldots, m//2 - 1$$

- $a \rightarrow$ `ir`

  $$\mathbb{ir}[i,j] = \mathrm{Im}\{\mathbb{S}[2k,j]\} \qquad \text{for } k = 0, \ldots, m//2 - 1$$

- $b \rightarrow$ `ri`

  $$\mathbb{ri}[i,j] = \mathrm{Re}\{\mathbb{S}[2k+1,j]\} \qquad \text{for } k = 0, \ldots, m//2 - 1$$

- $c \rightarrow$ `ii`

  $$\mathbb{ii}[i,j] = \mathrm{Im}\{\mathbb{S}[2k+1,j]\} \qquad \text{for } k = 0, \ldots, m//2 - 1$$

Clearly, the real part mentioned above is only the $\mathbb{rr}$ matrix.

3.7.3 Echo-Antiecho processing

The Echo-Antiecho acquisition mode employs gradients for the frequency discrimination. The Anti-Echo transients are saved in the even rows of the FID, whereas the Echo transients are saved in the odd rows. An Echo-Antiecho $m \times n$ FID $\mathbb{D}$ must be converted in a States-TPPI format by the following operation:

$$\begin{aligned}
\mathbb{S}^{\text{S.-TPPI}}[2k,j] &= +\,\mathrm{Re}\,\{\mathbb{D}[2k,j] - \mathbb{D}[2k+1]\} + \mathrm{i}\,\mathrm{Im}\,\{\mathbb{D}[2k,j] - \mathbb{D}[2k,j]\} \\
\mathbb{S}^{\text{S.-TPPI}}[2k+1,j] &= -\,\mathrm{Im}\,\{\mathbb{D}[2k,j] + \mathbb{D}[2k+1]\} + \mathrm{i}\,\mathrm{Re}\,\{\mathbb{D}[2k,j] + \mathbb{D}[2k,j]\} \\
&\qquad \text{for } k = 0, \ldots, m//2 - 1 \text{ and } j = 0, \ldots, n
\end{aligned} \tag{3.32}$$

Then, the processing pipeline is the same as the one presented above. In `KLASSEZ` , the function `xfb` of the `processing` module takes care of it, by applying the processing steps in the correct order and adapting the processing to the employed method to acquire the indirect dimension (see appendix A2.2).

4. Denoising techniques

This chapter describes several methods to increase the quality of the NMR spectra in the post-acquisition stage of the analysis by using dedicated algorithms. The theoretical foundations of the proposed approaches are presented from the rigorous mathematical point of view, and a few examples of applications are discussed throughout.

4.1 Signal-to-Noise Ratio

In this section, we will talk about techniques that aim to improve the quality of the spectra. In order to evaluate the performance of these techniques, it is important to define *what is* the quality of an NMR spectrum, and give a method to measure it. In general, a spectrum is considered "good" if the peak are much more intense than the noise level. For this reason, the parameter that is most commonly used for evaluating the quality of a spectrum is the signal-to-noise ratio (SNR), which is a measure of how much the signals "stand out" of the noise. The SNR is defined as:

$$SNR = \frac{S}{2\sigma_N} \tag{4.1}$$

where S is the maximum signal and σ_N is the standard deviation of the noise.

The problem here is that what is intended for "maximum signal" and "standard deviation of the noise" is somehow ambiguous. In fact, there are countless interpretations to this equation, that make the comparison of the data between different works quite difficult. It is therefore of utmost importance to clearly state how to compute the SNR, in order to leave no space for misunderstandings.

From the theoretical point of view, the intensity of the NMR signal is its area. However, for the evaluation of the SNR, this is not a good property to measure. In fact, an intense signal can also be very broad: the linewidth of the signal is determined by its transversal relaxation rate R_2, and can be further modified by chemical exchange, temperature, and other factors. These processes do not affect the intensity of the signal. As a consequence, by increasing the linewidth, the signal will start to appear lower and lower, until it disappears under the noise. This concept is shown in figure 4.1. Therefore, the optimal choice for the calculation of the SNR is the height of the highest peak in the spectrum:

$$S = \max\{\mathbb{S}(\omega)\} \tag{4.2}$$

To calculate σ_N, a portion of the spectrum where there are no signals must be chosen. We will refer to it as an array of r points $\mathbb{y}$. Then, the standard deviation of the noise is computed as:

$$\sigma_N = \frac{1}{\sqrt{r-1}}\sqrt{\sum_{k=0}^{r-1}(\mathbb{y}[k]^2) - \frac{1}{r}\left[\left(\sum_{k=0}^{r-1}\mathbb{y}[k]\right)^2 + \frac{3}{r^2-1}\left(\sum_{k=0}^{r//2-1}(k+1)(\mathbb{y}[r//2+k] - \mathbb{y}[r//2-k-1])\right)^2\right]} \tag{4.3}$$

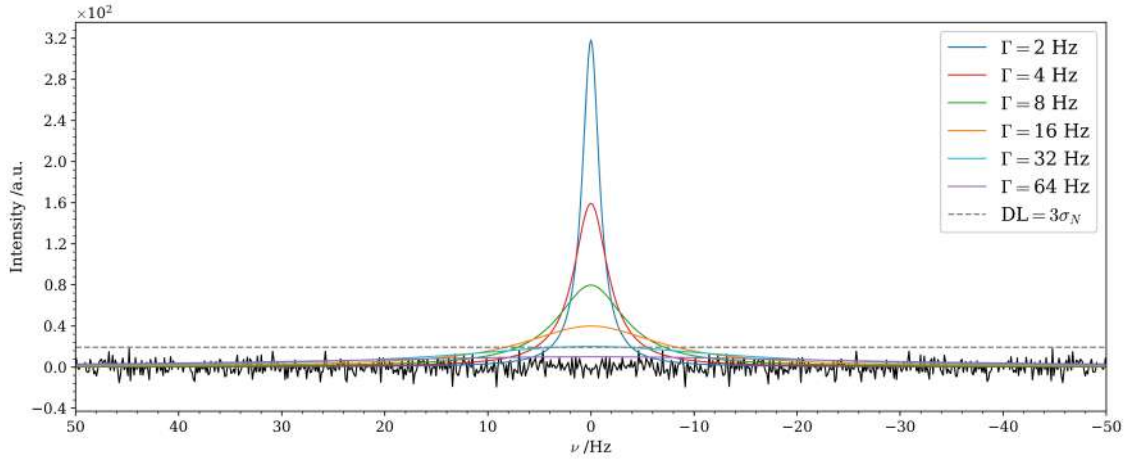


Figure 4.1: A Lorentzian signal centered at 0 Hz with intensity equal to 1 and linewidth $\Gamma = 2$ Hz was simulated. Then, the linewidth of the signal was progressively increased to 4, 8, 16, 32 and 64 Hz, while keeping the intensity constant. As a consequence, the height of the signal decreases. A pure noise trace with $\sigma_N^{\mathrm{FID}} = 0.1$ is shown in the figure as a black trace, and the detection limit of $3 \times \sigma_N$ is reported as a grey dashed line, to make the point of disappearance of the signal more apparent.

which is the equation implemented in Bruker TopSpin command `sino`. This implementation differs from the usual formula employed for the calculation of the standard deviation of a set of data (equation 4.4) for a factor that accounts for a systematic deviation is the noise region, such as a twisted baseline. In other than this conditions, the two equations are basically equivalent.

$$\sigma_N = \frac{1}{\sqrt{r-1}} \sqrt{\sum_{k=0}^{r-1} \left(\langle \mathrm{y} \rangle - \mathrm{y}[k]\right)^2} \tag{4.4}$$

The reason why the factor $2\sigma_N$ appears in the SNR equation is graphically shown in figure 4.2.

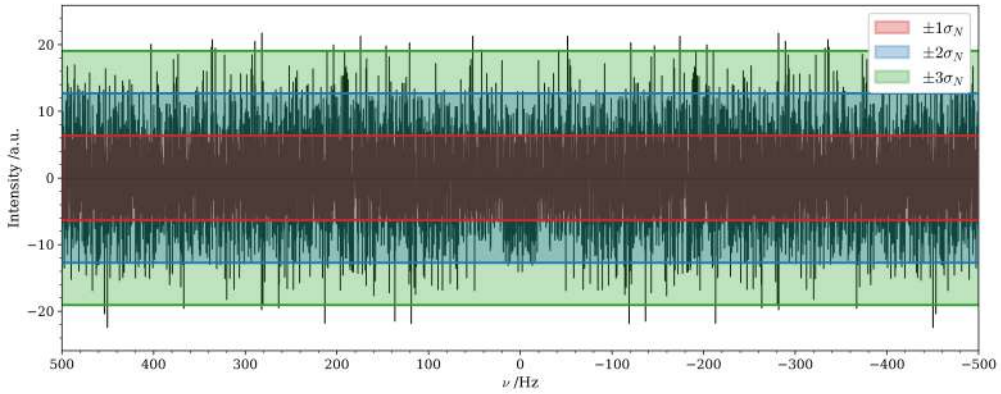


Figure 4.2: A pure noise trace was simulated in the time domain with $\sigma_N^{\mathrm{FID}} = 0.1$ and subsequently Fourier-transformed (black trace). The noise standard deviation σ_N in the spectrum was calculated with equation 4.4. The bands correspondent to σ_N, $2\sigma_N$ and $3\sigma_N$ (the latter being the detection limit) are shown in the figure as blue, red, and green spans, respectively.

4.2 Denoising techniques

The word "denoising" identifies the set of processing methods that allow to extract the information from noisy data. A good denoising method is able to decrease the noise level without altering the shape or the intensity of the peaks, thereby only affecting the SNR. It is important that also the statistical characteristics of the noise are preserved, i.e. a white noise must remain white after the processing.

In literature, there are several denoising algorithms available, that can be applied on either 1D or 2D data, each of them with its pros and cons: wavelet transform[10], Savitzky-Golay filters[11], random QR denoising[12], singular spectrum analysis[40], Cadzow filter[14,41].

In this section I will briefly illustrate the theoretical background behind some of them, in order to set the grounds for the applications presented in the following sections. The denoising algorithms presented herein can be applied either on the FIDs or on the Fourier-transformed spectra, yielding similar results. This can be justified by the fact that the Fourier transform does not add nor remove information from the data, it just presents them from a different perspective. Of course, the use of a window function in the Fourier transform can alter the shape of the peaks, but this is a different story.

4.2.1 Low-rank filtering

A very common and effective denoising technique is based on the singular value decomposition (SVD) of the data. The SVD is the unique factorization of the data matrix $\mathbb{D}$ in the product of two unitary square matrices, $\mathbb{U}$ and $\mathbb{V}$, and of a diagonal matrix $\mathbb{S}$ whose entries are the singular values, sorted in descending order:

$$\mathbb{D} = \mathbb{U}\mathbb{S}\mathbb{V}^H, \quad \mathbb{S} = \begin{pmatrix} s_0 & & & & & \\ & s_1 & & & & \\ & & s_2 & & & \\ & & & \ddots & & \\ & & & & \ddots & \\ & & & & & s_{m-1} \end{pmatrix} \tag{4.5}$$

Due to the way a 2D spectrum is acquired, the number of columns of the $\mathbb{D}$ matrix is much greater than the number of rows. Hence, if $\mathbb{D}$ is a $m \times n$ matrix, its singular value decomposition would yield m singular values.

The SVD of a matrix is strictly connected to its rank, as it can be defined as the number of nonzero singular values. In principle, the rank of the $\mathbb{D}$ matrix should be equal to the number of independent signals that appear in the spectrum. In practice, this is not true because of the presence of noise, which makes the matrix always full-rank. This concept is shown in figure 4.3.

The denoising approach consists on building a new $\mathbb{S}$ matrix that approximates the noiseless one assuming that the n_c singular values are fully associated with signal features, and the others are fully associated with the noise and can thus be set to zero. This new $\mathbb{S}'$ matrix is then used with the same $\mathbb{U}$ and $\mathbb{V}$ matrices to reconstruct the denoised $\mathbb{D}$ data matrix, $\mathbb{D}'$:

$$\mathbb{D}' = \mathbb{U}\mathbb{S}'\mathbb{V}^H, \quad \mathbb{S}' = \begin{pmatrix} s_0 & & & & & \\ & \ddots & & & & \\ & & s_{n_c-1} & & & \\ & & & 0 & & \\ & & & & \ddots & \\ & & & & & 0 \end{pmatrix} \tag{4.6}$$

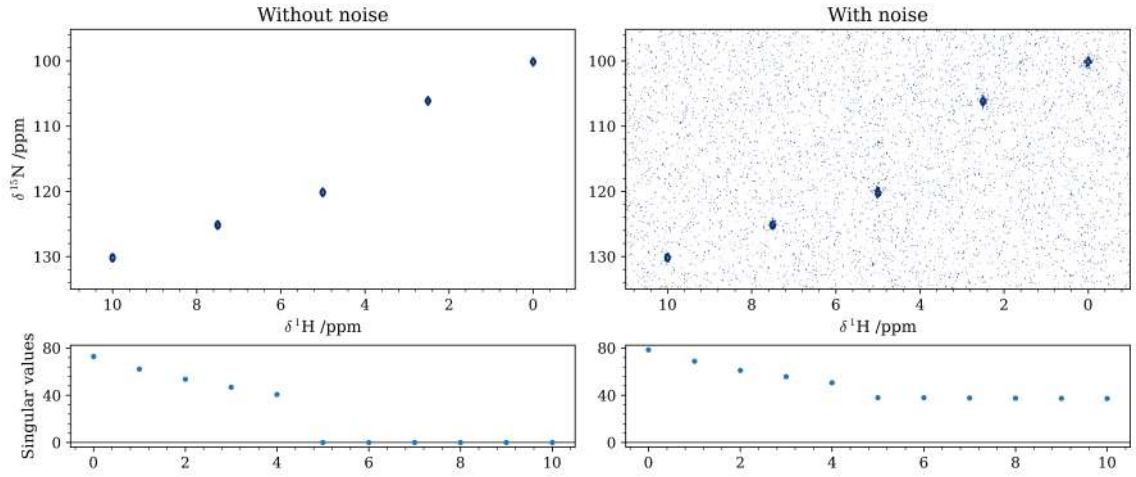


Figure 4.3: The same 2D simulated spectrum of figure 2.4 is shown as blue trace in the top left panel. As expected, its SVD has only five nonzero singular values (bottom left panel). Then, a noise array with $\sigma_N = 0.2$ was added to the pure simulated spectrum, obtaining the trace in the top right panel. The SVD of the noisy spectrum is only slightly different than the original one for the first five singular values, but it shows a significant offset from zero for all the remaining ones.

The whole algorithm is implemented in the `lrd` function of the `processing` module of `KLASSEZ` , as shown in listing 4.1.

Listing 4.1: Function to perform the low-rank filtering from the `processing` module of `KLASSEZ` .

```
        CS_f = np.zeros_like(input_data).astype(input_data.dtype)
        for j in range(nds):
            CS_f[j] = C_f[j] @ S_f[j]

    # Reshape if no matrix augmentation is performed
    if nds == 1:
        CS_f = CS_f[0]
        C_f = C_f[0]
        S_f = S_f[0]

    print('\n**********************************************************\n')

    return CS_f, C_f, S_f

def lrd(data, nc):
    """
```

```
    fid_den = kz.processing.lrd(fid, n_c)
```

This approach proves to be very convenient in case of additive noise with very good results, even when the spectra have a relatively broad dynamic range (see figure 4.4). Furthermore, it can also be applied on 1D spectra with a prior conversion to a Hankel (or Toeplitz) matrix. This goes under the name of Cadzow filter. However, in presence of multiplicative noise, this method cannot be applied. The contribution of multiplicative noise to the spectrum cannot be separated from the pure signals in terms of singular values.

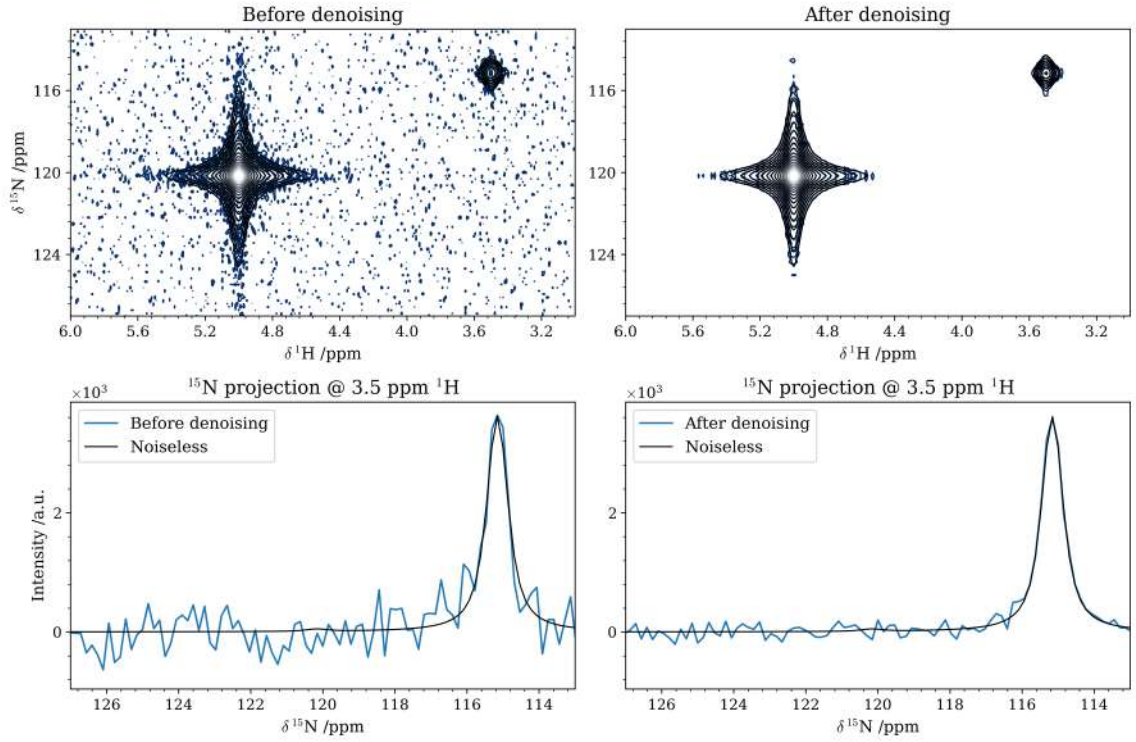


Figure 4.4: Two 2D Voigt signals were simulated with the following parameters: $\delta_1^a = 120\,\text{ppm}$, $\delta_2^a = 5.0\,\text{ppm}$, $\delta_1^b = 115\,\text{ppm}$, $\delta_2^b = 3.5\,\text{ppm}$, $K^a = 25$, $K^b = 1$, $\Gamma_1^{a,b} = \Gamma_2^{a,b} = 50\,\text{Hz}$, $\beta^a = \beta^b = 0.2$. Then, noise was added with $\sigma_N = 0.4$: the resulting spectrum is shown in the top-left panel, in blue. The result of the denoising with the low-rank filtering, using $n_c = 2$, is shown in the top-right panel, in blue as well. For comparison, the original, noiseless spectrum is reported in both panel in black. All contours are drawn starting from the 0.5% of the height of signal a. The denoising approach does not alter the linewidth nor the intensity of the peaks, and it is also beneficial to correct the lineshape, as also shown in the indirect dimension projection of signal b before and after the processing (bottom panels, left and right respectively).

4.2.2 Multivariate Curve Resolution

Multivariate Curve Resolution (MCR) is a chemometric method usually employed to recover pure components information from data acquired on complex mixtures, such as UV-visible spectra and chromatograms. Its original field of application was then expanded to include a wide plethora of analytical techniques. In addition to the straightforward separation purpose, MCR can also be used for denoising[15,42].

Following the original chemometric formalism, MCR aims to decompose the $m \times n$ data matrix $\mathbb{D}$ in a "concentration" matrix $\mathbb{C}$ and a "spectra" matrix $\mathbb{S}$, as shown in equation 4.7. The matrix $\mathbb{E}$ contains the part of the data that are not reproduced by the factorization.

$$\mathbb{D} = \mathbb{C}\mathbb{S} + \mathbb{E} \tag{4.7}$$

The MCR workflow can be separated in two parts. First, an initial guess of the concentration and spectra matrix, $\mathbb{C}_0$ and $\mathbb{S}_0$, must be generated, through the *SIMPLISMA* algorithm[43]. Then, an optimization step follows, in which the unwanted features of the data are removed by iteratively filling the $\mathbb{E}$ through an Alternating Least-Squares fit.

The python implementation of MCR in `KLASSEZ` is fully reported in appendix A3.

SIMPLISMA

The SIMPLISMA algorithm can be used to obtain an initial guess of either the $\mathbb{C}$ or $\mathbb{S}$ matrix through the research of the n_c *purest components* of $\mathbb{D}$, where n_c is a user-defined parameter. The purest components of $\mathbb{D}$ are the n_c rows or columns that are responsible for the biggest variations in the data variance. In principle, this is the same problem that a principal component analysis (PCA) is able to solve. However, a PCA rearranges the dataset into a linear combination of its feature without any user control, thus making impossible to establish a closed-form relationship with the original dataset. In contrast, the use of SIMPLISMA allows to preserve the structure of the original data.

We will present the SIMPLISMA algorithm for the estimation of the $n_c \times n$ matrix $\mathbb{S}$. The estimation of the $m \times n_c$ matrix $\mathbb{C}$ can be performed by applying it on $\mathbb{D}^T$. The procedure starts with the calculation of the mean and the standard deviation of each row of $\mathbb{D}$, which are stored in the $\mathbb{m}$ and $\mathbb{s}$ arrays, respectively.

$$\mathbb{m}[i] = \frac{1}{n}\sum_{j=0}^{n-1} \mathbb{D}[i,j] \qquad \text{for } i = 0, \ldots, m-1 \quad (4.8)$$

$$\mathbb{s}[i] = \sqrt{\frac{1}{n}\sum_{j=0}^{n-1} \left(\mathbb{D}[i,j] - \mathbb{m}[i]\right)^2} \quad (4.9)$$

The purity of a component is assessed by the ratio between its standard deviation and its mean, which takes the name of *variation coefficient*. At this point, the *purity spectrum* $\mathbb{p}^{(0)}$ can be defined as the variation coefficients of each row of $\mathbb{D}$, as shown in equation 4.10. It must be noted that transients with high levels of noise would have mean close to zero. As a consequence, the standard deviation-to-mean ratio would rapidly reach very high values. In these cases, the α parameter acts as a damping factor, thus preventing the overflow. The value of α is set to be a fraction $f \in [0,1]$ of the highest value in $\mathbb{m}$ (typically, $f = 0.10$).

$$\mathbb{p}^{(0)}[i] = \frac{\mathbb{s}[i]}{\mathbb{m}[i] + \alpha}, \quad \alpha = f \max\{\mathbb{m}\} \qquad \text{for } i = 0, \ldots, m-1 \quad (4.10)$$

The independency between variables is expressed by a weighting array $\mathbb{w}^{(c)}$, where the c superscript identifies the purest spectrum at which is referred to. The first weight, $\mathbb{w}^{(1)}$, is defined as:

$$\mathbb{w}^{(1)}[i] = \frac{\mathbb{m}^2[i] + \mathbb{s}^2[i]}{\mathbb{s}^2[i] + \left(\mathbb{m}[i] + \alpha\right)^2} \quad (4.11)$$

The first purity spectrum, $\mathbb{p}^{(1)}$, is simply the purity spectrum $\mathbb{p}^{(0)}$ corrected by the first weight, and the correspondent first purest variable ρ_1 is the index of the maximum entry of $\mathbb{p}^{(1)}$:

$$\mathbb{p}^{(1)} = \mathbb{w}^{(1)} \odot \mathbb{p}^{(0)}, \quad \rho_1 = \max_i \left\{\mathbb{p}^{(1)}\right\} \quad (4.12)$$

It should be pointed out that if $\alpha = 0$, the first weight $\mathbb{w}^{(1)}$ is equal to 1 for all its entries. Hence, when $\alpha \neq 0$, it should be interpreted as a correction factor for noisy spectra.

The second purest variable, ρ_2, will be the most independent variable from ρ_1. To compute it, we define the *correlation around the origin* matrix $\mathbb{Q}$ as in equation 4.13, where $\mathbb{D}_\lambda$ is a

normalized $\mathbb{D}$ matrix. It can be shown that the normalization factors λ_i are equal to the length of the i-th row of $\mathbb{D}$[44], thus making the $\mathbb{Q}$ matrix only proportional to the variation coefficient.

$$\mathbb{Q} = \frac{1}{n}\mathbb{D}_\lambda \mathbb{D}_\lambda^T, \quad \mathbb{D}_\lambda[i,j] = \frac{\mathbb{D}[i,j]}{\lambda_i} = \frac{\mathbb{D}[i,j]}{\sqrt{\mathbb{s}^2[i] + (\mathbb{m}[i] + \alpha)^2}} \tag{4.13}$$

The elements of the weighting array $\mathbb{w}^{(c)}$ for $c > 1$, needed to compute the c-th purest spectrum, are given by the determinants of the set of matrices $\mathbb{W}_i^{(c)}$, of dimensions $c \times c$, structured as in equation 4.14. These set of weighting arrays aim to remove the contributions to the variance of $\mathbb{D}$ that come from the $c-1$ purest components detected previously.

$$\mathbb{w}^{(c)}[i] = \det(\mathbb{W}_i) = \begin{vmatrix} \mathbb{Q}[i,i] & \mathbb{Q}[i,\rho_1] & \mathbb{Q}[i,\rho_2] & \dots & \mathbb{Q}[i,\rho_{c-1}] \\ \mathbb{Q}[\rho_1,i] & \mathbb{Q}[\rho_1,\rho_1] & \mathbb{Q}[\rho_1,\rho_2] & \dots & \mathbb{Q}[\rho_1,\rho_{c-1}] \\ \mathbb{Q}[\rho_2,i] & \mathbb{Q}[\rho_2,\rho_1] & \mathbb{Q}[\rho_2,\rho_2] & \dots & \mathbb{Q}[\rho_2,\rho_{c-1}] \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbb{Q}[\rho_{c-1},i] & \mathbb{Q}[\rho_{c-1},\rho_1] & \mathbb{Q}[\rho_{c-1},\rho_2] & \dots & \mathbb{Q}[\rho_{c-1},\rho_{c-1}] \end{vmatrix} \tag{4.14}$$

At this point, equation 4.12 can be generalized as follows:

$$\mathbb{p}^{(c)} = \mathbb{w}^{(c)} \odot \mathbb{p}^{(0)}, \quad \rho_c = \max_i \left\{\mathbb{p}^{(c)}\right\} \qquad \text{for } c = 1, \dots, n_c \tag{4.15}$$

The initial guess for the "spectra" matrix, $\mathbb{S}_0$, is computed by filling its rows with the n_c purest components of $\mathbb{D}$:

$$\mathbb{S}[c-1, j] = \mathbb{D}[\rho_c, j] \qquad \text{for } c = 1, \dots, n_c \text{ and } j = 0, \dots, n-1 \tag{4.16}$$

The initial guess for the "concentration" matrix $\mathbb{C}_0$ can be derived from the new $\mathbb{S}_0$ matrix as

$$\mathbb{C}_0 = \mathbb{D}\mathbb{S}^+ \tag{4.17}$$

MCR-ALS

The MCR optimization consists on iteratively calculate $\mathbb{C}$ and $\mathbb{S}$, in order to discard what cannot be represented by the MCR model into a residual matrix $\mathbb{E}$. This matrix should be initialized as

$$\mathbb{E}_0 = \mathbb{D} - \mathbb{C}_0\mathbb{S}_0 \tag{4.18}$$

At this point the actual Alternating Least Squares (ALS) procedure can start. Let us mark with a k subscript the three matrices $\mathbb{C}$, $\mathbb{S}$ and $\mathbb{E}$ at the k-th iteration of the optimization. Each iteration consists of the three steps listed in equation 4.19.

$$\begin{aligned} \mathbb{S}_k &= \mathbb{C}_{k-1}^+ \mathbb{D} \\ \mathbb{C}_k &= \mathbb{D}\mathbb{S}_k^+ \\ \mathbb{E}_k &= \mathbb{D} - \mathbb{C}_k\mathbb{S}_k \end{aligned} \tag{4.19}$$

The fit is stopped either when a certain number of iteration has been reached, or when a given arrest criterion is satisfied. Given a threshold value θ, the arrest criterion is defined as:

$$R_k^{\mathbb{C}} = \|\mathbb{C}_k - \mathbb{C}_{k-1}\|_F \leq \theta \quad \& \quad R_k^{\mathbb{S}} = \|\mathbb{S}_k - \mathbb{S}_{k-1}\|_F \leq \theta \tag{4.20}$$

When these conditions are satisfied, the $\mathbb{C}$ and $\mathbb{S}$ matrices are considered to not vary anymore.

Finally, the denoised spectrum $\mathbb{D}'$ can be computed as

$$\mathbb{D}' = \hat{\mathbb{C}}\hat{\mathbb{S}} \tag{4.21}$$

where $\hat{\mathbb{C}}$ and $\hat{\mathbb{S}}$ are the result of the optimization.

Dataset augmentation

One of the biggest advantages of MCR is that it can handle multiple data matrices simultaneously, and therefore it is ideal to co-process series of spectra. This feature is implemented by stacking the matrices with an axis in common, i.e. imposing the datasets to share a dimension.

If all the datasets have the same dimensions, the stacking may be done horizontally (same columns) or vertically (same rows). This means that, if one wanted to co-process z datasets, each of them of dimensions $m \times n$, in the former case one would obtain a final $\mathbb{D}$ matrix of dimensions $m \times (zn)$, whereas in the latter case the final dimensions would be $(zm) \times n$. On the other hand, if the datasets share only one dimension, then the stacking direction is a forced choice.

The stacking direction has a direct impact on the outcome of the method: in the case of horizontal stacking, a single column of $\mathbb{C}$ matrix will be a purest component of all the z datasets, whereas the rows of $\mathbb{S}$ will necessary have contributions from each of the datasets. This situation is reversed in the case of vertical stacking. One could also choose to assemble the datasets in two directions simultaneously, rather than a in a single dimension. The optimization is performed on these augmented matrices, and we have proven it is particularly beneficial for spectra that have a low SNR.

Formally speaking, let us suppose that we want to co-process a set of z $\mathbb{D}_k$ datasets, for $k = 0, \ldots, z-1$. We define a grid of dimensions $m_z \times n_z$ such that $m_z n_z = z$, that represents the shape of the augmented $\mathbb{D}$ matrix, in the form of a *positioning matrix* $\mathbb{P}$. The entries of $\mathbb{P}$ are the indices of the datasets, arranged in the positions they have to assume in the grid. At this point, a set of *mask matrices* $\mathbb{M}_k$ of dimensions $m_z \times n_z$ are computed, whose entries are all 0 except for a 1 in the position of k in $\mathbb{P}$. The $\mathbb{D}_k$ in the correct position of the grid is then computed as the Kronecker product of itself with the correspondent mask matrix. The final augmented $\mathbb{D}$ matrix is then given by the sum of all these matrices:

$$\mathbb{D} = \sum_{k=0}^{z-1} \mathbb{M}_k \otimes \mathbb{D}_k \tag{4.22}$$

This is better visualized with an example. Let us suppose that we want to achieve the following arrangement with 6 datasets:

$$\mathbb{D} = \begin{pmatrix} \mathbb{D}_2 & \mathbb{D}_4 & \mathbb{D}_0 \\ \mathbb{D}_5 & \mathbb{D}_1 & \mathbb{D}_3 \end{pmatrix} \tag{4.23}$$

The positioning matrix we have to write is:

$$\mathbb{P} = \begin{pmatrix} 2 & 4 & 0 \\ 5 & 1 & 3 \end{pmatrix} \tag{4.24}$$

The six mask matrices therefore will be:

$$\begin{aligned} \mathbb{M}_0 &= \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \quad \mathbb{M}_1 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \quad \mathbb{M}_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \\ \mathbb{M}_3 &= \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \mathbb{M}_4 = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad \mathbb{M}_5 = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} \end{aligned} \tag{4.25}$$

Therefore the Kronecker product of the masks with the datasets gives rise to the following block matrices:

$$\begin{aligned} \mathbb{M}_0 \otimes \mathbb{D}_0 &= \begin{pmatrix} \mathbb{0} & \mathbb{0} & \mathbb{D}_0 \\ \mathbb{0} & \mathbb{0} & \mathbb{0} \end{pmatrix} \quad \mathbb{M}_1 \otimes \mathbb{D}_1 = \begin{pmatrix} \mathbb{0} & \mathbb{0} & \mathbb{0} \\ \mathbb{0} & \mathbb{D}_1 & \mathbb{0} \end{pmatrix} \quad \mathbb{M}_2 \otimes \mathbb{D}_2 = \begin{pmatrix} \mathbb{D}_2 & \mathbb{0} & \mathbb{0} \\ \mathbb{0} & \mathbb{0} & \mathbb{0} \end{pmatrix} \\ \mathbb{M}_3 \otimes \mathbb{D}_3 &= \begin{pmatrix} \mathbb{0} & \mathbb{0} & \mathbb{0} \\ \mathbb{0} & \mathbb{0} & \mathbb{D}_3 \end{pmatrix} \quad \mathbb{M}_4 \otimes \mathbb{D}_4 = \begin{pmatrix} \mathbb{0} & \mathbb{D}_4 & \mathbb{0} \\ \mathbb{0} & \mathbb{0} & \mathbb{0} \end{pmatrix} \quad \mathbb{M}_5 \otimes \mathbb{D}_5 = \begin{pmatrix} \mathbb{0} & \mathbb{0} & \mathbb{0} \\ \mathbb{D}_5 & \mathbb{0} & \mathbb{0} \end{pmatrix} \end{aligned} \tag{4.26}$$

whose sum gives exactly the structure in equation 4.23.

In the simpler case of horizontal stacking, $m_z = 1$, $n_z = z$, and

$$\mathbb{P}[0, j] = j, \quad \mathbb{M}_k[0, j] = \delta_{\mathbb{P}[0,j],k} \qquad \text{for } j = 0, \ldots, z-1 \quad (4.27)$$

Equivalently, for vertical stacking, $m_z = z$, $n_z = 1$, and

$$\mathbb{P}[i, 0] = i, \quad \mathbb{M}_k[i, 0] = \delta_{\mathbb{P}[i,0],k} \qquad \text{for } i = 0, \ldots, z-1 \quad (4.28)$$

Although appearing very cumbersome, this approach offers the highest flexibility in how to arrange the datasets for a MCR coprocessing, which also adapts to the trivial row-wise and column-wise matrix augmentation.

4.3 Applications

4.3.1 MCR on solution NMR spectra

We have previously demonstrated that the application of Multivariate Curve Resolution can be extremely beneficial for the improvement of the quality of 2D solid-state NMR data[15,42]. As discussed in Publication **??**, the use of MCR was crucial to achieve the required data quality to support the conclusions, i.e. that in the investigated composite material lysozyme is trapped inside the silica scaffold by mostly steric effects, and that it interacts with the inorganic surface without altering its three-dimensional structure.

However, when we tried to use the same procedure in solution NMR, we noticed that the presence of multiplicative noise makes it completely useless. In fact, compared to solid-state NMR, t_1-noise covers a much more relevant role in the description of the not-signal component of the data. Because of its nature, which tightly bounds it to the signal itself, it cannot be efficiently separated by the MCR algorithm, resulting in an overall worsening of the spectrum quality. This effect is enhanced in the presence of the solvent ridge-like artifact: in these conditions, the SIMPLISMA algorithm is unable to recognize the signals as signals, no matter the chosen number of components.

The reason for such behavior is found in the MCR model described in equation 4.7, which assumes that the noise is an additive component, thus independent from the pure signal spectrum. Furthermore, each peak of the 2D FID can be consider as the product between an indirect evolution and a direct acquisition FID, as previously described in section 2.2. The idea behind this simulation is a perfect mirror image of the MCR model. In this framework, the problem of the ridge artifact is that it is not described by an evolution in the indirect dimension. As a result, the solvent appears as an offset-like component in the columns of the $\mathbb{C}$ matrix, hence the separation from the actual signals fails (figure 4.5).

In experiments that do not exhibit this kind of artifacts, such as direct $^{13}C-^{13}C$ correlation spectra, or CON spectra[45], MCR can still be successfully applied for denoising (figure 4.6). Therefore, we concluded that MCR cannot be employed for the denoising of 2D solution NMR data, unless the algorithm is modified in order to account for these problems.

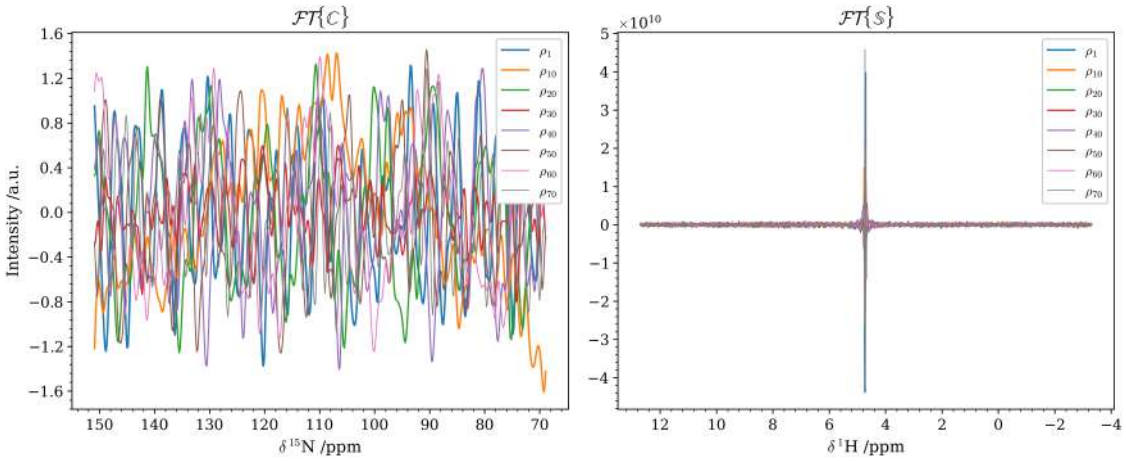


Figure 4.5: The MCR denoising approach was tested on the ^{1}H−^{15}N HSQC spectrum of lysozyme reported in figure 2.5b, using $n_c = 70$. 10 of the optimized purest spectra ($\mathbb{S}[\rho_k, :]$) and their correspondent indirect evolutions ($\mathbb{C}[:, \rho_k]$) are shown, Fourier-transformed, in the right and left panel of the figure. It is apparent that the only contribution present in the $\mathbb{S}$ matrix comes from the water signal, which does not have a dependence from the indirect frequency. Hence, the $\mathbb{C}$ matrix contains basically only noise.

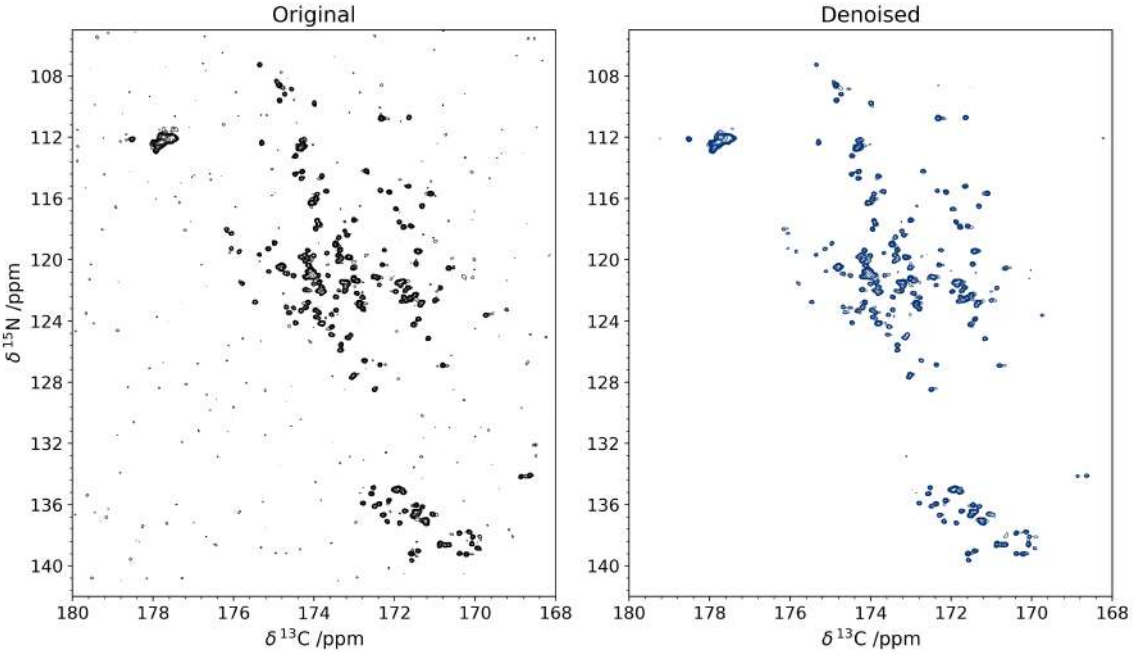


Figure 4.6: ^{15}N−^{13}C CON spectrum of the CD4 protein at 700 MHz ^{1}H Larmor frequency before (left, black trace) and after denoising (right, blue trace). The MCR algorithm was applied by setting $n_c = 85$. The spectra are displayed with the same contour levels for comparison. The decrease of noise level appears evident upon visual inspection.

4.3.2 Denoising series of spectra

There are NMR experiments where a certain property of the sample magnetization, that is not the chemical shift, is evolved during the indirect dimension by either means of the pulse sequence or because of a physico-chemical change of the sample properties. As this class of dataset do not encode for a frequency in the indirect dimension, they are Fourier-transformed only on the direct dimension, and therefore are referred to as *pseudo-2D* experiments. Examples of such sequences include inversion recovery, CPMG, DOSY.

The information that can be gathered from these data is twofold. On one side, there are the spectral features that appear in a normal 1D spectrum, as a function of the undergoing selection process. In parallel, the evolution of the peak intensities (or other parameters) tells about the changes that occur in the sample during the indirect dimension. These trends can be fitted with an appropriate model to gain quantitative insights on the dynamical properties of the system.

Recording this kind of data has one very strong limitation: the acquisition time, that must be kept within reasonable limits. This means that the total number of scans must be distributed between the number of transients to be acquired, and the number of scans per transients. As these two factors affect either the sensitivity of the single experiment or the accuracy of the fit, it follows that a thoughtful balance must be chosen in order to obtain the optimal results.

We recently investigated the role that a denoising algorithm based on low-rank thresholding (section 4.2.1) can have in this framework by studying the cross-polarization evolution of a simulated spectrum[46] (publication **??**). The conclusions of our work allow to set guidelines on how to distribute the total instrument time over the number of experiment to record, in order to maximize the accuracy of the subsequent analyses.

However, it should be noted that in absence of other priors or regularizations, the outcome of low-rank denoising and MCR is the same[47]. Furthermore, the whole MCR workflow is generally faster than the computation of an SVD, and can also be used as soft model for the spectral features and its evolutions, in line with its original application field.

To show another example of the increased accuracy in derivating chemical parameters from NMR data employing this denoising approach, we simulated the real-time NMR monitoring of the oxidation of toluene (R, *reactant*) to benzoic acid (P, product). The reaction follows a pseudo-first order kinetics, therefore we used the relations:

$$\frac{\mathrm{d}[R]}{\mathrm{d}t} = k[R] \quad \Rightarrow \quad [R](t) = [R](0)\, e^{-kt} \tag{4.29}$$

where $t = 0$ is the time of the mixing of the reactant with the oxidizing agent. Since at this time there is no product in the reaction vessel, and assuming that at $t = \infty$ the reactant is completely converted into the product (i.e. $[P](\infty) = [R](0)$), the concentration of product as function of t can be expressed as:

$$[P](t) = [P](\infty) - R(t) = R(0)\big(1 - e^{-\kappa t}\big) \tag{4.30}$$

We arbitrarily set the kinetic constant κ to $2 \cdot 10^{-3}\,\mathrm{s}^{-1}$ to simulate approximately 1 hour of measurement time. With these parameters we obtain the concentrations reported in the left panel of figure 4.7.

We simulated the ^{1}H-NMR spectra of toluene and benzoic acid at 700 MHz by taking the chemical shifts, relative intensities and coupling constants of the signals from literature data. The resulting spectra are shown in the right column of figure 4.7.

We simulate the acquisition of a series of 1D NMR spectra that covers one hour of reaction time, and arrange them in a pseudo-2D experiment. For this simulation, we assume that the

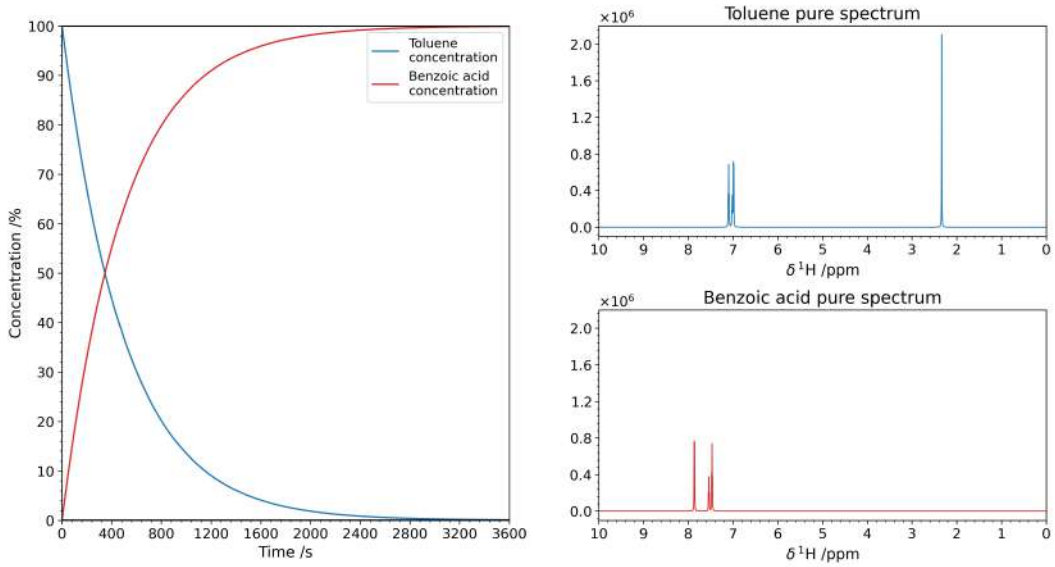


Figure 4.7: Left Evolution of the concentration of toluene (blue trace) and benzoic acid (red trace) as function of time. **Right** Simulated spectra of toluene (top) and benzoic acid (bottom).

reagent and the product are in slow exchange and that the intensity of the signals does not vary during the acquisition of the FID.

Accounting for 4 s of acquisition time and 1 s of recycle delay, the maximum number of scans that it is possible to record in one hour is 720. We could sum up n_s of these scans to improve the SNR: hence, the number of experiments M that will build the pseudo-2D FID is $720/n_s$.

The FID of the mixture at time t was simulated by summing up the FIDs of the pure components, weighted by their respective concentrations at time t, i.e.:

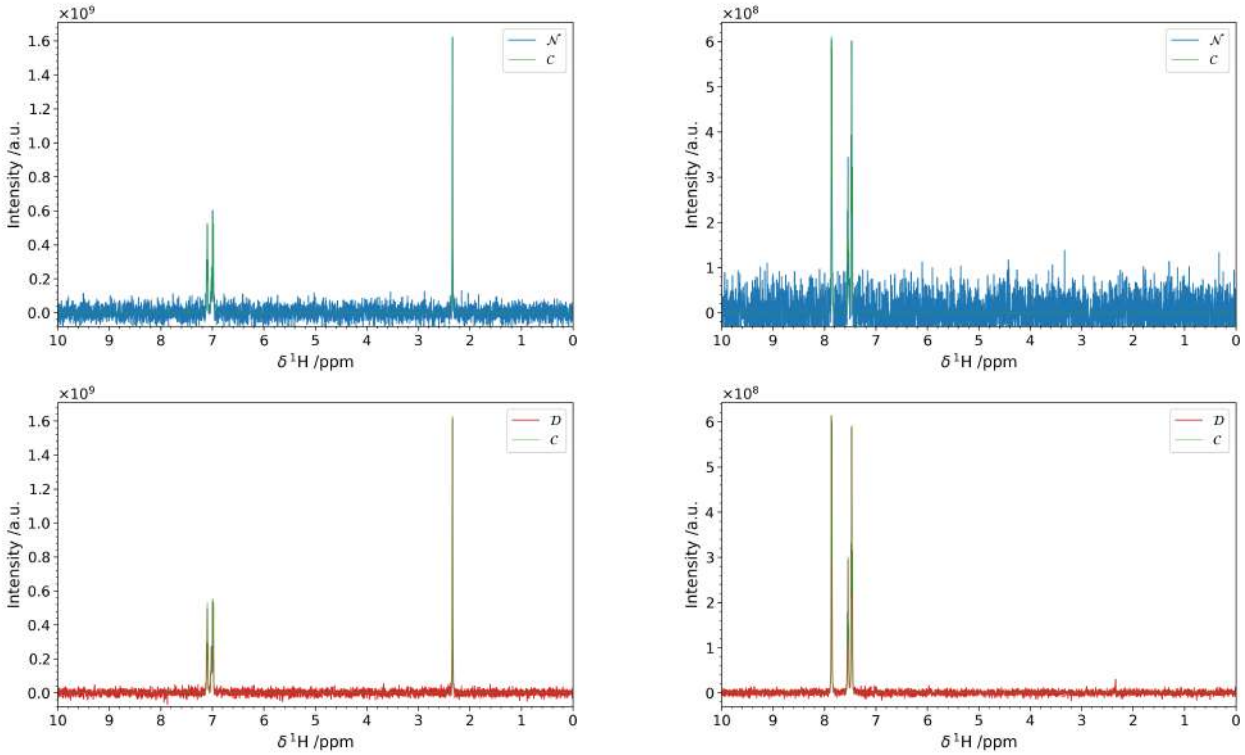
$$s(t) = [R](t)s_R + [P](t)s_P \tag{4.31}$$

The denoising procedure we apply on the pseudo-2D FID is based on Multivariate Curve Resolution[15]. We set the number of components to look for to 2: reactant and product.

Both the noisy ($\mathcal{N}$) and denoised ($\mathcal{D}$) spectra were processed by taking the Fourier transform of the direct dimension without prior apodization. The first and the last spectra of the pseudo-2D datasets (i.e. the spectra of pure reactant and of pure product, respectively) are reported in figure 4.8, whereas a stacked plot of the whole reaction is shown in figure 4.9 (only 16 experiments are displayed for visual clarity).

We chose as signature peaks of the reagents the methyl resonance of toluene ($\delta = 2.34$ ppm) and the *ortho*-hydrogens of benzoic acid ($\delta = 7.86$ ppm), as these are the most isolated signals. Their integrals were used to monitor the variation of the mixture composition (figure 4.10, after rescaling).

The denoising method we applied proves to be effective in increasing the SNR without altering the signal intensities and preserving the time-dependent information. Furthermore, as it can be noted by observing figure 4.11, it allows to improve the time-resolution of the experiment of a factor 8 with similar data quality.



(a) Toluene **(b)** Benzoic acid

Figure 4.8: Spectrum of pure reactant (a) and of pure product (b) extracted from the pseudo-2D dataset before (blue trace) and after the denoising (red trace).

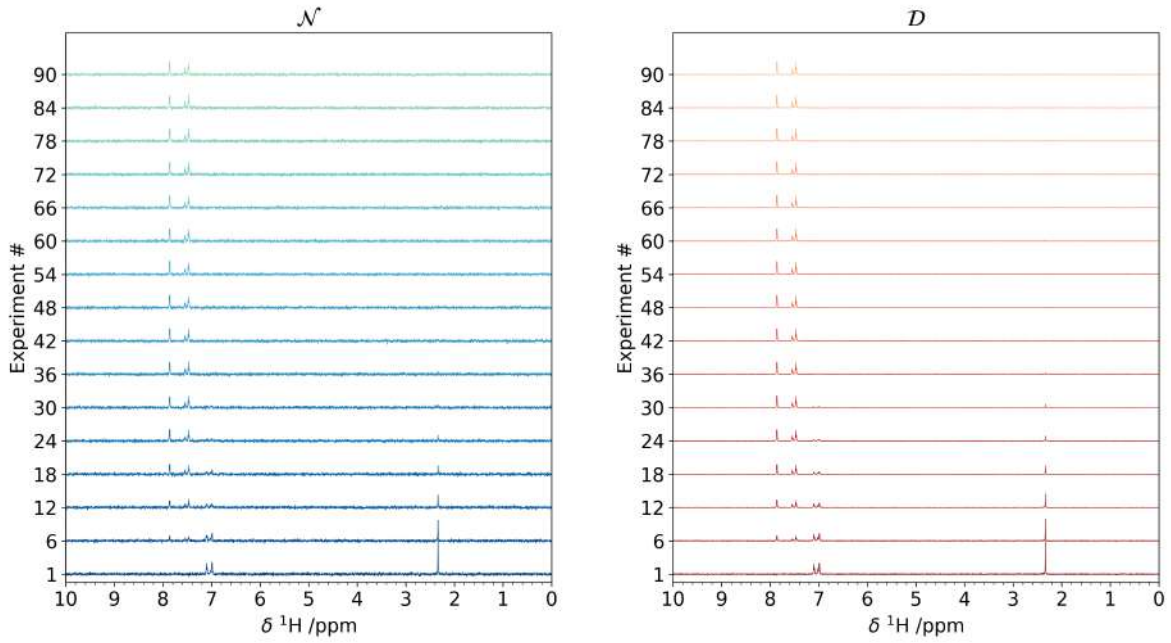


Figure 4.9: Stacked plot of 16 out of 90 experiments extracted from the pseudo-2D dataset with applied noise ($\mathcal{N}$, left panel) and after the denoising procedure ($\mathcal{D}$, right panel).

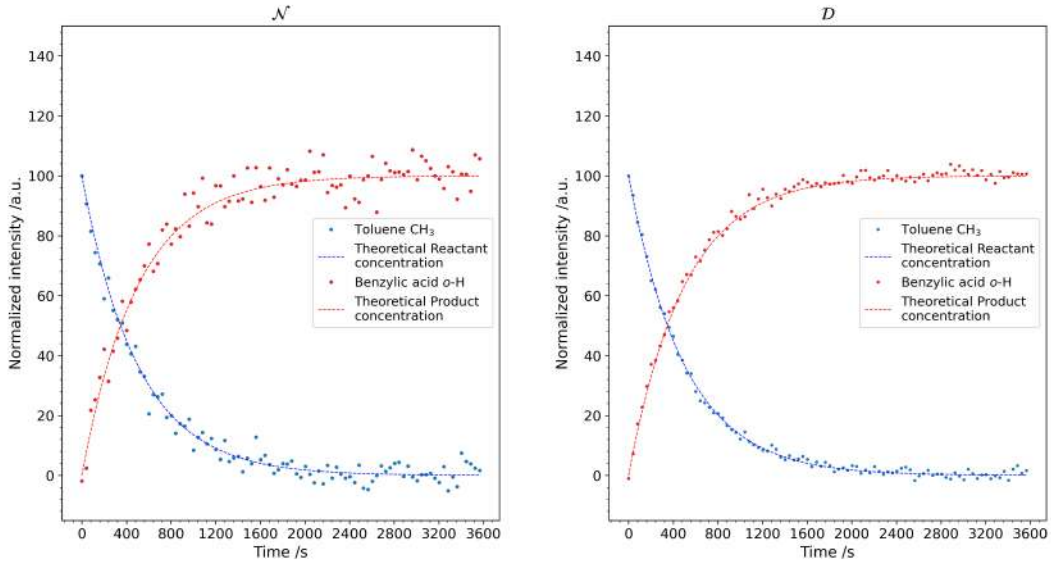


Figure 4.10: Percentage of reactant (blue dots) and product (red dots) as a function of the reaction time. For reference, the theoretical concentration trends (i.e. the ones in the left panel of figure 4.7) are shown, superimposed, as dashed lines.

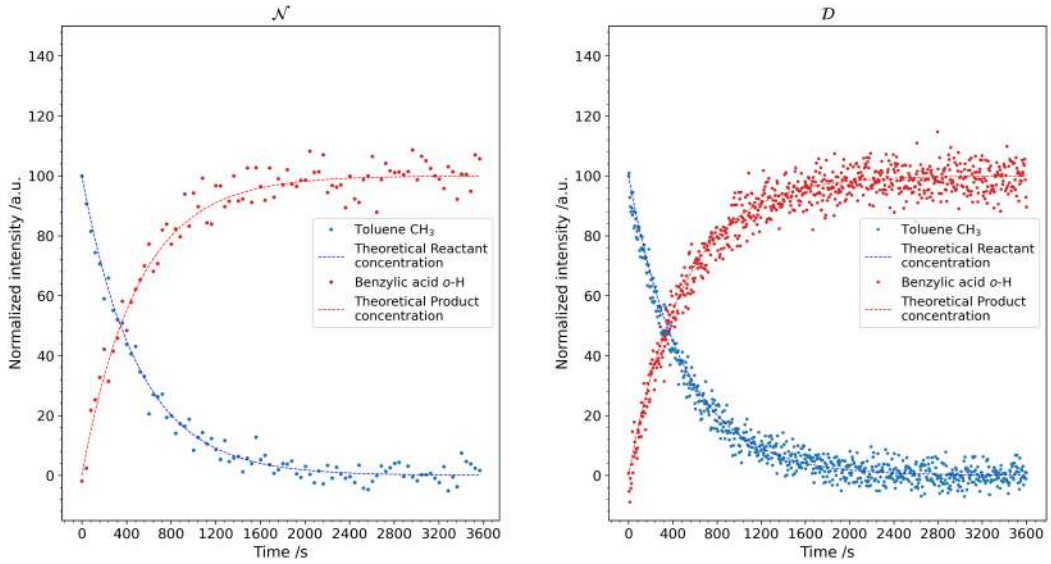


Figure 4.11: Percentage of reactant (blue dots) and product (red dots) as a function of the reaction time **Left** without denoising ($\mathcal{N}$) averaging 8 scans per experiment and **Right** with denoising obtained with a single scan per experiment ($\mathcal{D}$). For reference, the theoretical concentration trends (i.e. the ones in the left panel of figure 4.7) are shown, superimposed, as dashed lines.

5. Deconvolutions

In this chapter, the problem of fitting the NMR spectrum with appropriate models will be addressed. After laying the theoretical foundations of the technique, the `KLASSEZ` implementation of the fitting routines will be presented. A few example of applications of the proposed methods will be shown as well.

5.1 Theoretical backgrounds

5.1.1 Why fitting a spectrum is convenient?

The chemical information on the structure of the sample are encoded in the position of the peaks of the NMR spectrum, i.e. the chemical shift, which depends on the characteristics of the surrounding nuclei, and in its intensity, that depends on the number equivalent nuclei that contribute to the signal, and can be a function of the properties selected by the pulse sequence. The dynamic properties of the system, instead, reside in the linewidth of the peak, which depends in principle only on the transversal relaxation time T_2 and - if any - on chemical exchange processes, but can be affected by other factors like the field inhomogeneity, temperature, and instrumental imperfections (figure 2.1).

In the case of isolated and sharp peaks, these quantities can be easily measured by visual inspection and very simple interfaces. However, the difficulty of the task escalates rather quickly when the spectra become crowded, or when the SNR is low, or both. In fact, when two (or more) peaks overlap each other, evaluating the linewidth and the intensity is not trivial at all, as the classic method will include contributions coming from the other component. This problem becomes even more important when the chemical shift difference between the peaks falls under the resolution limit (i.e. $\Delta\nu < 2\Gamma$, chapter 1). As shown in figure 5.1, when two neighboring signals are close to each other, the superimposition region will be shared in the computation of the integral. Even if the integration region is limited to the valley between the two peaks, the computed intensity will not be correct. Furthermore, finding the exact center of a broad peak is a very error-prone procedure.

In these situations, it might be beneficial to fit the spectrum with an appropriate lineshape model. This approach is known as *spectra deconvolution* (although it is not an actual deconvolution from the rigorous mathematical perspective), and it comes with several advantages. As the Voigt model described in section 2.1 has proven to be effective for simulation of NMR signals, it is natural to use it also for deconvolution. Since the simulation of an NMR signal is a parametric function of chemical shift, linewidth and the intensity, at the end of the fit these quantities are returned in a single run with very high accuracy. Phase and baseline distortions can in principle hamper the accurate computation of the peak parameters that would deconvolve the spectra, up to the point where the optimization does not succeed. However, these distortions can be easily taken into account by adding a parametric correction in the model function, such as the phase angles ϕ_0 and ϕ_1 of equation 3.21, and the coefficients of a low-rank

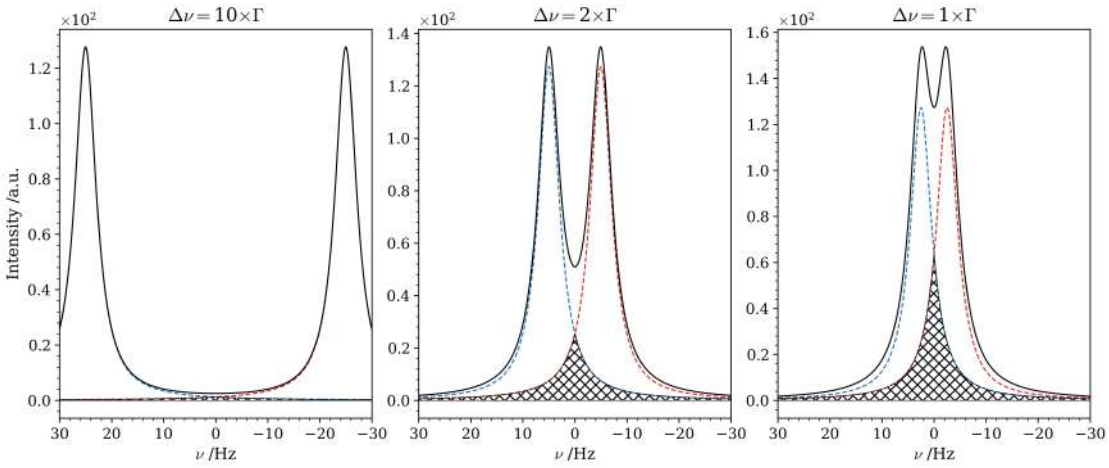


Figure 5.1: Two signal with equal intensity $K = 1$ and same linewidth $\Gamma = 5\,\mathrm{Hz}$ are simulated. Their difference in resonance frequency is set to be five times the resolution limit (left panel), to exactly the resolution limit (middle panel) and to half the resolution limit (right panel). The hatch pattern highlights the region of superimposition between the two peaks, i.e. the point where the contribution to the displayed intensity (black trace) comes from both signals. It is apparent that the extension of such region is the bigger as the closer are the two peaks.

polynomion to approximate the baseline. In addition, since the problem is over-determined (usually, a 50-parameters fit is carried out on thousands of experimental points), the fit captures the overall trend of the data, thus making the deconvolution approach less affected by the low sensitivity and resolution.

5.1.2 The optimization process

The fit of a spectrum in the least-squares sense is performed by minimizing the target function φ shown in equation 5.1, which is the squared sum of the difference between the experimental spectrum and the simulated model. More formally, to deconvolute an experimental spectrum $\mathbb{S}^{\mathrm{exp}}$ with M Voigt signals, we have to minimize

$$\varphi = \left\| \mathbb{S}^{\mathrm{exp}} - \mathcal{FT}\left\{ \sum_{k=0}^{M-1} \mathbb{s}^{\mathrm{Voigt}}(t|\mathbb{K}[k], \omega[k], \mathbb{\Gamma}[k], \boldsymbol{\beta}[k]) \right\} \right\|_2^2 \tag{5.1}$$

where $\mathbb{K}[k]$, $\omega[k]$, $\mathbb{\Gamma}[k]$, $\boldsymbol{\beta}[k]$ are respectively the intensity, relative frequency, FWHM and Gaussian fraction of the k-th Voigtian signal, as shown in equation 2.2.

For compactness of illustration, from now on we will refer to the experimental data array as $\mathbb{y}$, to the array of parameters to be minimized as $\mathbb{x}$, and to the model function as $f(\mathbb{x})$, as shown in equation 5.2.

$$f(\mathbb{x}) = \mathcal{FT}\left\{ \sum_{k=0}^{M-1} \mathbb{s}^{\mathrm{Voigt}}(t|\mathbb{x}) \right\},\ \mathbb{x} = \left(\mathbb{K}^T|\omega^T|\mathbb{\Gamma}^T|\boldsymbol{\beta}^T\right)^T \tag{5.2}$$

The target function in equation 5.1 thus draws down to

$$\varphi = \|\mathbb{y} - f(\mathbb{x})\|_2^2 \tag{5.3}$$

To account for the absolute intensity difference between the experimental data and the computed model, the cost function can be modified to include a multiplicative term I. This

term is not an actual parameter of the fit, as it can be easily expressed in closed form and calculated at each iteration (see appendix A4).

$$\varphi = \|\mathbb{y} - I\, f(\mathbb{x})\|_2^2, \quad I = \frac{\langle \mathbb{y} \odot f(\mathbb{x}) \rangle}{\langle f^2(\mathbb{x}) \rangle} \tag{5.4}$$

The optimized set of parameters $\hat{\mathbb{x}}$ is the one that minimizes the cost function in the least-squares sense:

$$\hat{\mathbb{x}} = \min_{\mathbb{x}} \{\varphi(\mathbb{x})\} = \min_{\mathbb{x}} \left\{ \|\mathbb{y} - I\, f(\mathbb{x})\|_2^2 \right\} \tag{5.5}$$

It is clear from equation 2.2 that the model function $f(\mathbb{x})$ is not linear with respect to the parameters $\mathbb{x}$, therefore a closed-form solution for the problem does not exist. This kind of optimization problem in fact falls under the category of *non-linear least-squares*, which is typically solved by computing an initial guess for the parameters $\mathbb{x}_0$, which is then iteratively refined until a given arrest criterion is satisfied.

Let us consider an optimization problem for an array $\mathbb{x}$ of p parameters. The $(p+1)$-dimensional hypersurface defined by the graph of $\varphi(\mathbb{x})$ is called the *error surface* of this optimization problem. For instance, the error surface for the optimization of one parameter is a 2D graph, for two parameters is a 3D surface, etc. In this representation, the optimized parameters $\hat{\mathbb{x}}$ represent the global minimum of the error surface. The algorithm employed for navigating the error surface, drawing the path from the initial guess to the minimum, is responsible for the computation of the step length and direction. Among the countless existing iterative optimization algorithms, a few worth mentioning are Levengberg-Marquardt[48], the Nelder-Mead simplex method[49], Dual Annealing[50], and sequential Monte Carlo[51].

5.2 Deconvolutions in `KLASSEZ`

The initial guess for the deconvolution is conveniently generated manually, in an interactive manner. To do so, `KLASSEZ` features a graphical user interface (GUI) that allows to place as many spectral component as wanted, and to edit their appearance trying to match the experimental spectrum. An example of this GUI is shown in figure 5.2. It is also possible to work on restricted portions of the spectrum at once: the already processed regions are marked with a green background. The guess is then saved in a text file, called "*.ivf*" file, structured as in listing 5.1. The format of the *.ivf* file is codified in order to be read by a specific function, so to make the upload in a python script trivial.

This kind of GUI is very convenient for the deconvolution of solid-state NMR experiments, where there are few, very broad peaks. Although in principle the same interface could be used for solution NMR spectra, the presence of tens (or hundreds) of very sharp peaks can make the process very tedious. To solve this issue, another, more automated version of the GUI was implemented, shown in figure 5.3. Here, the position of the peaks is determined using a pick-picker, which detects the maxima of the spectrum above a certain threshold and with a given prominence (i.e. the vertical distance between the top of the peak and the closest valley). The user can modify the threshold and prominence values interactively, in order to indirectly control the detected peaks. To allow a further degree of freedom in the computation of the guess, the program offers the possibility to manually add or remove peaks. Once the peak positions have been determined, the linewidth and intensity of each component are estimated. The model is finally drawn by substituting these values in equation 2.2 and setting $\beta = 0$. The possibility to work on restricted regions of the spectrum at the time and the format of the file that is saved at the end (listing 5.2) is retained from the previous GUI. Although not being

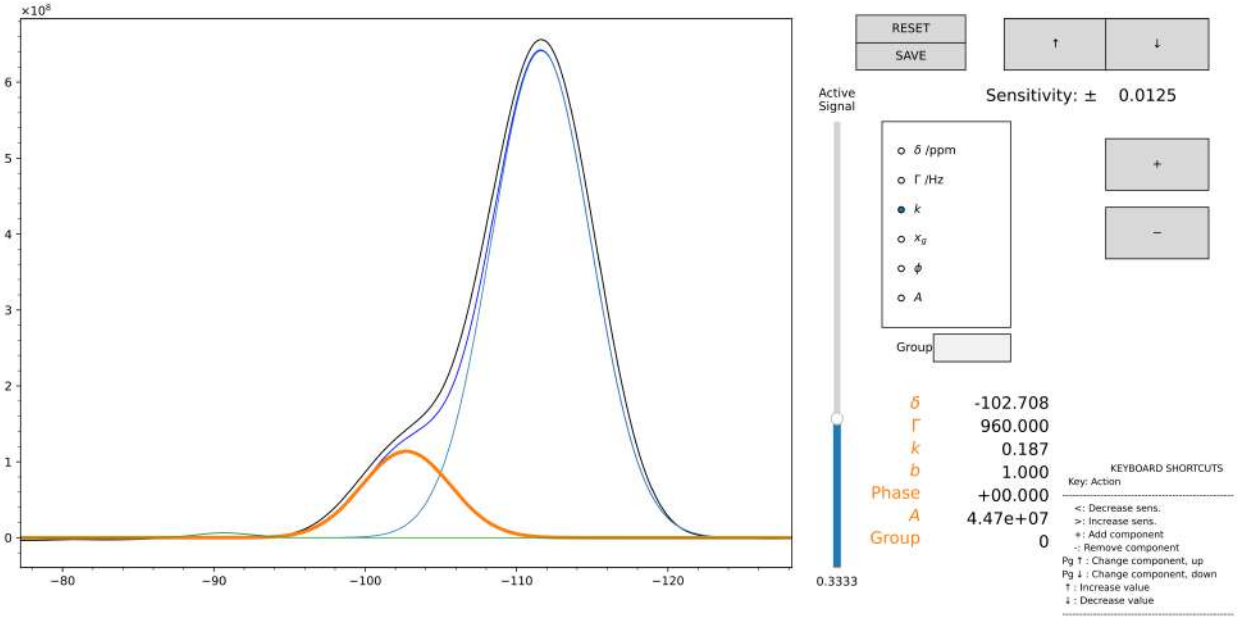


Figure 5.2: GUI for the manual computation of the initial guess for the deconvolution of a ^{29}Si spectrum of a lysozyme-silica composite[42]. The user can add or remove peak components by clicking on the "+" and "−" buttons, respectively. The active signal has a broader contour and is marked with the same color of the text guidelines. The user can change the active signal moving the vertical slider. The parameters that describe the active peak can be adjusted with the mouse scroll: the peak and the total trace (in blue) are thus updated in real time. When the "**SAVE**" button is clicked, the active region is marked with a green span, and the initial guess of that region is written as a section of the *.ivf* file.

as accurate as the manual peak-by-peak computation, this GUI is much faster and intuitive to use, and at the end of the optimization step can still give a satisfactory fit.

No matter how it is computed, the initial guess must be refined in order to obtain the set of parameters that best fit the spectrum. For this purpose, the target function in equation 5.4 is minimized using the Levenberg-Marquardt algorithm. The parameters of the peaks are allowed to move during the fit within a certain interval around their starting value, which is specified by the user at the initialization of the fit. Some parameters can also be blocked to improve the results. At the end of the optimization, a "*.fvf*" file is saved, with the same format of the *.ivf* file employed for the initial guess. The fit results of the two examples presented in this sections are shown in figure 5.4 and 5.5, and the correspondent *.fvf* files are given in listings 5.3 and 5.4.

The whole routine is implemented in `KLASSEZ` in the `fit.Voigt_Fit` class, which behaves as an interface that handles the input generation, the fit itself, the save of the output files, and the generation of figures. An example of a working code, taken from `KLASSEZ` test data, is shown in appendix A6.

This approach was successfully applied on solid-state NMR spectra of several graphene oxide samples, that were employed under different conditions as catalysts for the Povarov reaction[52] (publication **??**) and for the production of activated dienes trough the ring-opening of cyclobutanol moieties[53] (publication **??**). The ^{13}C solid-state NMR investigation showed that the material surface features a different ratio of aromatic, hydroxyl and epoxide groups. Upon repetitive cycles of catalytic use for the Povarov reaction, the graphene oxide presents a significative reduction and a partial deactivation. This is confirmed by the arise of aliphatic signals in the $\{^1\text{H}\}-^{13}\text{C}$ HETCOR experiment, which are not visible in the pristine catalyst. A similar observation was made on the graphene oxide samples used for the ring-opening

Listing 5.1: Input file generated with the manual GUI of figure 5.2. The header line, marked with a "!", contains information on who performed the fit and when. This also serves to separate different versions of the guess: in fact, the *.ivf* and *.fvf* always append the new data, in order to not erase previous version by mistake. Then, under the "Region" keyword there are the limits, in ppm, of the region where the guess was computed, and the total intensity. Finally, a table of the values to be used to generate the signals follows.

```
! Initial guess computed by francesco on 02/11/2024 at 16:35:05

         Region;     Intensity
-------------------------------------------------------------------------------------------------
-77.212:-128.204; 6.20012349e+07

   #;             u;           fwhm;    Rel. I.;     Phase;      Beta;     Group
-------------------------------------------------------------------------------------------------
   1;   -111.61818506; 1085.000000;  0.860360;     0.000;   1.00000;         0
   2;   -102.70807031;  960.000000;  0.135135;     0.000;   1.00000;         0
   3;   - 90.62022232;  565.000000;  0.004505;     0.000;   1.00000;         0
-------------------------------------------------------------------------------------------------

=================================================================================================
```

```
! Initial guess computed by francesco on 02/11/2024 at 17:02:34

         Region;     Intensity
-------------------------------------------------------------------------------------------------
     3.472:2.782;  8.41321287e+04

   #;             u;           fwhm;    Rel. I.;     Phase;      Beta;     Group
-------------------------------------------------------------------------------------------------
   1;     3.27228290;     1.137412;  0.065521;     0.000;   0.00000;         0
   2;     3.20786873;     1.000000;  0.036825;     0.000;   0.00000;         0
   3;     2.83578683;     1.557376;  0.057764;     0.000;   0.00000;         0
   4;     3.30709056;     1.367524;  0.060670;     0.000;   0.00000;         0
   5;     2.85899193;     1.483217;  0.056549;     0.000;   0.00000;         0
   6;     2.87099458;     1.616869;  0.067663;     0.000;   0.00000;         0
   7;     3.37110466;     1.941959;  0.054702;     0.000;   0.00000;         0
   8;     3.27908440;     1.419349;  0.070821;     0.000;   0.00000;         0
   9;     3.34269840;     1.833882;  0.075629;     0.000;   0.00000;         0
  10;     3.38150694;     1.812106;  0.051207;     0.000;   0.00000;         0
  11;     2.89379960;     1.597707;  0.069254;     0.000;   0.00000;         0
  12;     3.35350078;     2.095580;  0.081949;     0.000;   0.00000;         0
  13;     3.25027806;     1.848501;  0.077094;     0.000;   0.00000;         0
  14;     3.31389206;     1.122935;  0.056068;     0.000;   0.00000;         0
  15;     3.23587489;     1.919994;  0.080171;     0.000;   0.00000;         0
  16;     3.22227190;     1.000000;  0.038111;     0.000;   0.00000;         0
-------------------------------------------------------------------------------------------------

=================================================================================================
```

Listing 5.2: Input file generated with the automatic GUI of figure 5.3. The header line, marked with a "!", contains information on who performed the fit and when. This also serves to separate different versions of the guess: in fact, the *.ivf* and *.fvf* always append the new data, in order to not erase previous version by mistake. Then, under the "Region" keyword there are the limits, in ppm, of the region where the guess was computed, and the total intensity. Finally, a table of the values to be used to generate the signals follows.

Listing 5.3: Output file generated by the fit of figure 5.4. The header line, marked with a "!", contains information on who performed the fit and when. This also serves to separate different versions of the guess: in fact, the *.ivf* and *.fvf* always append the new data, in order to not erase previous version by mistake. Then, under the "Region" keyword there are the limits, in ppm, of the region where the guess was computed, and the total intensity. Finally, a table of the values to be used to generate the signals follows.

```
! Fit performed by francesco on 02/11/2024 at 16:43:39

         Region;     Intensity
-------------------------------------------------------------------------------------------------
-77.212:-128.204; 6.69065853e+07

   #;            u;          fwhm;   Rel. I.;     Phase;      Beta;     Group
-------------------------------------------------------------------------------------------------
   1;   -111.66304403; 1128.714138; 0.853892;     0.000;   1.00000;         0
   2;   -102.79302839; 1061.512045; 0.145809;     0.000;   1.00000;         0
   3;   -89.38605150;   365.000000; 0.000298;     0.000;   1.00000;         0
-------------------------------------------------------------------------------------------------

=================================================================================================
```

Listing 5.4: Output file generated by the fit of figure 5.5. The header line, marked with a "!", contains information on who performed the fit and when. This also serves to separate different versions of the guess: in fact, the *.ivf* and *.fvf* always append the new data, in order to not erase previous version by mistake. Then, under the "Region" keyword there are the limits, in ppm, of the region where the guess was computed, and the total intensity. Finally, a table of the values to be used to generate the signals follows.

```
! Fit performed by francesco on 02/11/2024 at 17:04:43

         Region;     Intensity
-------------------------------------------------------------------------------------------------
     3.472:2.782; 1.10121619e+05

   #;            u;          fwhm;   Rel. I.;     Phase;      Beta;     Group
-------------------------------------------------------------------------------------------------
   1;     3.27231736;     1.363647; 0.068956;     0.000;   0.15766;         0
   2;     3.20770664;     1.877137; 0.049832;     0.000;   0.31687;         0
   3;     2.83601523;     1.554151; 0.056769;     0.000;   0.30679;         0
   4;     3.30715577;     1.323781; 0.058790;     0.000;   0.16835;         0
   5;     2.85909688;     1.583577; 0.058694;     0.000;   0.25667;         0
   6;     2.87088008;     1.596544; 0.065985;     0.000;   0.30753;         0
   7;     3.37109130;     1.961057; 0.050079;     0.000;   0.34846;         0
   8;     3.27921788;     1.344226; 0.066084;     0.000;   0.26023;         0
   9;     3.34287817;     2.101984; 0.076398;     0.000;   0.30733;         0
  10;     3.38174544;     2.029247; 0.049399;     0.000;   0.38238;         0
  11;     2.89392837;     1.577143; 0.068741;     0.000;   0.29308;         0
  12;     3.35347762;     2.032095; 0.074968;     0.000;   0.40361;         0
  13;     3.25031465;     1.952447; 0.074867;     0.000;   0.37176;         0
  14;     3.31403504;     1.333613; 0.055634;     0.000;   0.27343;         0
  15;     3.23592492;     1.899370; 0.075398;     0.000;   0.31889;         0
  16;     3.22204868;     1.847788; 0.049406;     0.000;   0.35585;         0
-------------------------------------------------------------------------------------------------

=================================================================================================
```

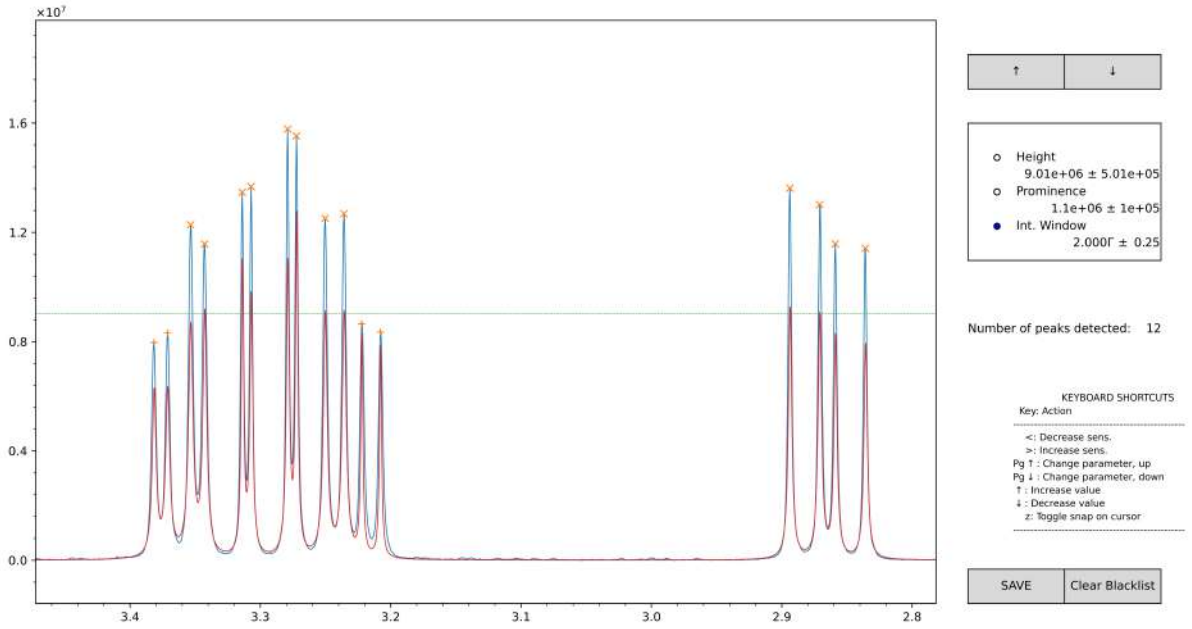


Figure 5.3: GUI for the automatic computation of the initial guess for the deconvolution of a ^{1}H spectrum of ochratoxin-a. The user can use the mouse scroll to move the threshold or prominence employed by the peak-picker for the detection of the signals. The detected positions are marked with orange ×, and the model (red trace) is automatically computed by estimating the linewidths and the integrals. In case the linewidth estimation would fail for some reason, a default FWHM of 5 Hz is set. The user can also manually add peak positions that are not detected automatically, which in this case would appear as orange +, or remove a peak that is automatically detected, by double-clicking with the left or right button of the mouse respectively. When the "**SAVE**" button is clicked, the active region is marked with a green span, and the initial guess of that region is written as a section of the *.ivf* file.

reaction. However, additional insights were gained by noticing that the epoxy-hydroxyl ratio did not change upon repetitive uses of the catalyst, indicating that the opening of the epoxy ring is not an intermediate step of the reaction. Furthermore, the intensity of the aromatic signals increased when the aliphatic signals showed up: the aromatic peaks at 130 ppm in the $\{^{1}\mathrm{H}\}-^{13}\mathrm{C}$ HETCOR spectra indeed showed a magnetization source coming from aliphatic protons (1-2 ppm), when a longer cross-polarization contact time is employed in the sequence. This information can be used to infer that the partial deactivation of the active sites is due to the absorption of small aliphatic molecules on the surface of the material.

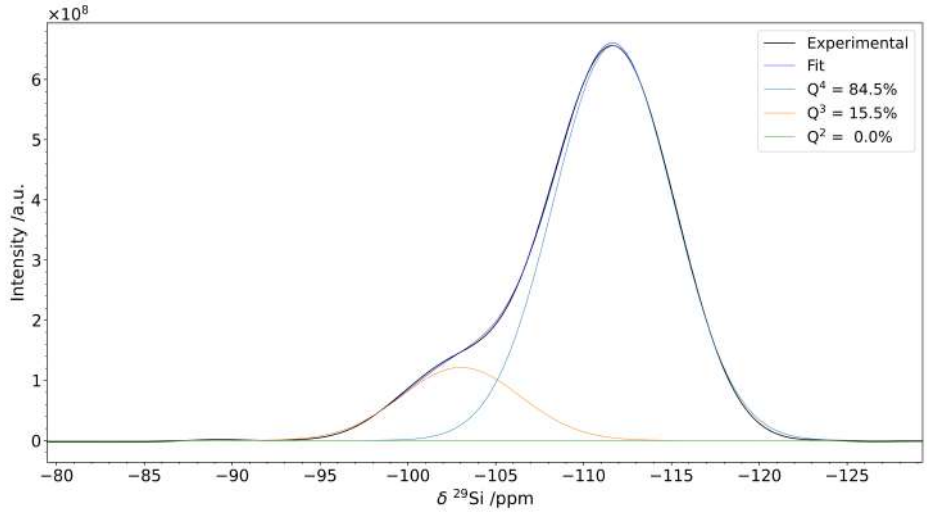


Figure 5.4: Result of the fit obtained using the file in listing 5.1 as initial guess.

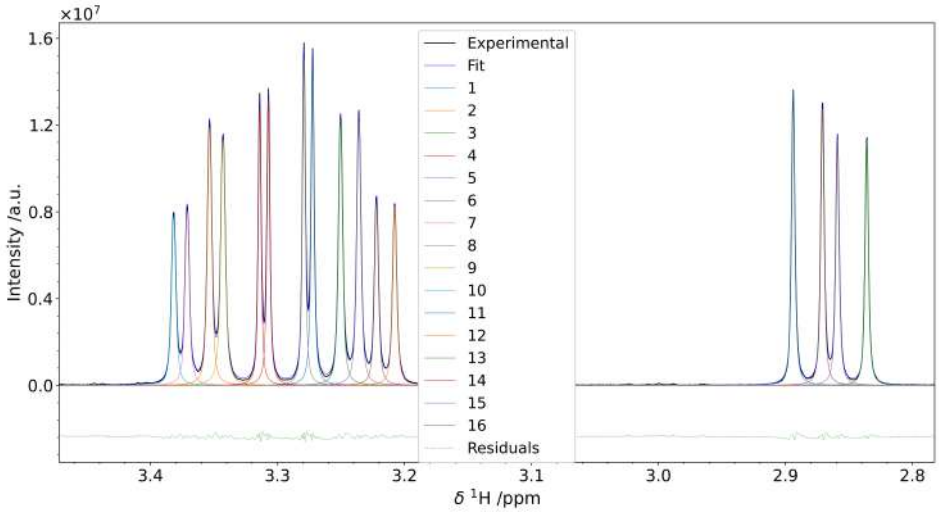


Figure 5.5: Result of the fit obtained using the file in listing 5.2 as initial guess.

5.3 The 2D-deconvolution problem

Quantitative analysis on 2D spectra, when possible, are normally performed through integration of the peaks, which consists in computing a cumulative sum of the spectral intensity over a restricted dominion around the base of the peaks. Intuitively this approach is quite error-prone, as it is greatly affected by the shape of the peak and often leads to an underestimation of the true peak volume because of the very long tails, which are eventually not taken into account due to the restriction on the integration domain. Furthermore, the definition of the shape of the integration region in 2D has great impact on the integral estimation. From these considerations, it appears natural to extend the method proposed in the previous section to bidimensional spectra, using as model the 2D Voigt described in section 2.2.

The computation of the initial guess requires two steps. First, the user must look at the overall 2D spectrum, and place a marker on the approximate positions of the peaks they want to fit. Then, for all the selected positions, the plots of the experimental spectrum and the model are displayed, superimposed, projected on the traces of the direct and indirect dimension. The user can adjust the peak parameters for fine adjustment of the model to match the experimental spectrum better. The process can be halted and resumed at any stage.

The actual fitting routine consists into minimizing the difference between the experimental spectrum and the model in the least-squares sense. The problem here is that the fitting procedure is computationally very expensive as the calculated spectrum results as the sum of many peaks: if we consider that each peak can be described by six parameters (two frequencies, two linewidths, fraction of gaussianity and relative intensity), it is easy to understand how quickly the total number of parameters of the fit escalate. Furthermore, the main components of a 2D NMR spectrum are normally the empty spaces. This means that the regions without peaks represent the major contribution in the calculation of the residuals, and throughout the fitting procedure they will remain quite steady: as a consequence, the fitting algorithm is likely to stop in a local minimum of the error surface. As an example of this situation, the result of the fit of a ^{1}H-^{15}N HSQC spectrum of a zinc-binding 32 kDa protein[54], acquired at 700 MHz ^{1}H Larmor frequency, is shown in figure 5.6. The optimized model (green trace) is apparently smeared, which means that the fitting program tried to accommodate for the very long tails of the 2D Lorentzian peaks in more crowded regions of the spectrum by increasing the linewidths of the model. The outcome is an unrealistic set of peaks parameters, which do not match the experimental spectrum at all.

A possible solution to this problem is to divide the fitting procedure into regions and defining boundaries around clusters of peaks, so to decrease the contribution of both long signals' tails and baseplane-only region, in order to get a more accurate deconvolution. The initial guess is computed on the contour plot of the spectrum, starting with a peak-picking step. The collection of the chemical shifts are then grouped into clusters of neighbors. Two peaks are considered to belong at the same cluster if their difference in chemical shifts is less than three times the estimated linewidths. At this point, the optimization regions are selected to be the smallest rectangles that are able to contain all the peaks in each cluster. The fitting is thus performed sequentially on each region.

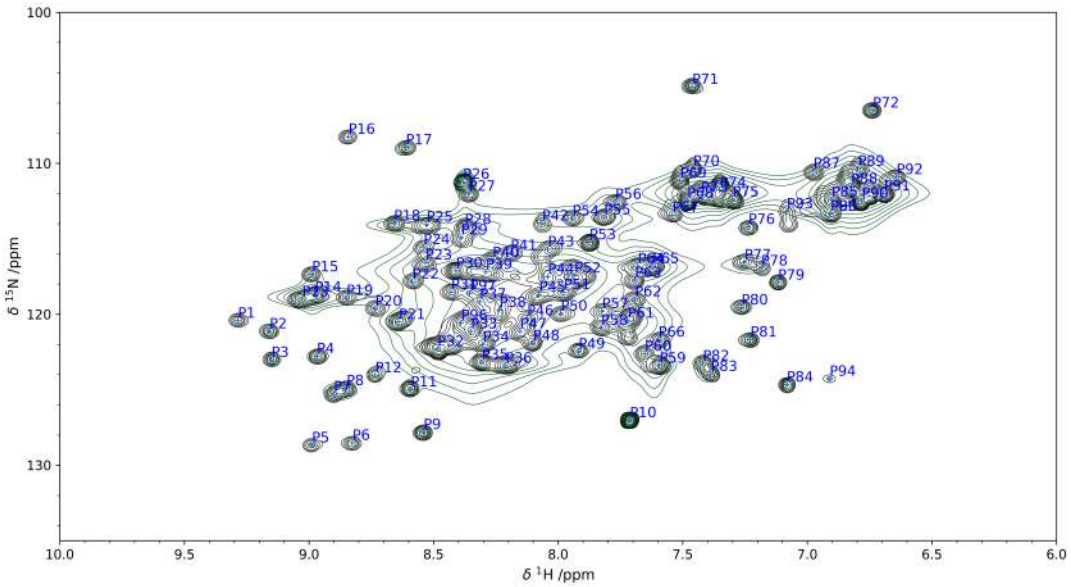


Figure 5.6: Result of 2D peak deconvolution performed on the ^{1}H-^{15}N HSQC spectrum of a zinc-binding 32 kDa protein. The position of the peaks are displayed on the figure with a blue cross marker. The modelled spectrum (green trace) greatly differs from the experimental one (black trace), proving the poor performances of the fit routine.

6. Quantitative NMR

This chapter explores the quantitative aspects of NMR spectroscopy. The available techniques to gather insights about the concentrations of a mixture will be presented, providing a comprehensive overview of their theoretical foundations, practical applications, and limitations. Furthermore, an open-source python software to deconvolve spectra of mixtures employing the Indirect Hard Modelling will be described. The outcomes of this program will be compared with the traditional peak-integration approach.

6.1 How to acquire a quantitative experiment

NMR is intrinsically a quantitative technique. As a matter of fact, the intensity of each signal solely depends on the number of nuclei that contribute to it times amount of substance. In other words, all species have the same molar extinction coefficient from the NMR point of view. This feature, combined with the outstanding qualitative power of the NMR spectra (i.e. each molecule has it own, unique spectrum), makes NMR spectroscopy an excellent tool for the quantitative analysis of chemical mixtures.

It must be noted, however, that to record an NMR experiment that is truly quantitative there are several requirements that must be fulfilled. The pulse sequence to be used for the acquisition should not contain elements that affect the signal intensity, as for instance relaxation-based filters (inversion recovery, CPMG, 1D-NOESY). Solvent-suppression elements can be used, but it must be considered that the intensity of the signals near the solvent resonance can be hampered.

It is important to set the recovery delay long enough to ensure the complete restore of the sample magnetization to equilibrium. An optimal choice would be 5-7 times the T_1 of the sharpest signal (see figure 6.1). For this reason, a preliminary inversion recovery can be useful to estimate the longitudinal relaxation times of the signals.

The acquisition time must be long enough to ensure the complete decay of the FID, in order to avoid truncation artifacts. This might appear paradoxical, as all the quantitative information is encoded in the first point of the FID. However, one has to bear in mind that the first point of the FID is never actually acquired, because of the delay between the end of the pulse and the start of the acquisition (the so-called *dead time*), which is required to avoid the ringing of the radiofrequency pulse to enter in the receiver circuitry. Therefore, the quantitative information must be recovered from the whole spectrum. As the complete decay of all the signals might take seconds, especially with small molecules, the use of decoupling sequences is discouraged, as they can cause sample overheating and destroy the probe circuitry.

To maximize sensitivity, quantitative NMR spectra are acquired on ^{1}H, as it has the highest gyromagnetic ratio among all the (common) magnetically active nuclei, and it has almost 100% of natural abundance. This also implies that ^{1}H-NMR signals have the fastest relaxation rates, due to the stronger coupling with the lattice. Even in this case, the excitation pulse should be thoroughly calibrated in order to convert the maximum amount of magnetization

into on-plane magnetization. However, the pulse duration should be short enough to give a flat excitation profile in the spectral region where the interested signals are. The common rule of $EW_{\text{flat}} = 1/(4\tau_p)$ applies (see figure 6.2).

The processing of the spectrum must be kept at minimum, as it can alter the lineshapes. It is common practice to not perform any apodization at all, or to employ a very mild exponential modulation to remove slight distortions to the lineshapes due to the non-ideality of the instrument components. After Fourier-transformation, the spectrum must be carefully phased and baseline-corrected, in order to avoid biases in the quantification (*vide infra*, section 6.2).

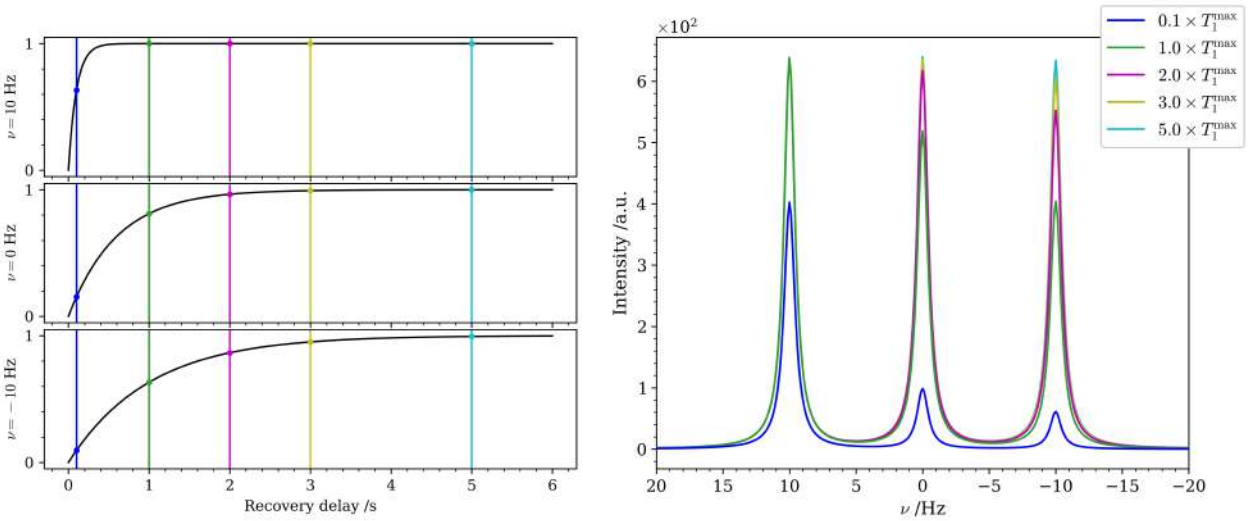


Figure 6.1: Three Lorentzian signals with same linewidth (2 Hz) were simulated. The intensity of the three signals was set to depend on their longitudinal relaxation time as in a saturation recovery experiment, i.e. the recovered intensity after the relaxation delay τ is given by: $I(\tau|T_1) = 1 - \exp\{-\tau/T_1\}$. This dependence is illustrated in the left panel of the figure, where $T_1^{10\,\mathrm{Hz}} = 0.1\,\mathrm{s}$, $T_1^{0\,\mathrm{Hz}} = 0.35\,\mathrm{s}$, $T_1^{-10\,\mathrm{Hz}} = 1\,\mathrm{s}$. The resulting spectra at different values of recovery delay are drawn in the right panel of the figure, where it can be observed that the native intensity equal to 1 for all signals is recovered only if the recovery delay is set longer than 5 times the T_1 of the slowest-relaxing signal.

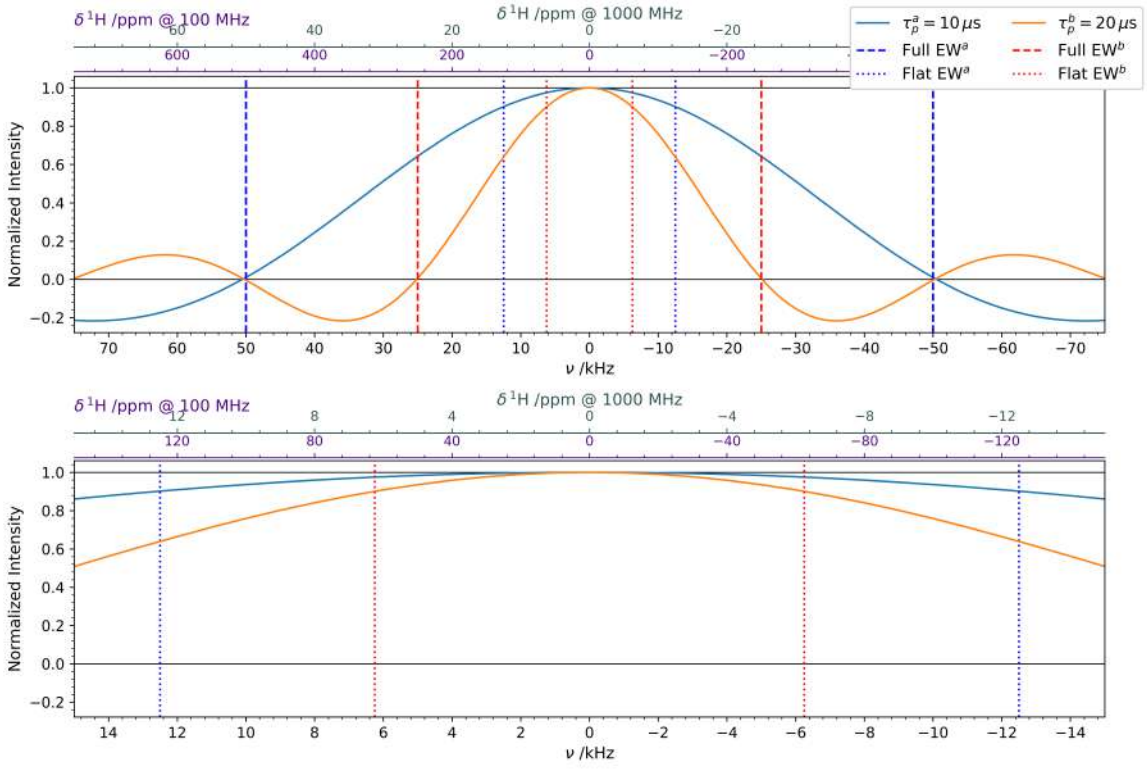


Figure 6.2: The excitation profiles of two rectangular pulses, of length $\tau_p^a = 10\,\mu$s (blue) and $\tau_p^b = 20\,\mu$s (orange), are shown in the figure as solid lines. Their full excitation window $EW = 1/\tau_p$, considered as the frequency region enclosed between the first two zeroes of the excitation profile, are marked with dotted lines, whereas the region of flat excitation $EW_f = 1/(4\tau_p)$ is delimited by dashed lines. The bottom panel shows the same situation of the upper panel in a more restricted range of frequencies, in order to better appreciate the differences between the profiles of the two pulses. The obtained values for the excitation windows are: $EW^a = 100\,\mathrm{kHz}$, $EW^b = 50\,\mathrm{kHz}$, $EW_f^a = 25\,\mathrm{kHz}$, $EW_f^b = 12.5\,\mathrm{kHz}$. For reference, the ppm scales for the ^{1}H chemical shifts at 100 and 1000 MHz corresponding to the displayed ranges of frequency are drawn on top of the panels.

6.2 Quantification through integration

Since the very dawn of the technique, the quantitative analysis of NMR spectra has been performed by integration of the characteristic signals of the molecules of interest. When computers where not available for such an analysis, the spectra were printed on graph paper - possibly through large-format printers, and the intensities were measured using a ruler, or the areas obtained by cutting and weighting each peak. An even fancier approach consisted in trimming around the signals and fit the cropped region with standard objects. Nowadays, with technological improvements in the digitalization of the FID and the advent of more powerful analysis softwares, the integration is performed through dedicated graphical user interfaces, which allows for selecting the integration regions and return the values in real time.

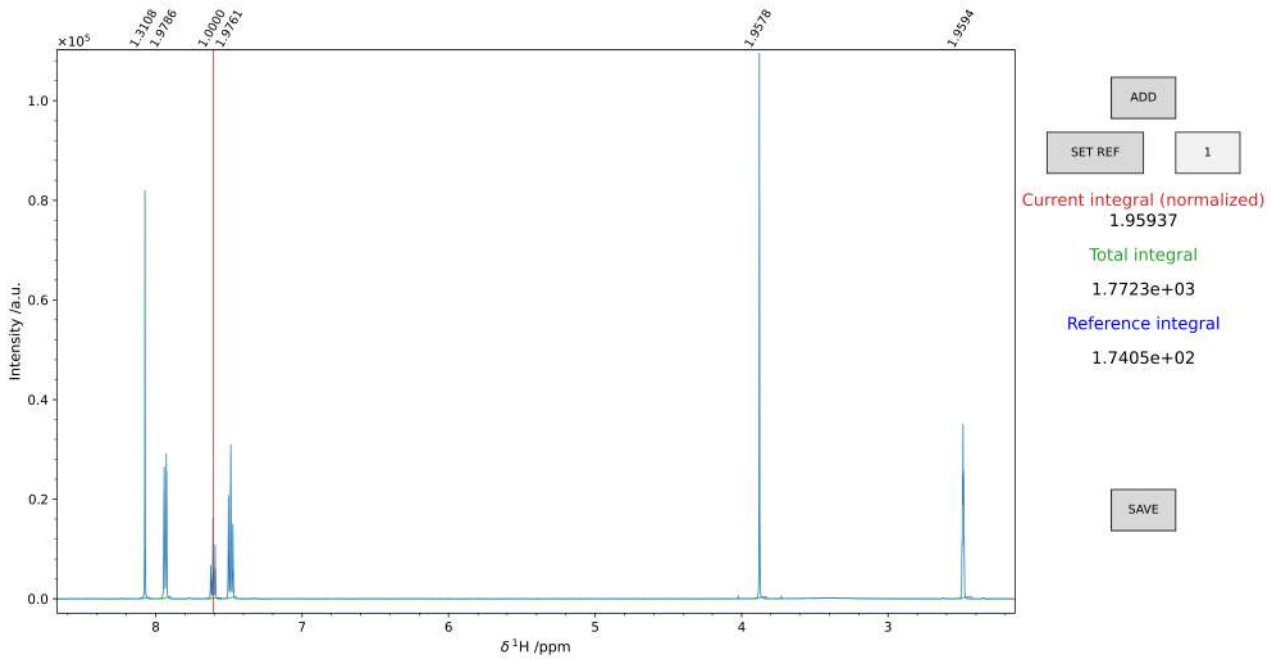


Figure 6.3: The GUI for integration implemented in `KLASSEZ` is shown here for the quantification of a known mixture of benzoic acid and dimethyl terephtalate in dimethylsulfoxide-*d*6. The user can select the integration region in a drag-and-drop fashion. By clicking the "ADD" button, the value of the integral is displayed at the top of the figure. It is also possible to draw a reference integral by selecting the desired region and clicking on the "SET REF" button. This normalizes all the computed integrals to this value. In this example, all integrals values are referred to the $p-$H of benzoic acid ($\delta = 7.609\,\text{ppm}$).

Given a spectrum $S(\omega)$, its integral for quantification in the range $[\nu_1, \nu_2]$ is:

$$I_{\nu_1,\nu_2} = \frac{1}{2\pi}\int_{\nu_1}^{\nu_2} S(\omega)\, d\nu, \quad \omega = 2\pi\nu \tag{6.1}$$

As in practice the spectrum is digitized in the array $\mathbb{S}$, the discrete sum of the spectrum array is computed over the frequency scale $\mathbb{v}$ either using the trapezoid method (equation 6.2) or by

exploiting the fundamental theorem of Calculus on the cumulative sum of $\mathbb{S}$ (equation 6.3).

$$I^{\text{trapz}} = \frac{\Delta\nu}{2} \sum_{k=k_0}^{k_1-1} (\mathbb{S}[k+1] + \mathbb{S}[k]) \tag{6.2}$$

$$I^{\text{ftc}} = \mathbb{G}[k_1] - \mathbb{G}[k_0], \quad \mathbb{G}[k] = \Delta\nu \sum_{j=0}^{k} \mathbb{S}[j] \tag{6.3}$$

$$\Delta\nu = |\nu[k+1] - \nu[k]| \quad \forall k,\, \nu[k_0] = \nu_0,\, \nu[k_1] = \nu_1$$

It is obvious that, in quantitative NMR conditions, the intensity of the signal does not depend on time nor on frequency, therefore it can be drawn out from the integral in the computation of the Fourier transform. This means that the signal intensity in the FID and in the spectrum are equal. However, this is a direct consequence of the fact that, in the continuous case, the frequency spacing $\Delta\nu \to 0$. When one comes to consider the real, discrete case, this assumption does not hold anymore, and the relationship between the two intensity becomes:

$$I^{\text{FID}} = 2\Delta\nu\, I^{\text{S}} = \frac{2}{SW} I^{\text{S}} \tag{6.4}$$

The quantitative assessment is performed by applying a very simple formula. Let us consider as signal a the signature peak of the analyte, and as signal r the signature peak of the reference compound. These two signals integrate for H_a and H_r protons, respectively. Then, the computed integrals I_a and I_r are connected by the following relationship:

$$c_a \frac{I_a}{H_a} = c_r \frac{I_r}{H_r} \tag{6.5}$$

where c_a and c_r are the concentrations of the analyte and the reference in the mixture. The unknown concentration of the analyte, relatively to the reference, is thus calculated by simple algebraic manipulations of equation 6.5:

$$\frac{c_a}{c_r} = \frac{I_r}{I_a} \frac{H_a}{H_r} \tag{6.6}$$

This is an extremely simple and intuitive approach, and guarantees very accurate results. However, it does so under some conditions, as discussed hereon. As phase correction aims to confine the absorption component of the NMR signal in the real part of the spectrum, incorrect phasing introduces a certain fraction of dispersion component in the lineshape of the peaks, which is subtracted from the actual intensity. Baseline distortions have the same effects, but can also include offsets errors, that further impact on the outcome of the analysis.

The most important bottleneck of the integration procedure, however, is the resolution of the spectrum. To account for the 99% of the intensity of a Lorentzian signal, the integration window must be extended to 64 times the FWHM of the peak[19], as shown in figure 6.4. Furthermore, the integration region should include all contributions arising from couplings (e.g. the ^{13}C satellites), as they are part of the total signal intensity. In practical applications, it is rather uncommon to find a signal for each component of the mixture that fulfills these requirements.

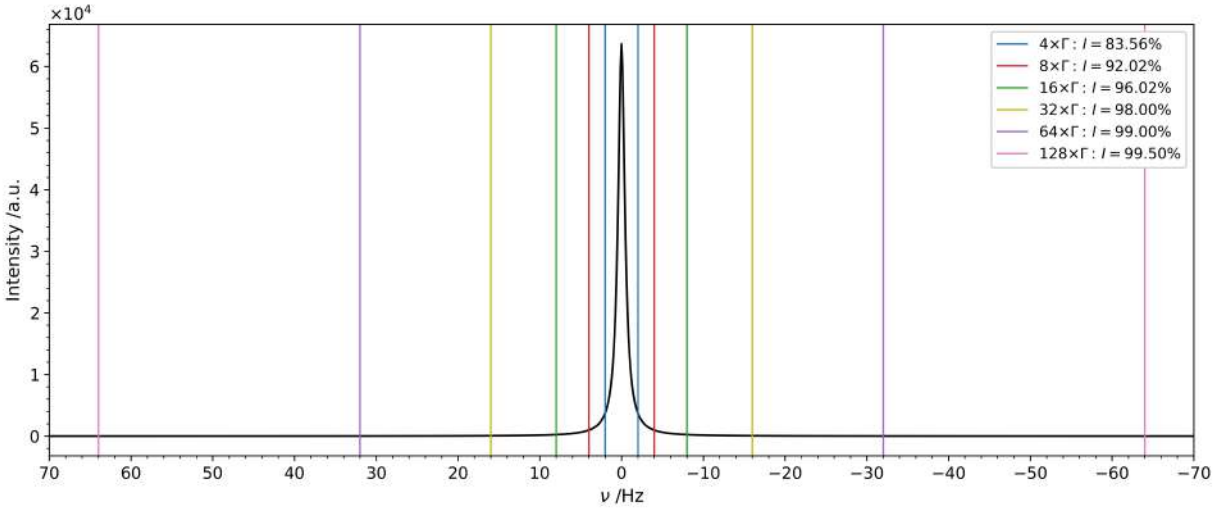


Figure 6.4: A Lorentzian signal centered at $\nu = 0\,\text{Hz}$ was simulated with full-width at half-maximum $\Gamma = 1\,\text{Hz}$ and intensity $K = 100$. The signal was integrated over a set of frequency ranges equal to 4Γ (blue), 8Γ (red), 16Γ (green), 32Γ (yellow), 64Γ (purple) and 128Γ (pink). The computed intensities are reported in the legend of the figure. It is apparent that the native intensity $K = 100$ is not retrieved even when it appears that the signal is fully decayed.

6.3 Indirect Hard Modelling

An alternative solution when the integration of the peaks is not possible is to deconvolve the spectrum and extract the intensities from the fit. As described in chapter 5, this approach is not limited by the resolution of the data, and is less sensitive to phase and baseline distortions, as these can be included in the model. However, if one considers that a single NMR signal is described by (at least) four parameters (i.e. intensity, chemical shift, linewidth, and Lorentzian/Gaussian ratio), it is easy to imagine how fast the difficulty of the deconvolution escalates with increasing number of components, if one tries to fit the spectrum as a whole. To give a couple of examples, considering p parameters to optimize, the Levenberg-Marquardt algorithm[48] (section 5.1.2) scales as $\mathcal{O}(p^3)$, while the Nelder-Mead simplex is thought to have a $\mathcal{O}(e^p)$ dependence, although it has not a well-defined behavior.

On the other hand, the task becomes rather simple if one considers the spectrum of the mixture S^{mix} as a linear combination of the spectra of its m components:

$$S^{\text{mix}} = \sum_{k=0}^{m-1} c_k \, S_k \tag{6.7}$$

where the coefficients c_k account for the concentration of the k-th species in the mixture. This procedure goes under the name of Indirect Hard Modelling (IHM).

The beauty of IHM lies in not addressing directly the highly complex mixture spectra but, instead, in building the individual hard models for each pure component based on separate, clean spectra. These component models then act as building blocks to predict the final mixture spectrum based on the concentration of each species. In addition, with minimal modifications, IHM can account for intermolecular interactions that alter peak positions, a common issue in concentrated mixtures, or lineshapes (across different instruments or in series of samples), as it is sufficient to allow for small adjustments of the peak features during the fit of the spectrum[55]. This structure favors the performance of both Levenberg-Marquardt and Nelder-Mead optimizations.

The IHM workflow consists of four main steps:

1. generate the basis set consisting of the spectra of each component of the mixture;

2. create an initial guess of the concentrations;

3. align the chemical shifts of the model with the spectrum of the mixture;

4. perform the global fit to get the actual concentrations.

In spite of the fact that the theoretical foundations for IHM were laid about 20 years ago, a lack of comprehensive software implementations has hindered its widespread adoption. For this reason, we developed `pyIHM` , an open-source python software that implements the IHM theory by offering an interface that guides the user through the various step of the algorithm, and takes care of the presentation of the results. For the reading and processing of the spectra, and for the generation of the basis set, the `pyIHM` implementation relies on `KLASSEZ` routines.

`pyIHM` reads a custom input file, a text file that instructs the program about where to find the input files and customize various options. This information goes under a series of keys, separated one from each other by a blank line. The list of keys that can be provided are:

- `BASE_FILENAME`
  Root of the filename for all the files and figures that will be saved during the whole run of the program;

- `MIX_PATH`
  Directory of the FID of the mixture spectrum to read and process;

- `MIX_SPECTRUM_TXT`
  Directory of a text file that contains the processed spectrum, in case the processing was performed with an external program;

- `COMP_PATH`
  Path to the *.fvf* files that contain the information of the pure components of the mixtures, together with the number of protons they integrate for;

- `PROC_OPTS`
  Processing options for the mixture spectrum:

  - `wf=<method>, **kws`
    Apodization. Allowed:

    * `wf=no` → no apodization (default)
    * `wf=em, lb=<lb>` → exponential modulation with `lb` Hz
    * `wf=sin, ssb=<ssb>` → sine apodization, `ssb` = 1,2,3,4,...
    * `wf=qsin, ssb=<ssb>` → squared sine apodization, `ssb` = 1,2,3,4,...
    * `wf=gmb, lb=<lb>, gb=<gb>` → Bruker-style Lorentzian to Gaussian apodization
    * `wf=gm, lb_gm=<lb>, gb_bm=<gb>, gc=<gc>` → General purpose Lorentzian to Gaussian apodization

  - `zf=<size>`
    Apply zero-filling up to size

  - `blp, pred=<n points to predict>, order=<n LP coefficients`
    Apply backward linear prediction

  - `pknl`
    Add this key to remove the group delay (mandatory for raw Bruker data!)
  - `adjph`
    Add this key to correct the phase of the spectrum

- `FIT_KWS`
  Algorithms to be used for the fit, with customized settings (only used with the flag `"-opt_method=custom"`;

- `FIT_BDS`
  Allowed displacement for the parameters during the optimization (`utol` for chemical shifts, `utol_sg` for the chemical shifts of signals within the same multiplet, `stol` for the FWHMs, `ktol` for the intensities);

- `PLT_OPTS`
  Format (`ext`) and resolution (`dpi`) of the figures that will be saved;

- `FIT_LIMITS`
  List of the regions of the spectrum to use for the fit;

- `I0`
  Initial guess for the relative concentrations.

The meaning of this information will be deepened in the following sections. Examples of such initial files are given in sections A7.2-A7.5.

6.3.1 Generation of the basis set

The k-th pure component spectrum comes from the Fourier transform of N_k Voigt signals (equation 2.2):

$$\mathbb{S}_k = \mathcal{FT}\left\{\sum_{j=0}^{N_k-1} S^{\text{Voigt}}(t|\mathbb{K}_k[j], \omega_k[j], \mathbb{\Gamma}_k[j], \mathbb{\beta}_k[j])\right\} \qquad \text{for } k = 0, \ldots, m-1 \quad (6.8)$$

To generate the collection of peaks that serves as basis set for the fit, `pyIHM` reads a set of *.fvf* files that come from the deconvolution procedures described in chapter 5. For creating the basis set, however, there might be cases where nor the individual spectra, nor other experimental data, are available for deconvolution. In these situations, it is possible to generate a *.fvf* file from knowledge gathered from previous analyses, or from literature, or from calculations[56,57,58], relying on the class `"Spectrum_1D"` in `KLASSEZ` . A simulation input file must be written (an example of which is shown in listing 2.2), which is interpreted following the procedure illustrated in section 2.1. Since the linewidths and the fraction of gaussianity are often not reported in spectral assignments, a good blind choice is to set Γ = 2-5 Hz and $\beta = 0$ (i.e. pure Lorentzian). This choice can be justified by the fact that, usually, solution NMR peaks deviates only slightly from the pure-Lorentzian lineshape, and they can be easily corrected with a mild exponential modulation, at the price of a resolution decrease (see section 3.2.2). In any case, the Lorentzian-Gaussian ratio is a fit parameter, therefore the lineshape can be adjusted during the optimization procedure. The conversion to *.fvf* file is automatically performed by the module `"to_vf"` of the class.

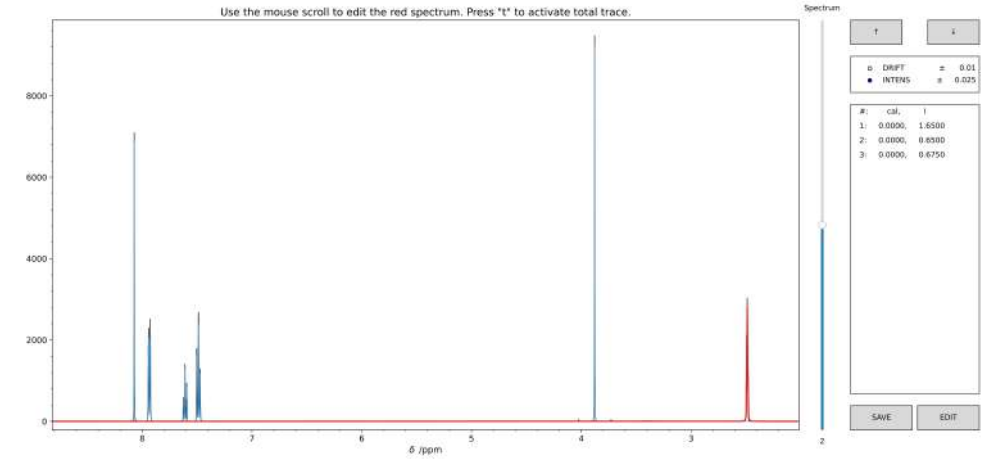
Once the *.fvf* files are generated, `pyIHM` must be instructed about where to find them. This information goes under the `COMP_PATH` key of the input file, in the format:

`<path to the file .fvf>, `n`H`

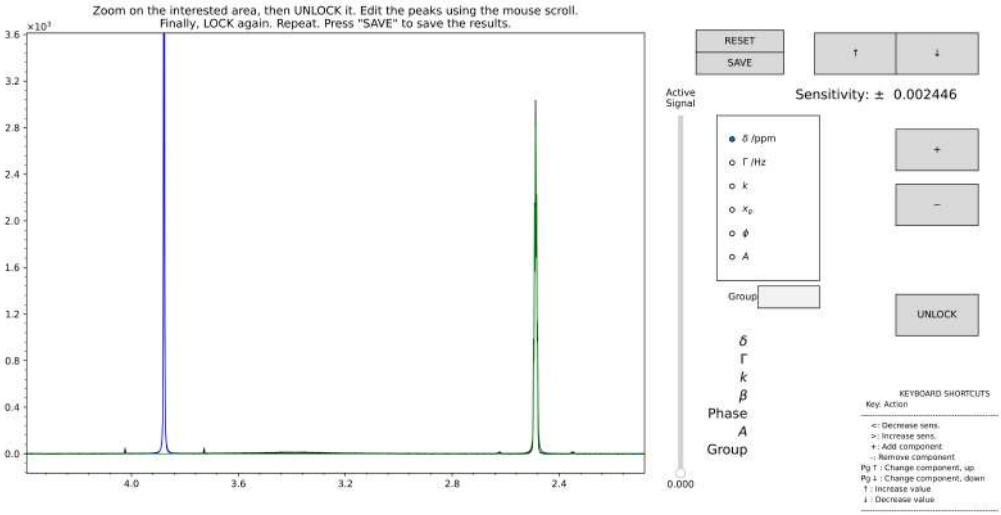
where n is the number of protons that component accounts for. `pyIHM` reads the files, and stores the reconstructed spectra as arrays $\mathbb{S}_k$ and their respective number of protons in an array $\mathbb{H}$, for $k = 0, \ldots, m-1$. The total intensities are then reworked: the k-th pure component will have its intensity set to $\mathbb{H}[k]$, whereas the mixture spectrum $\mathbb{S}^{\text{exp}}$ will be $\sum_{k=0}^{m-1} \mathbb{H}[k]$. This conversion makes the assumption that the integral of the spectrum can be totally explained by the components listed in `pyIHM` input file. Although this assumption is never true (as in this case fitting the spectrum would not be needed at all), its deviation from the truth is small enough to be compensated by the fit.

At this point, the input guess can be refined though a dedicated GUI. An example is shown in figure 6.5a. This interface displays the mixture spectrum, and the spectra of the components, superimposed. The user can shift the spectrum of each component left or right, in order to compensate for large field drifts, and to set the initial concentrations. 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 6.5b).

Finally, the region where the fit will focus on must be selected. This selection is useful to decrease the computational cost of the analysis, because of the minor number of points to be computed at each iteration, as well as to exclude poorly-modelled or distorted portions of the spectrum, such as the resonances close to the signal of the solvent. The list of borders for each window can 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 6.6). After the selection, the array $\mathbb{H}$ that contains the intensity of each component is corrected to account only the peaks within the selected region.



(a)



(b)

Figure 6.5: 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 user can change the active spectrum by moving the vertical slider. When the "EDIT" button is clicked, this GUI closes, and the one in panel **b** opens. In this interface, the user can edit the initial guess of the active spectrum peak-by-peak, in a similar fashion as for the manual computation of *.ivf* files (figure 5.2). To decrease the computational workload that sits behind this interface, the active spectrum appears as a green trace, whereas the other components are shown in blue. The experimental mixture spectrum is drawn in black. The user has to focus on a particular window, and then press "UNLOCK": the visible region of the green spectrum will then unpack in editable components. Clicking "SAVE" on this GUI closes the interface, and the GUI in panel **a** opens up again. The process can then continue, until the "SAVE" button in the first GUI is pressed.

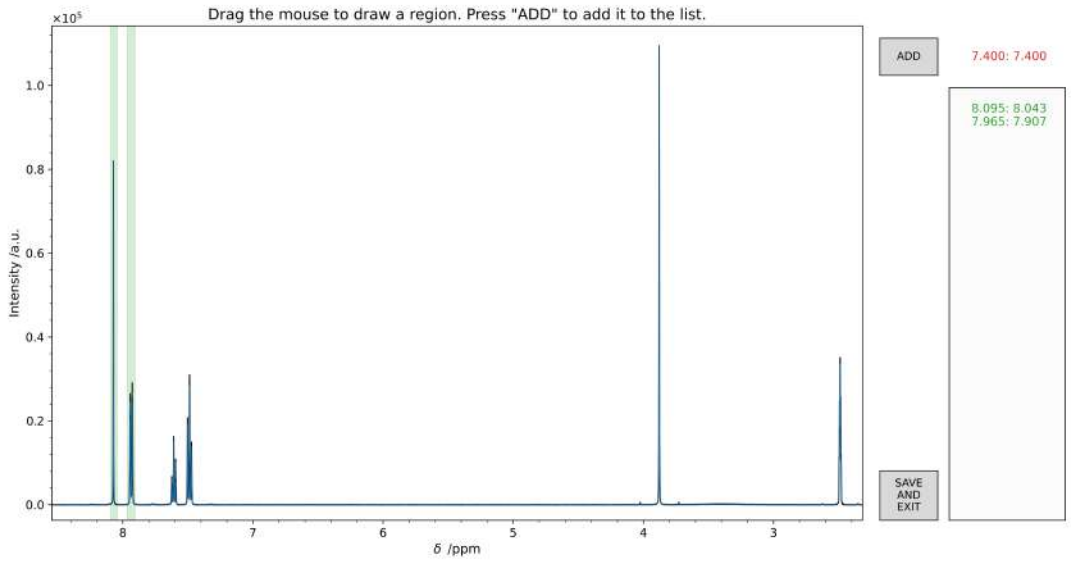


Figure 6.6: GUI for the selection of the regions to be used in the fit. The user can drag the red span with the mouse until it covers the desired region. Upon clicking on "ADD", the selection becomes permanent, and the span changes to green. The process can continue until the "SAVE" button is pressed.

6.3.2 The alignment fit

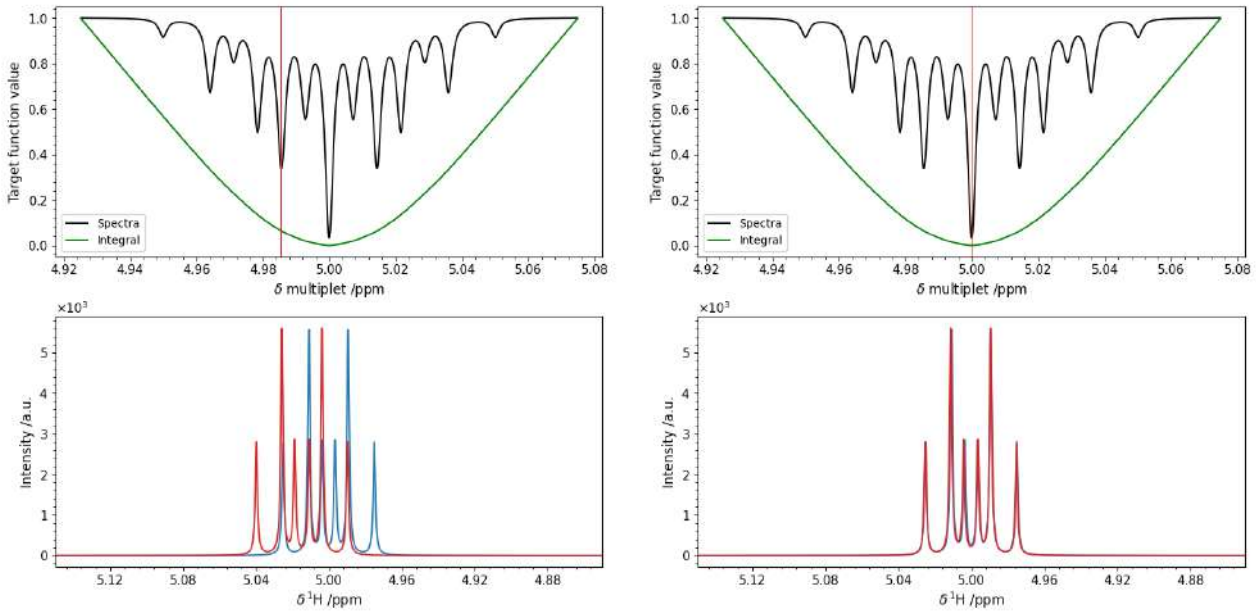
One unique feature of `pyIHM` is in the fit of the chemical shifts to get the correct peak positions. To illustrate the numerical issues that are associated to this part of the fit, let us suppose that we know exactly the structure of a given multiplet and the relative intensities of the fine structure of the multiplet, but that we do not know its chemical shift. In this case, the optimization of the target function would require changing a single parameter (that is, the chemical shift, δ). In this scenario, we can draw the complete target function across the values of δ (black trace in the top panels of figure 6.7). From this graph, it is apparent that the numerical problem is that a little misalignment between the experimental and calculated signal translates in several narrow local minima in the target function, which will create problems for most algorithms. Depending on the fine structure of the signal, the actual shape of the error surface may change, but its roughness remains a common feature.

To find the correct alignment, instead of going for global optimizers, which are slower than the local ones, we selected a target function that gives rise to a smoother error surface, as proposed earlier for EPR spectroscopy.[59,60,61] A convenient choice is the target function $\varphi^{\mathrm{algn}}(\omega)$ computed as the difference between the integral (i.e. the cumulative sum) of the experimental and calculated spectrum (equation 6.9), depicted in this case as green trace in figure 6.7.

$$\varphi^{\mathrm{algn}}(\omega) = \left\| \sum_{j=0}^{k} \left((\mathbb{S}^{\mathrm{exp}})[j] \right) - \sum_{j=0}^{k} \left(\left(\mathbb{S}^{\mathrm{model}}(\omega) \right)[j] \right) \right\|_2^2 \tag{6.9}$$

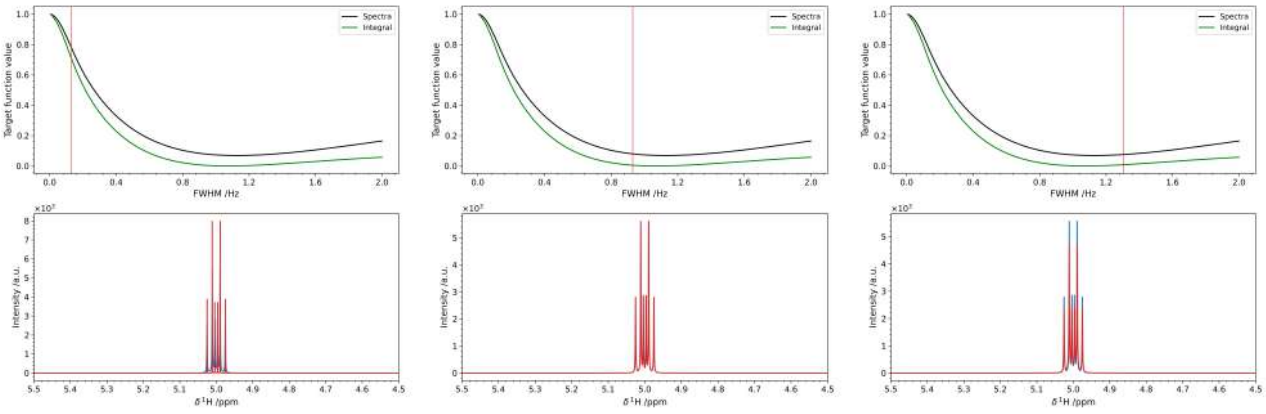
Of note, the alignment fit with this latter target function is insensitive to linewidth overestimates (see figure 6.8). At the end of this fit, the variation window for the chemical shifts in the core optimization of `pyIHM` is decreased to 1/10 of the original one set in the input file.

An extremely careful optimization of the initial guess through the interfaces described in the previous section allows to obtain a perfect alignment of the chemical shifts in a manual fashion. In this case, this alignment fit is redundant, and it can be skipped by setting the flag `--noalgn` when running the program.



(a) Partial superimposition. **(b)** Complete superimposition.

Figure 6.7: A signal centered at 5 ppm was simulated at 700 MHz ^{1}H Larmor frequency as a doublet of triplets with scalar coupling constants $J = 15$ and 10 Hz (blue trace, bottom panels). All the six features of the multiplet have the same linewidth of 1 Hz. The model signal (red trace, bottom panels) was simulated with the same parameter, and its chemical shift is varied from 4.975 to 5.075 ppm in steps of $6.10 \cdot 10^{-4}$ ppm. For each of these values, the value of the target function was computed on the difference of the spectra (equation 6.10, black trace in the top panels) and of their integrals (equation 6.9, green trace in the top panels). Here are shown two particular values of chemical shift, marked with a red line on the top panels: one correspondent to partial superimposition of the features (a) and the true value (b). The figure shows that a partial superimposition between experimental and model spectrum does not correspond to a secondary minimum of the integral target function.



(a) Underestimated linewidth. **(b)** Actual linewidth. **(c)** Overestimated linewidth.

Figure 6.8: A signal centered at 5 ppm was simulated at 700 MHz ^{1}H Larmor frequency as a doublet of triplets with scalar coupling constants $J = 15$ and 10 Hz (blue trace, bottom panels). All the six features of the multiplet have the same linewidth of 1 Hz. The model signal (red trace, bottom panels) was simulated with the same parameter, and its FWHM is varied from $5 \cdot 10^{-3}$ to 2 Hz in steps of $5 \cdot 10^{-3}$ Hz. For each of these values, the value of the target function was computed on the difference of the spectra (equation 6.10, black trace in the top panels) and of their integrals (equation 6.9, green trace in the top panels). Here are shown three particular values of FWHM, marked with a red line on the top panels: one lower than the actual one (a), the true value (b), and a greater one (c). The figure shows that both target functions have only one very broad minimum, thus they are quite insensitive to variations of the linewidth.

6.3.3 `pyIHM` core fit

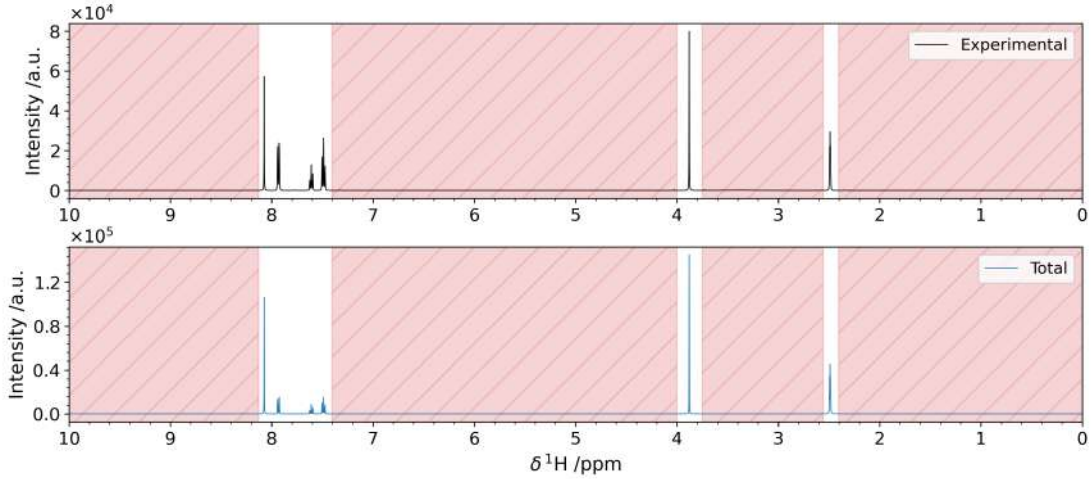
The goal of `pyIHM` is to find the optimal set of parameters $\hat{\mathbb{x}}$ that minimize the target function $\varphi(\mathbb{x})$ shown in equation 6.10, where the array $\mathbb{c}$ accounts for the relative concentrations of the component spectra $\mathbb{S}_k$.

$$\varphi(\mathbb{x}) = \|\mathbb{S}^{\text{exp}} - f(\mathbb{x})\|_2^2, \quad f(\mathbb{x}) = \mathbb{f} = \sum_{k=0}^{m-1} \mathbb{c}[k]\, \mathbb{S}_k, \quad \mathbb{x} = (\mathbb{c}^T|\mathbb{K}^T|\boldsymbol{\omega}^T|\mathbb{\Gamma}^T|\boldsymbol{\beta}^T) \tag{6.10}$$

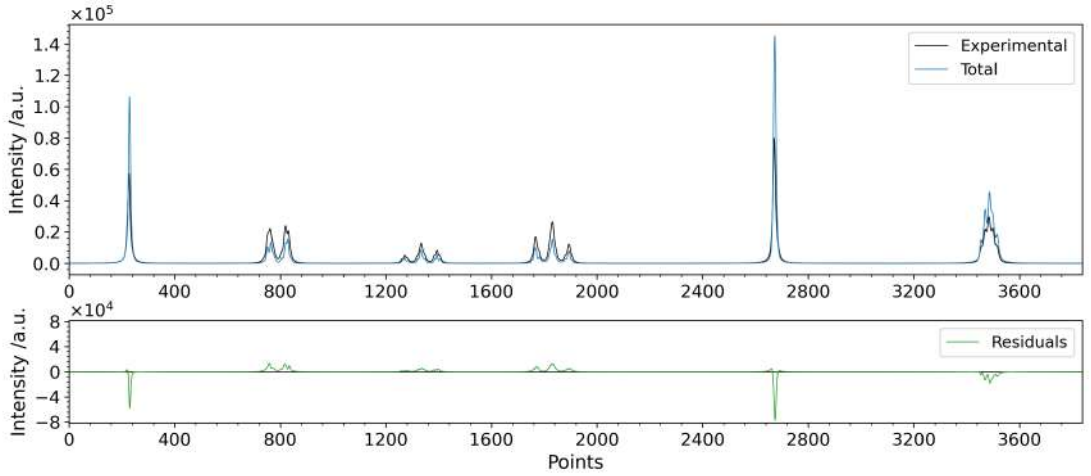
To do so, the target function is computed in several steps at every step of the optimization.

1. The array of the parameters is advanced according to the rules decided by the fitting algorithm.
2. The FID of each signal of each spectrum is computed using the new set of parameters using the time-domain Voigt model (equation 2.2). Then, the FIDs of each component is Fourier-transformed, and weighted for its concentration in the mixture. This is the model function $f(\mathbb{x})$ in equation 6.10.
3. Both the mixture spectrum $\mathbb{S}^{\text{exp}}$ and the model spectrum $\mathbb{f}$ are cropped around the selected regions for the fit, as shown in figure 6.9a, to give the resized arrays $\mathbb{S}'^{\text{exp}}$ and $\mathbb{f}'$.
4. The array of residuals $\mathbb{r} = \mathbb{S}'^{\text{exp}} - \mathbb{f}'$ is computed (figure 6.9b).
5. The target function $\varphi = \|\mathbb{r}\|_2^2$ is evaluated.

The user can select customized fitting settings, that will be listed under the `FIT_KWS` key of the input file, or they can choose among two default modes: "fast", that corresponds to a single optimization with Levenberg-Marquardt algorithm, and "tight", consisting of a Nelder-Mead optimization followed by another one, performed with Levenberg-Marquardt algorithm. The "tight" option might be preferred over the "fast" one because of the different behavior of the two optimization algorithms. Levenberg-Marquardt is better suited for refinement of the fit results near the optimum, as it falls towards the nearest minimum. Instead, a prior optimization step with the Nelder-Mead simplex is able to drive the parameters closer to the global minimizer, hence improving the overall result of the fit[26]. However, this comes at the cost of a substantial increase of the time required for the optimization.



(a) Cropping of the spectra



(b) Computation of the residuals

Figure 6.9: Graphical representation of two crucial steps in the computation of the target function for a `pyIHM` optimization. In panel **a**, the experimental spectrum (black) and the model function (blue) are cropped around the user-defined regions that delimit the interested signals. Then, in panel **b**, the trimmed regions are joined together, and the array of residuals (green) is computed.

6.3.4 Presentation of the results

The outcome of the fit is the optimized set of parameters $\hat{\mathbb{x}}$, of which the interesting ones from the quantitative point of view are the concentrations $\hat{\mathbb{c}}$. However, these are not the relative concentrations, as they do not sum up to 1, and they have to be further corrected. For ease of reading, the $\hat{}$ superscript will be omitted.

As a consequence of how the input is generated, before the start of the optimization step the relative intensities of the k-th pure component sum up to $\mathbb{H}[k]$. Since the relative intensities $\mathbb{K}$ are indeed a parameter of the fit, this condition is not met anymore, and this deviation from the theoretical values must be transferred to $\mathbb{c}[k]$ as:

$$\mathbb{c}[k]^{\text{corr.}} = \mathbb{c}[k] \frac{\sum_{j=0}^{N_k} \mathbb{K}_k[j]}{\mathbb{H}[k]} \qquad \mathbb{K}_k^{\text{corr.}}[j] = \frac{\mathbb{K}_k[j]}{\sum_{j=0}^{N_k} \mathbb{K}_k[j]} \mathbb{H}[k] \tag{6.11}$$

In this way, the original integrals of the pure component are restored to their theoretical value.

Finally, the actual relative concentrations $\bar{\mathbb{c}}$ are computed:

$$\bar{\mathbb{c}}[k] = \frac{\mathbb{c}[k]^{\text{corr.}}}{\sum_{j=0}^{m-1} \mathbb{c}[j]^{\text{corr.}}} \tag{6.12}$$

The outcome of the whole `pyIHM` run are collected in a *.out* file, which contains the relative concentrations of the components, as well as the information to recompute the spectra that fit the mixture. The spectra are also saved in a *.csv* file for convenience.

6.4 Applications of `pyIHM`

6.4.1 Mixtures of internal standards

The examples of application of `pyIHM` reported in this section are about very simple and well-resolved spectra. This choice is justified by the need of reference data to prove the reliability of the program, which necessarily comes from the peak integration approach. It is therefore needed that the resolution of the signals fulfils the requirements for a quantitatively accurate analysis. In this section, the results were obtained using the "tight" option in the core fit.

As first result, I present the fit of a mixture of two internal standards commonly used in quantitative NMR, benzoic acid (BzAc) and dimethylterephtalate (DMTP) in deuterated dimethylsulfoxide (DMSO-d6). In this example, the positions of the initial guess was perfectly superimposed with the experimental spectrum, leaving the concentrations as the only parameters to be determined. The input file for the fit and the output file obtained at the end of the optimization are listed in section A7.2. Figure 6.10 and table 6.1 show the results of this fit. The deconvolution is very good, as the fit residuals are not polarized and tightly clustered around the mean (figure A7.1), and the intensities are in line with the ones obtained with the traditional integration approach. It should be noted that, since it is deuterated, the actual amount of DMSO in the mixture will not be accurate. However, this is not relevant for our analysis, because we are comparing the `pyIHM` results with the integrals of the spectra, which bear the same information.

To prove the power of the chemical shift optimization described in the previous section, I analyzed a mixture of two different internal standard than in the previous test, benzoic acid and ethyl carbonate (EC) in DMSO-d6, whose chemical shifts were not aligned with the experimental signals. The input file for the fit and the output file obtained at the end of the optimization are listed in section A7.3. As shown in figure 6.11 and table 6.2, also in this case

Table 6.1: Relative concentrations of the mixture of BzAc and DMTP in DMSO, obtained by `pyIHM` and with the traditional integration approach.

Component	from `pyIHM`	from integr.	Diff.
BzAc	59.82%	60.24%	0.42%
DMSO	19.82%	19.89%	0.07%
DMTP	20.36%	19.87%	0.49%

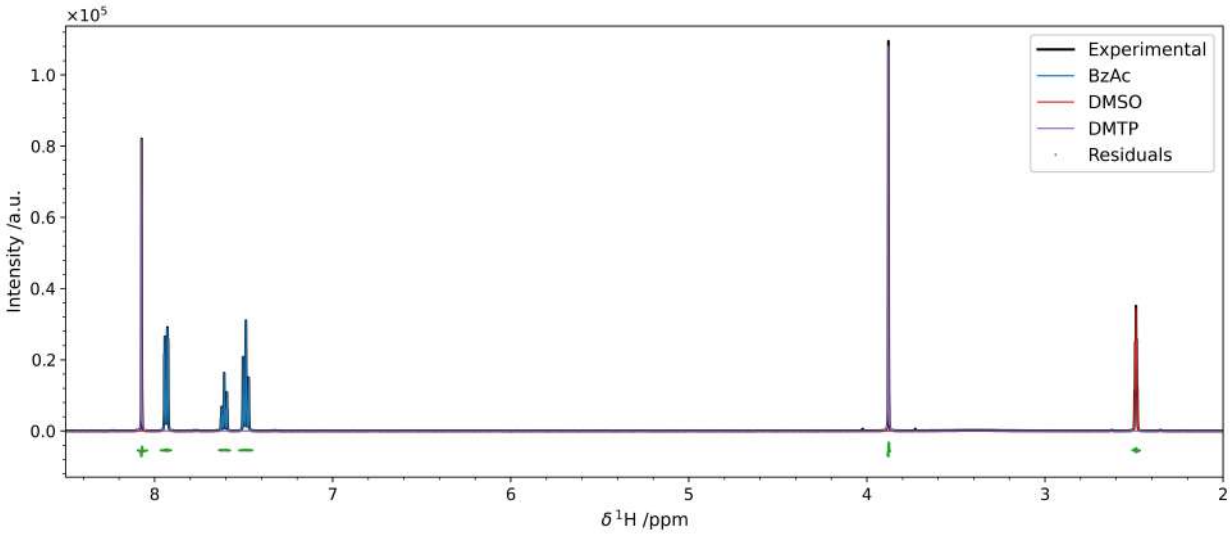


Figure 6.10: Results of the `pyIHM` run on the mixture of BzAc and DMTP in DMSO. The residuals are shown in green with a slight offset. The obtained concentrations are reported in table 6.1.

the results are comparable with the integrated intensities. The histogram of the residuals for this deconvolution is reported in figure A7.2.

Table 6.2: Relative concentrations of the mixture of BzAc and EC in DMSO, obtained by `pyIHM` and with the traditional integration approach.

Component	from `pyIHM`	from integr.	Diff.
BzAc	49.47%	50.24%	0.77%
DMSO	12.96%	12.75%	0.21%
EC	37.57%	37.01%	0.56%

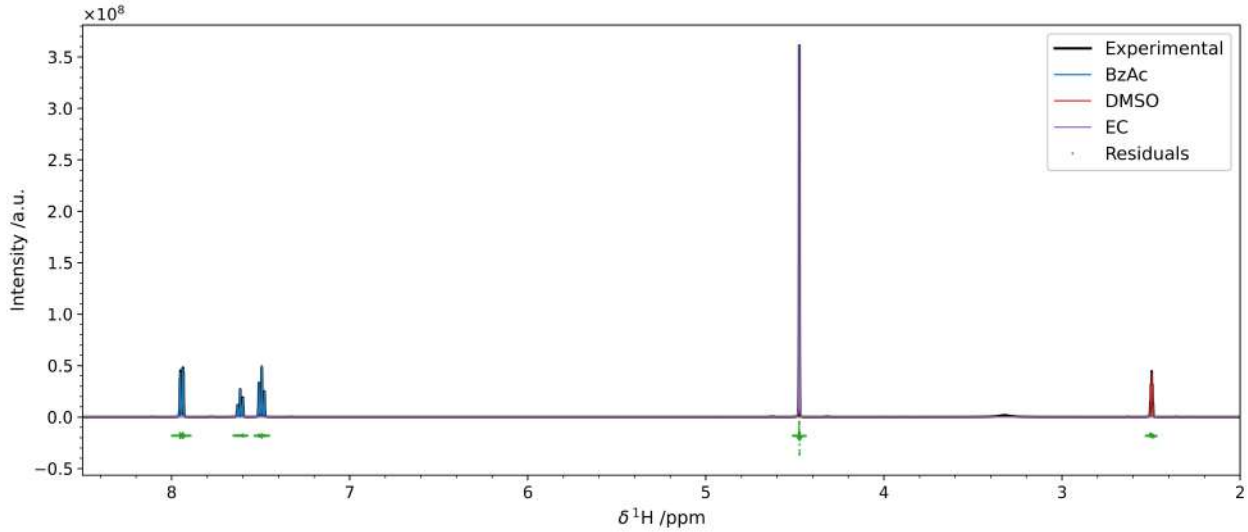


Figure 6.11: Results of the `pyIHM` run on the mixture of BzAc and DMTP in DMSO. The residuals are shown in green with a slight offset. The obtained concentrations are reported in table 6.2.

6.4.2 Purity assessment of a sample

Here I show a specific branch of mixture quantification, which is the purity check of a given compound. In this case, the mixture is composed of known amounts of analyte (ochratoxin-a) and internal standard (tetrachloronitrobenzene, TCNB). To this end, we weighted $m_A = 6.8033\,\text{mg}$ of analyte sample and $m_S = 4.8674\,\text{mg}$ of 99.79% pure internal standard, which correspond to $n_A = \kappa \times 16.848\,\text{mmol}$ of ochratoxin-a and $n_S = 18.618\,\text{mmol}$ of TCNB. The theoretical ratio between the two components is 0.90493. The relative concentrations given by `pyIHM` are $x_A^f = 0.4684$ and $x_S^f = 0.5316$, with an intensity ratio of 0.8811. The purity κ of ochratoxin-a is then simply the ratio between the observed intensity ratio and the theoretical intensity ratio $\kappa = 0.8811/0.90493 = 0.9737$. The obtained value of 0.9737 is comparable with the value obtained with gravimetric analysis, which is 0.983, thus proving the reliability of the approach. The input file for the fit and the output file obtained at the end of the optimization are listed in section A7.4. The result of the deconvolution is shown in figure 6.12, and the histogram of the residuals is reported in figure A7.3.

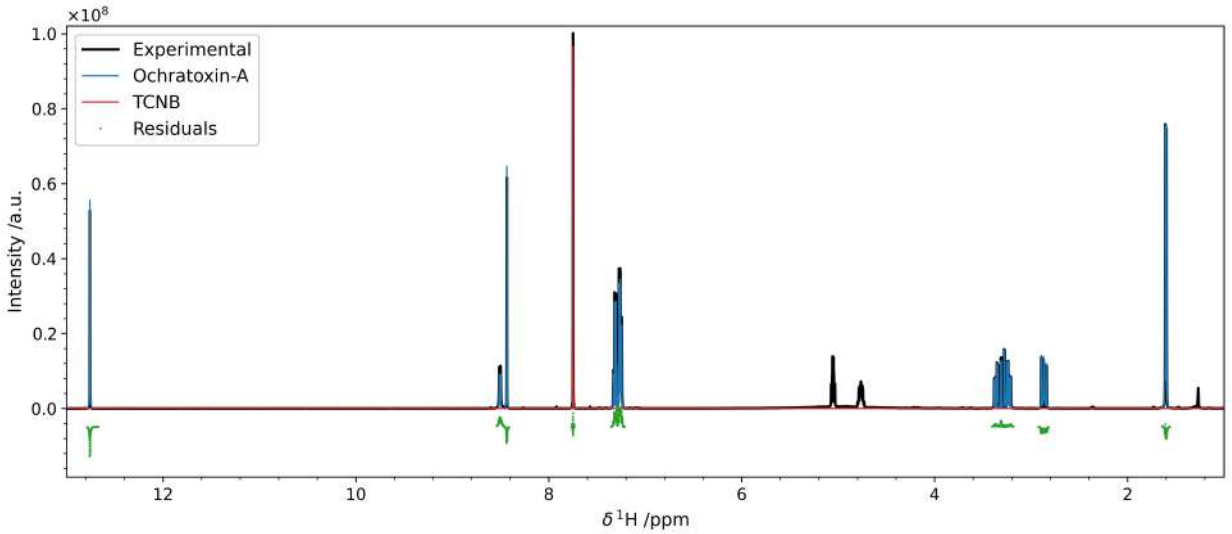


Figure 6.12: Results of the `pyIHM` run for the sample of ochratoxin-A in presence of TCNB as internal standard.

6.4.3 Simulated urine sample

A more challenging application would be the fit of a real mixture of unknown concentration. To this end, a mock urine sample composed of eleven metabolites was used[62]. The input file for the fit and the output file obtained at the end of the optimization are listed in section A7.5. The result of the deconvolution is shown in figure 6.13, and the obtained composition is reported in table 6.3. The histogram of the residuals for this deconvolution is reported in figure A7.4.

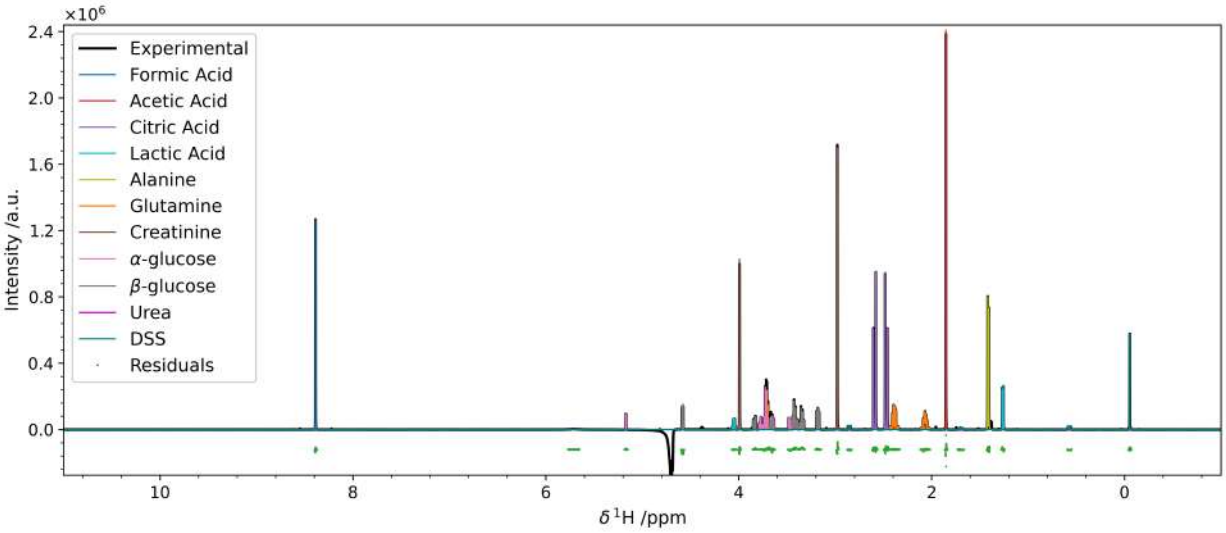


Figure 6.13: Results of the `pyIHM` deconvolution of the synthetic urine spectrum. The obtained concentrations are listed in table 6.3.

The quantitative assessment of a few metabolites was also performed with the Chenomx software suite. The results for the quantification of formate are shown in figure 6.14. Although the fit seems to be acceptable, the returned intensity of formate with respect to DSS was estimated to be about 31, which is very distant from both the 13.7 given by `pyIHM` and from the true value of around 10. An explanation of such high inaccuracy can be found in the incorrect linewidth estimation in the bottom panel of figure 6.14.

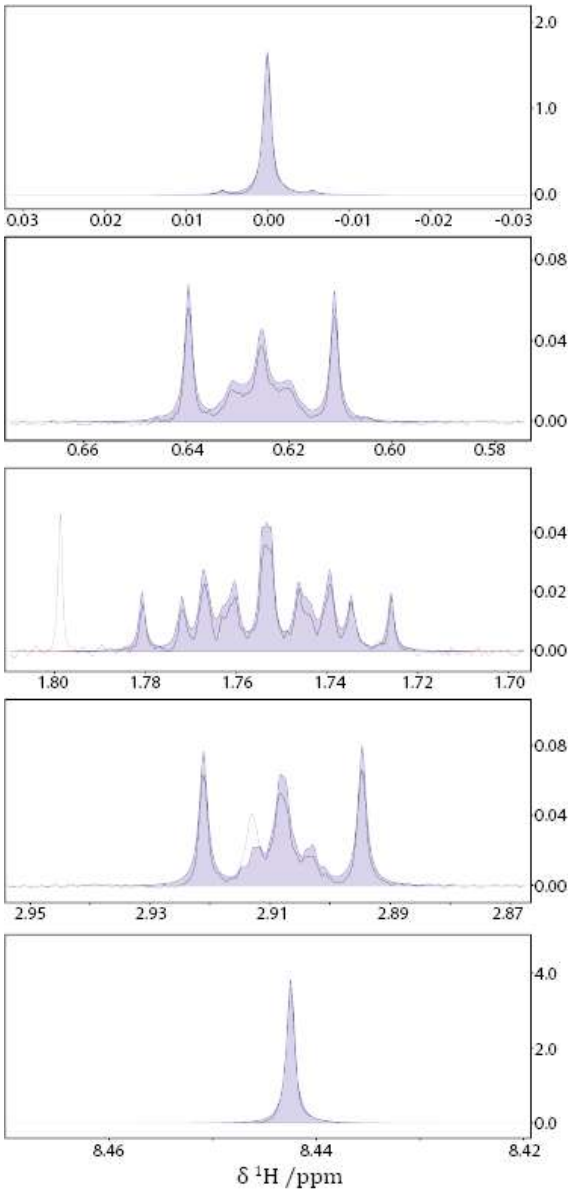


Figure 6.14: Results of the fit performed by Chenomx on DSS (1st to 4th panel) and on formate (bottom panel). The concentrations estimated by the program yield an intensity ratio of 31.663. This high discrepancy from the true value of around 10 can be explained by the incorrect linewidth estimation of formate in the bottom panel.

Table 6.3: Relative concentrations of the synthetic urine spectrum obtained by `pyIHM` , compared with the one obtained by a Chenomx expert user. The real concentrations are about $1\,\mathrm{mmol\,dm^{-3}}$ DSS, and about $10\,\mathrm{mmol\,dm^{-3}}$ for all the other metabolites. Of note, the concentration of lactate was poorly estimated from the calculations due to the high viscosity of the sample. It was also impossible to assess the relative amounts of α and β-glucose in the glucose sample *a priori*. The quantification of urea by NMR is known to be very inaccurate.

Component	**Rel. conc.**		**Rel. DSS**	
	`pyIHM`	Chenomx	`pyIHM`	Chenomx
Formic Acid	16.658%	22.994%	13.720	31.663
Acetic acid	13.919%	14.451%	11.464	19.899
Citrate	15.717%	17.858%	12.945	24.592
Lactate	3.128%	2.252%	2.577	3.101
Alanine	9.634%	10.380%	7.935	14.294
Glutamine	12.054%	9.723%	9.928	13.390
Creatinine	13.799%	11.175%	11.365	15.389
α-glucose	6.045%		4.979	
β-glucose	7.063%		5.817	
Glucose total	13.108%	8.237%	10.797	11.343
Urea	0.768%	2.203%	0.633	3.034
DSS	1.214%	0.726%	1.000	1.000

7. Conclusions

NMR spectroscopy is sometimes regarded as a very complex technique. This is not necessarily a complete misconception. The journey from the acquisition of the raw FID to the outcome of the data analysis can be challenging, if compared to the manipulation needed for (e.g.) UV-Vis spectroscopy, and each step of the workflow can contribute with errors if not applied carefully.

We presented a comprehensive exploration of `KLASSEZ` , an open-source python suite for the processing and analysis of NMR spectra. The routines implemented in `KLASSEZ` for the processing, i.e. the conversion from raw data to a spectrum ready for analysis, have been explained in great detail, putting particular effort in the rigorous mathematical description of the function according to the state-of-the-art NMR theory. The pros and cons of applying windowing functions, zero-filling, linear prediction, phase correction and baseline computation have been discussed, together with the actual implementation of the techniques. Also, an innovative method for the simultaneous and synergic baseline and phase correction has been presented.

`KLASSEZ` also implements method to perform denoising on spectra. This kind of processing is able to increase the overall quality of the spectra, as well as to make the fit of secondary properties from the data more robust. We showed this aspect by simulating the evolution of the signal intensities according to the cross-polarization of five synthetic peaks as function of the number of simulated experiments and their number of scans. The SVD-based denoising approach proved to be quite effective in both recovering the signals buried under the noise level, and estimating the two parameters that control the evolution of the intensities. In a similar manner, we also simulated the NMR monitoring of a chemical reaction. The denoising performed using Multivariate Curve Resolution was very effective in increasing the overall quality of the spectra used as snapshots of the reaction course. This in turn caused the dispersion of the intensity trends over the reaction time to significantly decrease, thus boosting the accuracy of the kinetic constant that can be gathered by fitting such trends.

Regarding the analysis perspective, `KLASSEZ` offers a very user-friendly set of interfaces for the deconvolution of spectra. These have been successfully use for the analysis of data coming from several application fields, proving to be a reliable tool. In particular, the deconvolution of solid-state NMR spectra recorded to investigate the structural changes of graphene oxide catalysts during the Povarov reaction and ring-opening of cyclobutanols showed that the catalyst deactivates due to the adsorption of aliphatic molecules on its surface. This deactivation is evident in the reduction of aromatic signals and the emergence of aliphatic signals in the NMR spectra. In addition, the epoxy-hydroxyl ratio remained constant during the ring-opening reaction, suggesting that epoxy ring opening is not a necessary step in the catalytic process.

Furthermore, a dedicated software for the quantitative analysis of mixtures, `pyIHM` , has been presented. The program is based on the Indirect Hard Modelling approach, but takes it to the open-source platform, and offers an interesting alternative when the traditional approach of peak integration cannot be used for quantitative purposes. `pyIHM` is highly customizable, to adapt it to a wide plethora of mixture spectra, and features a particular algorithm for chemical shift alignment based on the minimization of the integrals that greatly improves the

performance of the fit methods. The program was tested on several spectra, ranging from very simple mixtures of internal standards, to a mock sample of urine consisting of 11 metabolites. The results were compatible with alternative approaches (when applicable), but they come with the increased robustness due to the use of the whole spectrum instead of focusing of single, isolated peaks. This increased accuracy became particularly evident when comparing the results of the `pyIHM` deconvolution of the mock urine sample with the ones given by a commercial software. The quantification given by the latter method of a few metabolites, e.g. the formate, showed a significant discrepancy from the true value (about 3 times higher), maybe due to an incorrect linewidth estimation.

The main point of strength of `KLASSEZ` is its modular structure, that allows to integrate its routines in tailored scripts with great ease, without forcing the user to use a specific method for either processing or analysis. In addition, it is compatible with 1D and 2D data coming from all the main spectrometers vendors, thus proving to be an excellent tool for automated and customized analysis.

`KLASSEZ` is still in development: new functions are constantly added, and the existing ones are refined to improve performance and usability. In the near future, the field application of the software will be expanded to include the processing and analysis of 3D data and Diffusion Ordered (DOSY) spectra.

8. Methods

8.1 Softwares

8.1.1 KLASSEZ

KLASSEZ is a modular, open-source python software that can offer processing and analysis routines for NMR data. It was developed keeping in mind the potential needs of users from academia and industry as an alternative (or, compatible) tool to commercial softwares. The models and the interfaces to use and test them were implemented in python. The computer specifications, the employed packages and their versions are listed in table 8.1

KLASSEZ is freely available for download on GitHub (https://github.com/MetallerTM/klassez) and PyPI (https://pypi.org/project/klassez/).

Table 8.1: Computer specifications, employed packages and their versions. The characteristics of the hardware make the employed laptop fit in the low-medium range performance. A particularly weak component is the processor, which operates in low energy consumption by construction, thus sacrificing the performance.

Hardware model	Acer Aspire A315-55G
CPU	Intel Core i7-10510U
Memory	16 GB
Operative System	Ubuntu 22.04 − 24.04 64-bit

Package	**Version**
Python	3.10.12 − 3.12.3
Numpy	1.26.0 − 1.26.4
Scipy	1.11.2 − 1.14.0
Matplotlib	3.7.3 − 3.9.0
Seaborn	0.13.2
Lmfit	1.2.2 − 1.3.1
Csaps	1.1.0
Nmrglue	0.9 − 0.10
Jeol-parser	0.1.2

8.1.2 pyIHM

pyIHM is an open-source python program for the quantification of mixtures through the Indirect Hard Modelling approach. The full rationale behind the functioning of pyIHM is presented and discussed in detail in chapter 6. For the reading and processing of the mixture spectrum, as well for the generation and manipulation of the spectra of the components of the mixture, pyIHM relies on KLASSEZ routines. Hence, the computer specifications of the computer employed

for development and testing, and the required additional packages are the same of `KLASSEZ` (table 8.1).

`pyIHM` is freely available for download on GitHub (`https://github.com/MetallerTM/pyihm`) and PyPI (`https://pypi.org/project/pyihm/`).

8.2 Experimental data

The NMR experiments presented throughout the thesis work were acquired on several instruments.

8.2.1 Solid State NMR

All solid-state NMR experiments were recorded on a Bruker Avance II spectrometer operating at 16.4 T, corresponding to 700 MHz ^{1}H Larmor frequency, 139 MHz ^{29}Si Larmor frequency and 176 MHz ^{13}C Larmor frequency. The spectrometer is equipped with a 3.2 BVT MAS probehead in double resonance mode. Cross-polarization was achieved by matching the $k = 1$ Hartmann–Hahn condition[63]. During the ^{1}H magnetization evolution under the chemical shift in the indirect dimension of heteronuclear correlation experiments, the PMLG decoupling sequence was used to suppress the homonuclear dipolar couplings. The ^{29}Si spectra were acquired with CPMG echo train acquisition, and then the echoes were coadded.

8.2.2 Mixtures

The NMR experiments were performed on a 500 MHz NMR spectrometer system (VNMRS500, Varian Associates, Palo Alto, USA) operating at ^{1}H Larmor frequency of 499.9 MHz, which was equipped with a 5 mm OneNMR probe. The acquisition was performed with accumulation of 16 to 32 scans. The relaxation delays were set to exceed 7-times T_1 relaxation time to ensure quantitative response from all nuclei. The T_1 relaxation times were either estimated by a preliminary inversion-recovery experiment or from certificate information of the used internal standard.

8.2.3 Mock urine sample

The NMR spectrum was acquired using a Bruker NEO spectrometer (Bruker BioSpin) operating at 600 MHz ^{1}H Larmor frequency, featured with a 5 mm PATXI probe. The Bruker `noesygppr1d` sequence was used for the acquisition. The relaxation delay was set to 25 s to ensure quantitativeness of the spectrum.

8.2.4 ^{1}H$-^{15}$N HSQC of CzrA and ^{15}N$-^{13}$C CON of CD4

The experiments were performed on a Bruker Avance NEO spectrometer operating at 16.4 T, corresponding to 700 MHz ^{1}H Larmor frequency, 176 MHz ^{13}C Larmor frequency and 71 MHz ^{15}N Larmor frequency.

The HSQC experiment was acquired with a triple-resonance TCI cryo-probe using the `hsqcfpf3gpphwg` Bruker pulse sequence, where the water suppression is achieved using the WATERGATE sequence.

The ^{13}C-detected saturation recovery CON was acquired with a triple-resonance TXO cryo-probe, optimized for carbon detection. The employed sequence was an in-house modified se-

quence, derived by the `c_con_iasq` Bruker sequence. The $^{13}C-^{13}C$ homonuclear decoupling in the direct dimension was achieved using the IPAP scheme[64].

A1. Adopted symbols and conventions

Symbol	Description
$\mathbb{N}$	Set of natural numbers
$\mathbb{Z}$	Set of integer numbers
$\mathbb{R}$	Set of real numbers
x, α	Scalar variable
(a, b)	Open interval (extremes not included): $x \in (a, b)$ if $a < x < b$
$[a, b]$	Closed interval (extremes included): $x \in [a, b]$ if $a \leq x \leq b$
i	Imaginary unit
z^*	Complex conjugate: $z = x + \mathrm{i}y, \quad z^* = x - \mathrm{i}y$
$\mathrm{Re}\{z\}$	Real part: $z = x + \mathrm{i}y, \quad \mathrm{Re}\{z\} = x$
$\mathrm{Im}\{z\}$	Imaginary part: $z = x + \mathrm{i}y, \quad \mathrm{Im}\{z\} = y$
δ_{ij}	Kronecker delta: $\delta_{ij} = 1$ if $i = j, 0$ otherwise
%	Module operator (remainder)
$\lceil x \rceil$	Ceiling operator: $\lceil x \rceil := \min\{y \in \mathbb{Z} : y \geq x\}$
$\lfloor x \rfloor$	Floor operator: $\lfloor x \rfloor := \max\{y \in \mathbb{Z} : y \leq x\}$
//	Integer division: $x//y := \lfloor x/y \rfloor$
$f(x, y\|a, b, c)$	Function with arguments x, y and parameters a, b, c
$*$	Convolution operator: $f(x) * g(x) = \int_{-\infty}^{+\infty} f(t)g(x-t)dt$
$\mathcal{FT}$	Fourier transform: $\mathcal{FT}\{f(x)\}(\omega) = \int_{-\infty}^{+\infty} f(t)e^{-\mathrm{i}\omega t}dt$ (on arrays, it is considered Fast Fourier Transform)
$\overline{\mathcal{FT}}$	Inverse Fourier transform: $\overline{\mathcal{FT}}\{f(\omega)\}(t) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} f(\omega)e^{\mathrm{i}\omega t}d\omega$ (on arrays, it is considered Inverse Fast Fourier Transform)
$\mathbb{a}$, $\mathbb{A}$	Array (if 1D, it is considered as column vector)
$\mathbb{a}[i]$	i-th element of the array $\mathbb{a}$ (count starts from 0)

Symbol	Description
$\mathbb{a}[a:b]$	Portion of array included between index a and index b: $\mathbb{a}[a:b] := \{\mathbb{a}[k] \in \mathbb{a}[a:b] \text{ if } k \in [a,b]\}$
$\mathbb{a}[i,j]$	Element at i-th row and j-th column of the array $\mathbb{a}$ (count starts from 0)
$\mathbb{a}[i,:]$	i-th row of the array $\mathbb{a}$ (count starts from 0)
$\mathbb{a}[:,j]$	j-th column of the array $\mathbb{a}$ (count starts from 0)
$\langle \mathbb{a} \rangle$	Mean of array $\mathbb{a}$: $\langle \mathbb{a} \rangle := \frac{1}{N} \sum_{k=0}^{N-1} \mathbb{a}[k]$
$\left\|\mathbb{a}\right\|_\infty$	Sup-norm of array $\mathbb{a}$: $\left\|\mathbb{a}\right\|_\infty = \max\{\mathbb{a}[k]\}$
$\left\|\mathbb{a}\right\|_p$	p-norm of array $\mathbb{a}$ ($p \in [1,\infty)$): $\left\|\mathbb{a}\right\|_p = (\sum_k \lvert a[k]\rvert^n)^{\frac{1}{n}}$
$\left\|\mathbb{A}\right\|_F$	Frobenius norm of matrix $\mathbb{A}$: $\left\|\mathbb{A}\right\|_F = \left(\sum_{i,j} \mathbb{A}[i,j]^2\right)^{\frac{1}{2}}$
$\mathbb{A}^T$	Transposed matrix: $\mathbb{A}^T[i,j] = \mathbb{A}[j,i]$
$\mathbb{A}^*$	Conjugate matrix: $\mathbb{A}^*[i,j] = (\mathbb{A}[i,j])^*$
$\mathbb{A}^H$	Hermitian-transposed matrix: $\mathbb{A}^H[i,j] = (\mathbb{A}[j,i])^*$
$\mathbb{A}^{-1}$	Inverse of matrix $\mathbb{A}$
$\mathbb{A}^+$	Moore-Penrose pseudoinverse of matrix $\mathbb{A}$
$\mathbb{C} = \mathbb{A}\mathbb{B}$	Matrix product: $\mathbb{C}[i,j] = \sum_k \mathbb{A}[i,k]\mathbb{B}[k,j]$
$\mathbb{C} = \mathbb{A} \odot \mathbb{B}$	Element-wise multiplication: $\mathbb{C}[i,j] = \mathbb{A}[i,j]\mathbb{B}[i,j]$
$\mathbb{C} = \mathbb{A} \otimes \mathbb{B}$	Kronecker product
$\mathbb{C} = (\mathbb{A}\lvert\mathbb{B})$	Column-wise matrix augmentation
$\mathbb{1}$	Identity matrix. Unless stated otherwise, it is assumed of opportune dimensions.
$\mathbb{0}$	Null matrix. Unless stated otherwise, it is assumed of opportune dimensions.

A2. Processing spectra in KLASSEZ

A2.1 1D spectra

Here there is a script for reading and processing 1D NMR data.

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

from klassez import *

# Be aware that this is a BASIC processing
# Read the documentation of the functions to see the full powers

if 1:
    # This example is for the simulated data
    s = Spectrum_1D('acqus_1D', isexp=False)
    s.to_vf()  # You can convert info on peaks to .ivf for fitting
else:
    # Use the following to read experimentals:
    spect = 'bruker', 'jeol', 'varian', 'magritek', 'oxford' # One of these
    s = Spectrum_1D(path_to_dataset, spect=spect)

# Setup the processing
#   Apodization
#       Follow the table in the user manual to see what reads what
s.procs['wf']['mode'] = 'em'
s.procs['wf']['lb'] = 5
#   Zero-filling
s.procs['zf'] = 2**14

#   Apply processing and do FT
s.process()
# Remove the digital filter
s.pknl()
# Phase correction
s.adjph()
# Plot the data
s.plot()
```

The method `Spectrum_1D.process` calls the function `processing.fp`, listed here below. The parameters are read from the `Spectrum_1D.procs` dictionary, which can be modified from within the script, and it is saved as a text file in the dataset directory.

```
def fp(data, wf=None, zf=None, fcor=0.5, tdeff=0):
    """
    Performs the full processing of a 1D NMR FID (data).
    --------
    Parameters:
    - data: 1darray
        Input data
    - wf: dict
        {'mode': function to be used, 'parameters': different from each function}
    - zf: int
        final size of spectrum
    - fcor: float
        weighting factor for the FID first point
    - tdeff: int
        number of points of the FID to be used for the processing.
    ------
```

```
    Returns:
    - datap: 1darray
        Processed data
    """
    # Window function
    datap = processing.td_eff(data, tdeff)
    if wf is not None:
        if wf['mode'] == 'qsin':
            datap = processing.qsin(datap, ssb=wf['ssb'])
        if wf['mode'] == 'sin':
            datap = processing.sin(datap, ssb=wf['ssb'])
        if wf['mode'] == 'em':
            datap = processing.em(datap, lb=wf['lb'], sw=wf['sw'])
        if wf['mode'] == 'gm':
            datap = processing.gm(datap, lb=wf['lb_gm'], gb=wf['gb_gm'], sw=wf['sw'], gc=wf['gc'])
        if wf['mode'] == 'gmb':
            datap = processing.gmb(datap, lb=wf['lb'], gb=wf['gb'], sw=wf['sw'])
    # Zero-filling
    if zf is not None:
        datap = processing.zf(datap, zf)
    # FT
    datap = processing.ft(datap, fcor=fcor)
    return datap
```

A2.2 2D spectra

Here there is a script for reading and processing 1D NMR data.

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

from klassez import *

# Be aware that this is a BASIC processing
# Read the documentation of the functions to see the full powers

if 1:
    # This example is for the simulated data
    s = Spectrum_2D('acqus_2D', isexp=False)
else:
    # For experimentals, at version 0.4a.7 klassez reads only 2D bruker
    s = Spectrum_2D(path_to_dataset)

# Setup the processing
#   Apodization
#       Follow the table in the user manual to see what reads what
#       REMEMBER: index 0 is F1, index 1 is F2, for procs
s.procs['wf'][1]['mode'] = 'em'
s.procs['wf'][1]['lb'] = 5
s.procs['wf'][0]['mode'] = 'qsin'
s.procs['wf'][0]['ssb'] = 2
#   Zero-filling
s.procs['zf'] = 512, 2048

#   Apply processing and do FT
s.process()
# Remove the digital filter
s.pknl()
# Phase correction
s.adjph()
# Plot the data
s.plot()

# Extract projections
ppm_f2 = 180
ppm_f1 = 10
s.projf1(ppm_f2)  # Extract F1 trace @ ppm_f2 ppm
f1 = s.Trf1[f'{ppm_f2:.2f}'] # Call it back: it is a Spectrum_1D object!
f1.plot()
s.projf2(ppm_f1)  # Extract F2 trace @ ppm_f1 ppm
f2 = s.Trf2[f'{ppm_f1:.2f}'] # Call it back: it is a Spectrum_1D object!
```

```
f2.plot()
```

The method `Spectrum_2D.process` calls the function `processing.xfb`, listed here below. The parameters are read from the `Spectrum_2D.procs` dictionary, which can be modified from within the script, and it is saved as a text file in the dataset directory.

```
            apod = processing.sin(pdata, ssb=wf['ssb'])/pdata
        if wf['mode'] == 'em':
            apod = processing.em(pdata, lb=wf['lb'], sw=wf['sw'])/pdata
        if wf['mode'] == 'gm':
            apod = processing.gm(pdata, lb=wf['lb'], gb=wf['gb'], sw=wf['sw'])/pdata
        pdata = pdata / apod
    return pdata



def xfb(data, wf=[None, None], zf=[None, None], fcor=[0.5,0.5], tdeff=[0,0], u=True, FnMODE='States-TPPI'):
    """
    Performs the full processing of a 2D NMR FID (data).
    The returned values depend on u: it is True, returns a sequence of 2darrays depending on FnMODE, otherwise
         just the complex/hypercomplex data after FT in both dimensions
    --------
    Parameters:
    - data: 2darray
        Input data
    - wf: sequence of dict
        (F1, F2); {'mode': function to be used, 'parameters': different from each function}
    - zf: sequence of int
        final size of spectrum, (F1, F2)
    - fcor: sequence of float
        weighting factor for the FID first point, (F1, F2)
    - tdeff: sequence of int
        number of points of the FID to be used for the processing, (F1, F2)
    - u: bool
        choose if to unpack the hypercomplex spectrum into separate arrays or not
    - FnMODE: str
        Acquisition mode in F1
    ------
    Returns:
    - datap: 2darray or tuple of 2darray
        Processed data or tuple of 2darray
    """

    data = processing.td_eff(data, tdeff)

    # Processing the direct dimension
    # Window function
    if wf[1] is not None:
        if wf[1]['mode'] == 'qsin':
            data = processing.qsin(data, ssb=wf[1]['ssb'])
        if wf[1]['mode'] == 'sin':
            data = processing.sin(data, ssb=wf[1]['ssb'])
        if wf[1]['mode'] == 'em':
            data = processing.em(data, lb=wf[1]['lb'], sw=wf[1]['sw'])
        if wf[1]['mode'] == 'gm':
            data = processing.gm(data, lb=wf[1]['lb'], gb=wf[1]['gb'], sw=wf[1]['sw'])
    # Zero-filling
    if zf[1] is not None:
        data = processing.zf(data, zf[1])
    # FT
    data = processing.ft(data, fcor=fcor[1])

    # Processing the indirect dimension
    # If FnMODE is 'QF', do normal transpose instead of hyper
    if FnMODE == 'QF':
        data = data.T
    else:
        data = processing.tp_hyper(data)

    # Window function
    if wf[0] is not None:
        if wf[0]['mode'] == 'qsin':
            data = processing.qsin(data, ssb=wf[0]['ssb'])
```

```
    if wf[0]['mode'] == 'sin':
        data = processing.sin(data, ssb=wf[0]['ssb'])
    if wf[0]['mode'] == 'em':
        data = processing.em(data, lb=wf[0]['lb'], sw=wf[0]['sw'])
    if wf[0]['mode'] == 'gm':
        data = processing.gm(data, lb=wf[0]['lb'], gb=wf[0]['gb'], sw=wf[0]['sw'])
# Zero-filling
if zf[0] is not None:
    data = processing.zf(data, zf[0])
# FT
# Discriminate between F1 acquisition modes
if FnMODE == 'States-TPPI':
    data = processing.ft(data, alt=True, fcor=fcor[0])
elif FnMODE == 'Echo-Antiecho' or FnMODE == 'QF':
    data = processing.ft(data, fcor=fcor[0])
else:
    raise NotImplementedError('Unknown acquisition mode in F1. Aborting...')
if FnMODE == 'States-TPPI' or FnMODE == 'QF':
    data = processing.rev(data)                # reverse data
# Transpose back
if FnMODE == 'QF':
    data = data.T
else:
    data = processing.tp_hyper(data)
# Unpack and/or return processed data
if u:                                       # unpack or not
    if FnMODE == 'QF':
        return data.real, data.imag
    else:
        return processing.unpack_2D(data)       # rr, ir, ri, ii
else:
    return data
```

A3. MCR implementation in KLASSEZ

Listing A3.1: This function can be used to denoise a 2D FID with MCR with KLASSEZ .

```
def main(data, nc, f=10, tol=1e-5, itermax=1e4, P='H', oncols=True):
    datap, C, S = mcr(data, nc, f, tol, itermax, P, oncols)
    return datap, C, S
```

A3.1 MCR main

Listing A3.2: Main MCR function. The implementation of SIMPLISMA and MCR-ALS are given in listings A3.3 and A3.4, respectively.

```
def mcr(input_data, nc, f=10, tol=1e-5, itermax=1e4, P='H', oncols=True):
    """
    This is an implementation of Multivariate Curve Resolution for the denoising of 2D NMR data.
    Let us consider a matrix D, of dimensions m x n, where the starting data are stored. The final purpose of MCR
         is to decompose the D matrix as follows:
        D = CS + E
    where C and S are matrices of dimension m x nc and nc x n, respectively, and E contains the part of the data
         that are not reproduced by the factorization.
    Being D the FID of a NMR spectrum, C will contain time evolutions of the indirect dimension, and S will
         contain transients in the direct dimension.

    The total MCR workflow can be separated in two parts: a first algorithm that produces an initial guess for the
          three matrices C, S and E (simplisma), and an optimization step that aims at the removal of the unwanted
          features of the data by iteratively filling the E matrix (MCR ALS).
    This function returns the denoised datasets, CS, and the single C and S matrices.
    -------
    Parameters:
    - input_data: 2darray or 3darray
        a 3D array containing the set of 2D NMR datasets to be coprocessed stacked along the first dimension. A
             single 2D array can be passed, if the denoising of a single dataset is desired.
    - nc: int
        number of purest components to be looked for;
    - f: float
        percentage of allowed noise;
    - tol: float
        tolerance for the arrest criterion;
    - itermax: int
        maximum number of allowed iterations
    - P: str or 2darray
        'H' for horizontal stacking, 'V' for vertical stacking, or custom matrix as explained in the description
             of mcr_stack
    - oncols: bool
        True to estimate S with processing.simplisma, False to estimate C.
    -------
    Returns:
    - CS_f: 2darray or 3darray
```

```
        Final denoised data matrix
    - C_f: 2darray or 3darray
        Final C matrix
    - S_f: 2darray or 3darray
        Final S matrix
    """

    # Get number of datasets (nds) from the shape of the input tensor
    if isinstance(input_data, list):
        nds = len(input_data)
    else:
        if len(input_data.shape) == 3:
            nds = input_data.shape[0]
        elif len(input_data.shape) == 2:
            nds = 1
            input_data = np.reshape(input_data, (1, input_data.shape[0], input_data.shape[1]))
        else:
            print('Input data is not a matrix!')
            exit()


    print('\n******************************************************')
    print('*                                                    *')
    print('*         Multivariate Curve Resolution              *')
    print('*                                                    *')
    print('******************************************************\n')

    D = processing.mcr_stack(input_data, P=P)      # Matrix augmentation

    # Get initial estimation of C, S and E
    C0, S0 = processing.simplisma(D, nc, f, oncols=oncols)

    # Optimize C and S matrix through Alternating Least Squares
    C, S = processing.mcr_als(D, C0, S0, itermax=itermax, tol=tol)

    # Revert matrix augmentation
    C_f, S_f = processing.mcr_unpack(C, S, nds, P)

    # Obtain the denoised data of the same shape as the input
    CS_f = [C_f[j] @ S_f[j] for j in range(nds)]

    # Reshape if no matrix augmentation is performed
    if nds == 1:
        CS_f = CS_f[0]
        C_f = C_f[0]
        S_f = S_f[0]

    print('\n******************************************************\n')

    return CS_f, C_f, S_f
```

A3.2 SIMPLISMA

Listing A3.3: Implementation of SIMPLISMA in KLASSEZ .

```
def simplisma(D, nc, f=10, oncols=True):
    """
    Finds the first nc purest components of matrix D using the simplisma algorithm, proposed by Windig and
         Guilment (DOI: 10.1021/ac00014a016 ). If oncols=True, this function estimates S with simplisma, then
         calculates C = DS+ . If oncols=False, this function estimates C with simplisma, then calculates S = C+ D.
          f defines the percentage of allowed noise.
    -------
    Parameters:
    - D: 2darray
        Input data, of dimensions m x n
    - nc: int
```

```
    Number of components to be found. This determines the final size of the C and S matrices.
- f: float
    Percentage of allowed noise.
- oncols: bool
    If True, simplisma estimates the S matrix, otherwise estimates C.
-------
Returns:
- C: 2darray
    Estimation of the C matrix, of dimensions m x nc.
- S: 2darray
    Estimation of the S matrix, of dimensions nc x n.
"""

rows = D.shape[0]     # number of rows of D
cols = D.shape[1]     # number of columns of D

if oncols:
    # on columns
    m = np.zeros(rows).astype(D.dtype)
    s = np.zeros(rows).astype(D.dtype)

    for i in range(rows):
        m[i] = np.mean(D[i,:])    # mean of the i-th row
        s[i] = np.std(D[i,:])     # STD of the i-th row

    # Correction factor for the noise 'alpha'
    a = 0.01 * f * max(m)

    print('Computing 1 purest variable...', end='\r')
    # Rescaling of data for lambda: makes determinant of COO
    # proportional only to the independance between variables
    l = ( s**2 + (m + a)**2 )**0.5 # lambda corrected for alpha
    Dl = np.zeros_like(D)
    for i in range(rows):
        Dl[i,:] = D[i,:] / l[i]

    Q = (1/cols) * Dl @ Dl.T   # Correlation-around-origin matrix

    # Calculation of the weighting factors:
    # express the independency between the variables
    w = np.zeros((rows, nc)).astype(D.dtype)   # Weights
    p_s = np.zeros((rows, nc)).astype(D.dtype) # Pure components spectra
    s_s = np.zeros((rows, nc)).astype(D.dtype) # STD spectra

    # First weight
    w[:,0] = (s**2 + m**2) / (s**2 + (m + a)**2)
    p0 = s / (m + a)  # First purity spectrum
    pv, ipv = [], []  # Purest variables and correspondant index

    p_s[:,0] = w[:,0] * p0
    s_s[:,0] = w[:,0] * s

    # 1st purest variable
    pv.append(max(p_s[:,0]))
    ipv.append(np.argmax(p_s[:,0]))

    # Matrix for computing the determinants
    #   It has the following structure, where Q denotes the COO matrix
    #   and p# the index of the # purest component:
    """
        Q[i,i]         Q[i,p1]        Q[i,p2]        ... Q[i,p(i-1)]
        Q[p1,i]        Q[p1,p1]       Q[p1,p2]       ... Q[p1,p(i-1)]
        Q[p2,i]        Q[p2,p1]       Q[p2,p2]       ... Q[p2,p(i-1)]
        ...            ...            ...            ... ...
        Q[p(i-1),i]    Q[p(i-1),p1]   Q[p(i-1),p2]   ... Q[p(i-1),p(i-1)]
    """
    for c in range(1, nc):   # 'c' cycles on number of components
        print('Computing '+str(c+1)+' purest variable...', end='\r')
        for i in range(rows): # i cycles on the number of rows
            W = np.zeros((c+1,c+1)).astype(D.dtype)
            W[0,0] = Q[i,i]
            for k in range(1, c+1):             # cycles inside W
                W[0,k] = Q[i,ipv[k-1]]          # first row \{0,0}
                W[k,0] = Q[ipv[k-1],i]          # first column \{0,0}
                for q in range(1, c+1):
```

```
                W[k,q] = Q[ipv[k-1],ipv[q-1]] # all the rest, going row per row
            w[i,c] = linalg.det(W)

        p_s[:,c] = p0 * w[:,c]           # Create pure spectrum of c-th component
        s_s[:,c] = s_s[:,0] * w[:,c]     # Create STD spectrum of c-th component
        pv.append(max(p_s[:,c]))         # Update pure component
        ipv.append(np.argmax(p_s[:,c])) # Update pure variable

    print('Purest variables succesfully found.\n')
    for c in range(nc):
        print('{} purest variable:\t\t{}'.format(c+1, ipv[c]))

    # MCR "S" matrix (D = CS + E)
    S = np.zeros((nc, cols)).astype(D.dtype)
    for c in range(nc):
        S[c,:] = D[ipv[c],:]
    C = D @ linalg.pinv(S)

else:
    # on rows
    m = np.zeros((cols)).astype(D.dtype)
    s = np.zeros((cols)).astype(D.dtype)

    for j in range(cols):
        m[j] = np.mean(D[:,j])    # mean of the i-th row
        s[j] = np.std(D[:,j])     # STD of the i-th row

    # Correction factor for the noise 'alpha'
    a = 0.01 * f * max(m)

    print('Computing 1 purest variable...', end='\r')
    # First purity spectrum
    p1 = s / (m + a)      # First purity spectrum
    pv, ipv = [], []      # Purest variables and correspondant index

    # 1st purest variable
    pv.append(max(p1))
    ipv.append(np.argmax(p1))

    # Rescaling of data for lambda: makes determinant of COO
    # proportional only to the independance between variables
    l = ( s**2 + (m + a)**2 )**0.5 # lambda corrected for alpha
    Dl = np.zeros_like(D)
    for j in range(cols):
        Dl[:,j] = D[:,j] / l[j]

    Q = (1/rows) * Dl.T @ Dl   # Correlation-around-origin matrix

    # Calculation of the weighting factors:
    # express the independency between the variables

    w = np.zeros((cols, nc)).astype(D.dtype)  # Weights
    p_s = np.zeros((cols, nc)).astype(D.dtype) # Pure components spectra
    s_s = np.zeros((cols, nc)).astype(D.dtype) # STD spectra

    # First weight
    w[:,0] = (s**2 + m**2) / (s**2 + (m + a)**2)
    p_s[:,0] = w[:,0] * p1
    s_s[:,0] = w[:,0] * s

    # Matrix for computing the determinants
    # It has the following structure, where Q denotes the COO matrix
    # and p# the index of the # purest component:
    """
        Q[j,j]         Q[j,p1]        Q[j,p2]        ... Q[j,p(j-1)]
        Q[p1,j]        Q[p1,p1]       Q[p1,p2]       ... Q[p1,p(j-1)]
        Q[p2,j]        Q[p2,p1]       Q[p2,p2]       ... Q[p2,p(j-1)]
        ...            ...            ...            ... ...
        Q[p(j-1),j]    Q[p(j-1),p1]   Q[p(j-1),p2]   ... Q[p(j-1),p(j-1)]
    """
    for c in range(1, nc):   # 'c' cycles on number of components
        print('Computing '+str(c+1)+' purest variable...', end='\r')
        for j in range(cols): # j cycles on the number of colums
            W = np.zeros((c+1,c+1)).astype(D.dtype)
            W[0,0] = Q[j,j]
```

```
            for k in range(1, c+1): # cycles inside W
                W[0,k] = Q[j,ipv[k-1]]      # first row \{0,0}
                W[k,0] = Q[ipv[k-1],j]      # first column \{0,0}
                for q in range(1, c+1):
                    W[k,q] = Q[ipv[k-1],ipv[q-1]] # all the rest, going row per row
            w[j,c] = linalg.det(W)

        p_s[:,c] = p_s[:,0] * w[:,c]   # Create pure spectrum of c-th component
        s_s[:,c] = s_s[:,0] * w[:,c]   # Create STD spectrum of c-th component
        pv.append(max(p_s[:,c]))       # Update pure component
        ipv.append(np.argmax(p_s[:,c])) # Update pure variable

    print('Purest variables succesfully found.\n')
    for c in range(nc):
        print('{} purest variable:\t\t{}'.format(c+1, ipv[c]))

    # MCR "C" matrix (D = CS + E)
    C = np.zeros((rows, nc)).astype(D.dtype)
    for c in range(nc):
        C[:,c] = D[:,ipv[c]]
    S = linalg.pinv(C) @ D

return C, S
```

A3.3 MCR-ALS

Listing A3.4: Implementation of the MCR Alternating Least Squares optimization in KLASSEZ .

```
def mcr_als(D, C, S, itermax=10000, tol=1e-5):
    """
    Performs alternating least squares to get the final C and S matrices. Being the fundamental MCR equation:
        D = CS + E
    At the k-th step of the iterative cycle:
        1. C(k) = DS+(k-1)
        2. S(k) = C+(k) D
        3. E(k) = D - C(k) S(k)
    Defined rC and rS as the Frobenius norm of the difference of C and S matrices between two subsequent steps:
        rC = || C(k) - C(k-1) ||
        rS = || S(k) - S(k-1) ||
    The convergence is reached when:
        rC <= tol && rS <= tol
    -------
    Parameters:
    - D: 2darray
        Input data, of dimensions m x n
    - C: 2darray
        Estimation of the C matrix, of dimensions m x nc.
    - S: 2darray
        Estimation of the S matrix, of dimensions nc x n.
    - itermax: int
        Maximum number of iterations
    - tol: float
        Threshold for the arrest criterion.
    -------
    Returns:
    - C: 2darray
        Optimized C matrix, of dimensions m x nc.
    - S: 2darray
        Optimized S matrix, of dimensions nc x n.
    """

    itermax = int(itermax)
    E = D - C @ S

    start_time = datetime.now()
    print('\n---------------------------------------------------\n')
```

```
    print('           MCR optimization running...         \n')

    convergence_flag = 0
    print( '#  \tC convergence\tS convergence')
    for kk in range(itermax):
        # Copy from previous cycle
        C0 = np.copy(C)
        E0 = np.copy(E)
        S0 = np.copy(S)

        # Compute new C, S and E
        C = D @ linalg.pinv(S)
        S = linalg.pinv(C) @ D
        E = D - C @ S

        # Compute the Frobenius norm of the difference matrices
        # between two subsequent cycles
        rC = linalg.norm(C - C0)
        rS = linalg.norm(S - S0)

        # Ongoing print of the residues
        print(str(kk+1)+' \t{:.5e}'.format(rC)+ '\t'+'{:.5e}'.format(rS), end='\r')

        # Arrest criterion
        if (rC < tol) and (rS < tol) and kk:
            end_time = datetime.now()
            print( '\n\n\tMCR converges in '+str(kk+1)+' steps.')
            convergence_flag = 1  # Set to 1 if the arrest criterion is reached
            break

    if not convergence_flag:
        print ('\n\n\tMCR does not converge.')
    end_time = datetime.now()
    print( '\tTotal runtime: {}'.format(end_time - start_time))

    return C, S
```

A3.4 Data augmentation using positioning matrix

Listing A3.5: Functions to perform data augmentation according to a positioning matrix, as described in section 4.2.2, and to unpack the obtained solution.

```
def mcr_stack(input_data, P='H'):
    """
    Performs matrix augmentation by assembling input_data according to the positioning matrix P.
    P has two default modes: 'H' = horizontal stacking; 'V' = vertical stacking. Otherwise, a custom P matrix can
        be given as follows.
    The entries of the P matrix are the indices of the data in input_data. The shape of the matrix determines the
        final arrangement.
    Example: if input_data is [a, b, c, d, e, f], and one wants to obtain [[a, b], [d,c], [f, e]] the
        correspondant P matrix is:
    [[0, 1], [3, 2], [5, 4]]
    If each dataset in input_data has dimensions (m, n) and P has dimensions (u,v), then the returned data matrix
        will have dimensions (mu, nv).
    -------
    Parameters:
    - input_data: 3darray
        Contains the spectra to be stacked together. The index that runs on the datasets must be the first one.
    - P: str or 2darray
        'H' for horizontal stacking, 'V' for vertical stacking, or custom matrix as explained in the description
    -------
    Returns:
    - data: 2darray
        Augmented data matrix.
    """
    # Get the number of datasets
```

```
    if isinstance(input_data, list):
        nds = len(input_data)
        Q = input_data
    else:  # if it is not a list, make it be manually
        nds = input_data.shape[0]
        Q = [input_data[w] for w in range(nds)]

    # Compute the P matrix
    if isinstance(P, str): # default options
        if P == 'H':  # Horizontal
            P = np.arange(nds).reshape(1,-1)
        elif P == 'V': # Vertical
            P = np.arange(nds).reshape(-1,1)
        else:  # Unknown
            raise ValueError('Unrecognized P type')
    elif isinstance(P, np.ndarray): # Check if the dimensions are compatible
        assert np.prod(P.shape) == nds, 'Wrong P shape'

    # Assemble the data
    for k in range(nds):
        # Compute mask matrix
        Mk = np.zeros_like(P)
        # Find the position of the k-th spectrum in P
        i, j = np.where(P == k)
        # Set that position as a 1 in Mk, all the rest is 0
        Mk[i[-1],j[-1]] = 1   # np.where returns lists of 1 number each
        if k == 0: # Make the variable
            data = np.kron(Mk, Q[k])
        else:      # Add it
            data += np.kron(Mk, Q[k])
    return data


def mcr_unpack(C, S, nds, P='H'):
    """
    Reverts matrix augmentation of mcr_stack.
    The denoised spectra can be calculated by matrix multiplication: D[k] = C_f[k] S_f[k], for k = 0,..., nds-1
    --------
    Parameters:
    - C: 2darray
        MCR C matrix
    - S: 2darray
        MCR S matrix
    - nds: int
        number of experiments
    - P: str or 2darray
        'H' for horizontal stacking, 'V' for vertical stacking, or custom matrix as explained in the description
             of mcr_stack
    ---------
    Returns:
    - C_f: list of 2darray
        Disassembled MCR C matrix
    - S_f: list of 2darray
        Disassembled MCR C matrix
    """
    # Compute the P matrix
    if isinstance(P, str): # default options
        if P == 'H':  # Horizontal
            P = np.arange(nds).reshape(1,-1)
        elif P == 'V': # Vertical
            P = np.arange(nds).reshape(-1,1)
        else:  # Unknown
            raise ValueError('Unrecognized P type')
    elif isinstance(P, np.ndarray): # Check if the dimensions are compatible
        assert np.prod(P.shape) == nds, 'Wrong P shape'

    # Compute the dimension of each original dataset
    m = C.shape[0] // P.shape[0] # num. rows of C / num. exp. per column
    n = S.shape[-1] // P.shape[-1] # num. columns of S / num. exp. per row

    # Initialize variables for storing the final C and S matrices
    C_f, S_f = [], []

    for k in range(nds):  # Loop on the datasets
        i, j = np.where(P == k)  # find whe position of the k-th dataset in P
```

```
    # Compute slices for delimiting the k-th spectrum
    #   in C: all the columns, rows according to P
    rslice = slice(i[0]*m, i[0]*m + m)
    C_f.append(C[rslice, ...])
    #   in S: all the rows, columns according to P
    cslice = slice(j[0]*n, j[0]*n + n)
    S_f.append(S[...,cslice])

return np.array(C_f), np.array(S_f)
```

A4. Least-Squares Optimized Intensity and Offset

A4.1 Intensity-only correction

Let us consider a set of N independent variables $\mathbb{x}$, for which correspond a set of N experimental observations $\mathbb{y}$. We assume that the data can be fitted with a model function $f(\mathbb{x}|\boldsymbol{\beta})$, that in turns depends on p parameters $\boldsymbol{\beta}$. Hence, the target function to be minimized in a least-squares optimization φ can be written as:

$$\varphi(\mathbb{x}|\boldsymbol{\beta}, I) = \sum_{k=0}^{N-1} \Big[\mathbb{y}[k] - I\, f(\mathbb{x}[k]|\beta) \Big]^2 \tag{A4.1}$$

where I is the intensity factor that matches the scaling difference between the experimental data and the model. The model function f can be nonlinear in the parameters $\boldsymbol{\beta}$. However, the intensity factor I indeed depends linearly with respect to the target function, therefore a closed form expression for I exists.

To lighten up the tractation, the explicit dependence from the set of parameters $\boldsymbol{\beta}$ will be omitted. All the summation will be assumed to always run for k from 0 to $N-1$. Equation A4.1 thus becomes

$$\varphi(\mathbb{x}|I) = \sum \Big[\mathbb{y}[k] - I\, f(\mathbb{x}[k]) \Big]^2 \tag{A4.2}$$

The best-fit solution is a minimum of the target function: in this point of the error surface, the differential of φ is equal to the null vector. It is therefore true that

$$\frac{\partial \varphi(\mathbb{x}|I)}{\partial I} = 0$$

Substituting the expression of φ and considering the linearity of both the summation and differential operator:

$$\begin{aligned}
\frac{\partial \varphi(\mathbb{x}|I)}{\partial I} &= \sum \frac{\partial}{\partial I} \Big[\big(\mathbb{y}[k] - I\, f(\mathbb{x}[k]) \big)^2 \Big] \\
&= 2 \sum \Big[\big(\mathbb{y}[k] - I\, f(\mathbb{x}[k]) \big) \big(-f(\mathbb{x}[k]) \big) \Big] \\
&= \cancel{-2} \Big[\sum \mathbb{y}[k] f(\mathbb{x}[k]) - I \sum f^2(\mathbb{x}[k]) \Big] = 0
\end{aligned}$$

Dividing by N allows to substitute the sums with the means:

$$\langle \mathbb{y} \odot f(\mathbb{x}) \rangle - I \left\langle f^2(\mathbb{x}) \right\rangle = 0$$

The optimal value for I is readily obtained as:

$$I = \frac{\langle \mathbb{y} \odot f(\mathbb{x}) \rangle}{\langle f^2(\mathbb{x}) \rangle} \tag{A4.3}$$

A4.2 Including the offset

To include a constant contribution q to the target function, equation A4.2 must be modified into

$$\varphi(\mathbb{x}|I,q) = \sum \Big[\mathbb{y}[k] - I\, f(\mathbb{x}[k]) - q \Big]^2 \tag{A4.4}$$

When imposing the minimum conditions, the partial derivatives of φ with respect to I and to q have to be zero at the same time:

$$\begin{cases} \dfrac{\partial \varphi(\mathbb{x}|I,q)}{\partial I} = 0 \\ \dfrac{\partial \varphi(\mathbb{x}|I,q)}{\partial q} = 0 \end{cases}$$

The partial derivative with respect to I yields:

$$\begin{aligned} \frac{\partial \varphi(\mathbb{x}|I,q)}{\partial I} &= \sum \frac{\partial}{\partial I} \Big[\big(\mathbb{y}[k] - I\, f(\mathbb{x}[k]) - q \big)^2 \Big] \\ &= 2 \sum \Big[\big(\mathbb{y}[k] - I\, f(\mathbb{x}[k]) - q \big) \big(-f(\mathbb{x}[k]) \big) \Big] \\ &= \cancel{2} \Big[\sum \mathbb{y}[k] f(\mathbb{x}[k]) - I \sum f^2(\mathbb{x}[k]) - q \sum f(\mathbb{x}[k]) \Big] = 0 \end{aligned}$$

Dividing by N and rearranging:

$$I \left\langle f^2(\mathbb{x}) \right\rangle + q \left\langle f(\mathbb{x}) \right\rangle = \langle \mathbb{y} \odot f(\mathbb{x}) \rangle \tag{A4.5}$$

Equivalently, the partial derivative with respect to q yields:

$$\begin{aligned} \frac{\partial \varphi(\mathbb{x}|I,q)}{\partial q} &= \sum \frac{\partial}{\partial I} \Big[\big(\mathbb{y}[k] - I\, f(\mathbb{x}[k]) - q \big)^2 \Big] \\ &= 2 \sum \Big[\big(\mathbb{y}[k] - I\, f(\mathbb{x}[k]) - q \big) \big(-1 \big) \Big] \\ &= \cancel{2} \Big[\sum \mathbb{y}[k] f(\mathbb{x}[k]) - I \sum f^2(\mathbb{x}[k]) - q \sum 1 \Big] = 0 \end{aligned}$$

Dividing by N, considering that $\sum_k 1 = N$, and rearranging:

$$I \left\langle f(\mathbb{x}) \right\rangle + q = \langle \mathbb{y} \rangle \tag{A4.6}$$

Equations A4.5 and A4.6 represents the terms of a 2-equations linear system in the unknowns I and q, that can be conveniently represented in matrix formalism as:

$$\mathbb{A}\mathbb{v} = \mathbb{w} \rightarrow \begin{pmatrix} \langle f^2(\mathbb{x}) \rangle & \langle f(\mathbb{x}) \rangle \\ \langle f(\mathbb{x}) \rangle & 1 \end{pmatrix} \begin{pmatrix} I \\ q \end{pmatrix} = \begin{pmatrix} \langle \mathbb{y} \odot f(\mathbb{x}) \rangle \\ \langle \mathbb{y} \rangle \end{pmatrix}$$

The problem can be elegantly solved by computing the inverse of the coefficient matrix $\mathbb{A}$, so that:

$$\mathbb{v} = \mathbb{A}^{-1}\mathbb{w} \rightarrow \begin{pmatrix} I \\ q \end{pmatrix} = \frac{1}{Q}\begin{pmatrix} 1 & -\langle f(\mathbb{x})\rangle \\ -\langle f(\mathbb{x})\rangle & \langle f^2(\mathbb{x})\rangle \end{pmatrix}\begin{pmatrix} \langle \mathbb{y} \odot f(\mathbb{x})\rangle \\ \langle \mathbb{y}\rangle \end{pmatrix}$$

where $Q = \det\{\mathbb{A}\} = \left\langle f^2(\mathbb{x})\right\rangle - \langle f(\mathbb{x})\rangle^2$

The final expressions for I and q therefore are:

$$I = \frac{\langle \mathbb{y} \odot f(\mathbb{x})\rangle - \langle \mathbb{y}\rangle \langle f(\mathbb{x})\rangle}{\left\langle f^2(\mathbb{x})\right\rangle - \langle f(\mathbb{x})\rangle^2} \tag{A4.7}$$

$$q = \frac{\langle \mathbb{y}\rangle \left\langle f^2(\mathbb{x})\right\rangle - \langle \mathbb{y} \odot f(\mathbb{x})\rangle \langle f(\mathbb{x})\rangle}{\left\langle f^2(\mathbb{x})\right\rangle - \langle f(\mathbb{x})\rangle^2} \tag{A4.8}$$

A5. Fitting polynomials

Let us suppose that we have an array $\mathbb{y}$ of N experimental observations, as a function of the array of independent variables $\mathbb{x}$. Then, assume that there exists a model function $m(\mathbb{x}|\mathbb{c})$ that depends on a set of n parameters, $\mathbb{c}$. The set of parameters that better describes the experimental data $\mathbb{y}$ in the least-squares sense, $\hat{\mathbb{c}}$, is the one that minimizes the target function φ, which is the squared sum of the residuals:

$$\hat{\mathbb{c}} = \min_{\mathbb{c}} \varphi(\mathbb{x}|\mathbb{c}) = \min_{\mathbb{c}} \sum_{k=0}^{N-1} \left[\mathbb{y}[k] - m(\mathbb{x}[k]|\mathbb{c})\right]^2 \tag{A5.1}$$

When the model function m is linear in the parameters, there exist a matrix representation of the problem, and thus a closed-form solution. This is indeed the case of fitting the data with a polynomion of rank r: the model function can be expressed as

$$p(\mathbb{x}|\mathbb{c}) = \sum_{i=0}^{r} \mathbb{c}[i]\mathbb{x}^i \tag{A5.2}$$

where the parameters $\mathbb{c}$ are the $n = r + 1$ coefficients of the polynomion. The set of optimal coefficient $\hat{\mathbb{c}}$ can be found by solving the linear system

$$\mathbb{y} = \mathbb{T}\,\mathbb{c} \tag{A5.3}$$

where $\mathbb{T}$ is the $N \times n$ Vandermonde matrix built on the independent variable $\mathbb{x}$:

$$\mathbb{T} = \begin{pmatrix} 1 & x_1 & x_1^2 & x_1^3 & \dots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & x_2^3 & \dots & x_2^{n-1} \\ 1 & x_3 & x_1^2 & x_3^3 & \dots & x_3^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \\ 1 & x_N & x_N^2 & x_N^3 & \dots & x_N^{n-1} \end{pmatrix} \tag{A5.4}$$

Since the columns of $\mathbb{T}$ are linearly independent, its pseudoinverse $\mathbb{T}^+$ behaves as a true left inverse, hence the optimal set of polynomion coefficient is readily given by:

$$\hat{\mathbb{c}} = \mathbb{T}^+\,\mathbb{y} \tag{A5.5}$$

To prevent computational overflows for large values of $\mathbb{x}$, it is convenient to work on a normalized scale.

A6. Use of the `Voigt_Fit` class in `KLASSEZ`

Listing A6.1: Use of the `Voigt_Fit` class, as an attribute of the `Spectrum_1D` class, to deconvolve a simulated NMR spectrum. The same rationale applies for experimental spectra as well.

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

from klassez import *

s = Spectrum_1D('acqus_1D', isexp=False)

# Setup the processing
#   Apodization
s.procs['wf']['mode'] = 'em'
s.procs['wf']['lb'] = 1
#   Zero-filling
s.procs['zf'] = 2**14

#   Apply processing and do FT
s.process()
# Remove the digital filter
s.pknl()
# Phase correction
s.adjph()


# s.F is a fit.Voigt_Fit object
filename = 'test_1D_fit' # base filename for everything fit-related
# Compute the initial guess
auto = False      # True for peak-picker, False for manual
s.F.iguess(filename=filename, auto=auto)

if 0:  # Do the fit
    s.F.dofit(            # Parameters of the fitting
           u_lim=5,           # movement for chemical shift /ppm
           f_lim=50,          # movement for linewidth /Hz
           k_lim=(0, 3),      # limits for intensity
           vary_phase=True,   # optimize the phase of the peak
           vary_b=True,       # optimize the lineshape (L/G ratio)
           method='leastsq', # optimization method
           itermax=10000,     # max. number of iterations
           fit_tol=1e-10,     # arrest criterion threshold (see lmfit for details)
           filename=filename, # filename for the .fuf file
           )
else:
    # Load an existing .fuf file
    s.F.load_fit(filename=filename)

# Plot the results
s.F.plot(what='result',  # what='iguess' for initial guess
        show_total=True, # Show the total trace or not
        show_res=True,   # Show the residuals
        res_offset=0.1,  # Displacement of the residuals (plots residuals - res_offset)
        labels=None,     # Labels for the peaks
        filename=filename, # Filename for the figures
        ext='png',       # format of the figure
        dpi=300,         # Resolution of the figure
        )
```

```
# Compute histogram of the residuals
s.F.res_histogram(what='result',
            nbins=500,   # Number of bins of the histogram
            density=True, # Normalize them
            f_lims=None, # Limits for x axis
            xlabel='Residuals', # Guess what!
            x_symm=True, # Symmetrize the x-scale
            barcolor='tab:green',   # Color of the bars
            fontsize=20, # Guess what!
            filename=filename, ext='png', dpi=300)

# Convert the tables of numbers in arrays
peaks, total, limits = s.F.get_fit_lines(what='result')
```

Input file acqus_1D

```
B0  16.4              # Magnetic field /T
nuc 13C               # Observed nucleus
o1p 100               # Center of the spectrum
SWp 350               # Sweep width /ppm
TD  1024              # Number of sampled complex points

shifts 180, 40, 25    # Chemical shift /ppm
fwhm   200, 200, 150  # FWHM /Hz
amplitudes 10, 15, 50 # Intensity /a.u., only relative values actually matter
x_g 0.2, 0.2, 0.2     # Fraction of gaussianity
phases -5, 0, 5       # Phase angles /degrees
```

Initial guess test_1D_fit.ivf

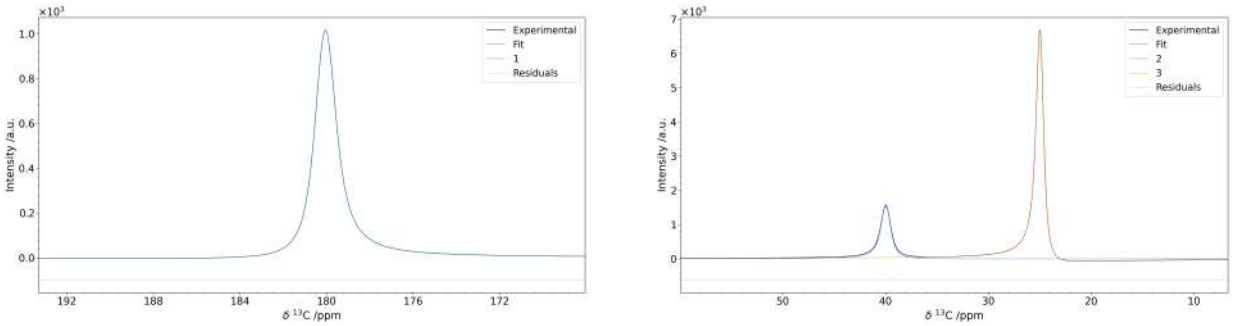
```
! Initial guess computed by francesco on 11/11/2024 at 15:48:44

         Region;     Intensity
-----------------------------------------------------------------------------------------------
 193.317:168.041; 8.08246575e+00

   #;             u;          fwhm;    Rel. I.;     Phase;      Beta;     Group
-----------------------------------------------------------------------------------------------
   1;   179.94060191;    172.500000;  1.000000;   -10.000;   0.00000;        0
-----------------------------------------------------------------------------------------------

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

         Region;     Intensity
-----------------------------------------------------------------------------------------------
    59.936:6.662;  5.02908980e+01

   #;             u;          fwhm;    Rel. I.;     Phase;      Beta;     Group
-----------------------------------------------------------------------------------------------
   2;    40.29851786;    150.000000;  0.214286;     0.000;   0.00000;        0
   3;    24.98695246;    140.000000;  0.785714;    10.000;   0.00000;        0
-----------------------------------------------------------------------------------------------

===============================================================================================
```

Output file test_1D_fit.fvf

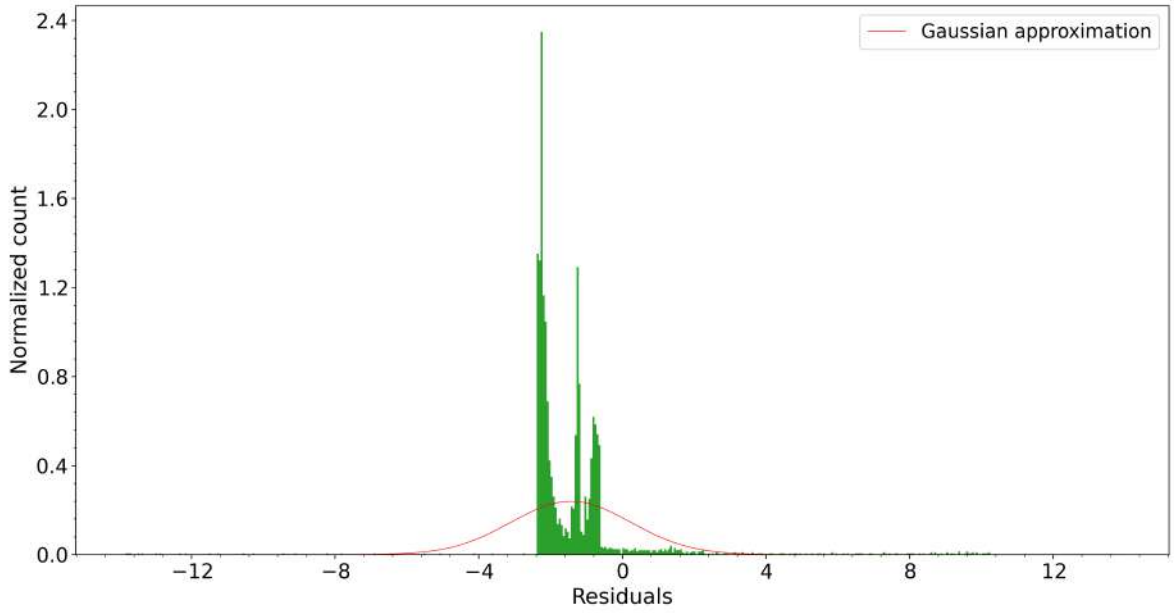
```
! Fit performed by francesco on 11/11/2024 at 16:11:26
```

```
        Region;     Intensity
-------------------------------------------------------------------------------------------------
 193.317:168.041; 1.00107776e+01

   #;            u;          fwhm;   Rel. I.;     Phase;      Beta;     Group
-------------------------------------------------------------------------------------------------
   1;   180.00035537;    204.170109; 1.000000;    -5.397;   0.19035;         0
-------------------------------------------------------------------------------------------------

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

        Region;     Intensity
-------------------------------------------------------------------------------------------------
    59.936:6.662; 6.48017235e+01

   #;            u;          fwhm;   Rel. I.;     Phase;      Beta;     Group
-------------------------------------------------------------------------------------------------
   2;    39.99971634;    200.000000; 0.229725;     4.673;   0.17705;         0
   3;    24.99990717;    154.179890; 0.770275;    10.193;   0.19326;         0
-------------------------------------------------------------------------------------------------

=================================================================================================
```

Figures



(a) **(b)**



(c)

Figure A6.1: Results of the fit of the simulated data (**a** and **b**), and histogram of the residuals (**c**).

A7. Files applications pyIHM

A7.1 Models

A7.1.1 bzac.fvf

```
! Fit performed by francesco on 01/11/2024 at 09:48:52

        Region;     Intensity
-------------------------------------------------------------------------------------------------
  15.994:-1.995; 5.00000000e+00

  #;            u;          fwhm;   Rel. I.;     Phase;      Beta;     Group
-------------------------------------------------------------------------------------------------
  1;     7.94400000;     1.136000;  0.050860;     0.000;   0.99900;         1
  2;     7.94700000;     1.641000;  0.011996;     0.000;   0.25500;         1
  3;     7.94100000;     1.070000;  0.089541;     0.000;   0.00000;         1
  4;     7.93900000;     1.946000;  0.042168;     0.000;   0.95300;         1
  5;     7.93100000;     1.067000;  0.023460;     0.000;   0.00400;         1
  6;     7.92700000;     1.544000;  0.108697;     0.000;   0.78100;         1
  7;     7.92500000;     0.729000;  0.034376;     0.000;   1.00000;         1
  8;     7.92300000;     1.772000;  0.038904;     0.000;   0.49300;         1
  9;     7.62600000;     0.938000;  0.009230;     0.000;   0.00000;         2
 10;     7.62400000;     0.938000;  0.021538;     0.000;   0.00000;         2
 11;     7.62100000;     0.938000;  0.010000;     0.000;   0.00000;         2
 12;     7.61200000;     0.625000;  0.008462;     0.000;   0.00000;         2
 13;     7.61000000;     0.625000;  0.013076;     0.000;   0.00000;         2
 14;     7.60900000;     0.938000;  0.050770;     0.000;   0.00000;         2
 15;     7.60700000;     0.938000;  0.016924;     0.000;   0.00000;         2
 16;     7.60500000;     0.625000;  0.009230;     0.000;   0.00000;         2
 17;     7.59600000;     0.938000;  0.018462;     0.000;   0.00000;         2
 18;     7.59400000;     0.625000;  0.023076;     0.000;   0.00000;         2
 19;     7.59100000;     0.938000;  0.019230;     0.000;   0.00000;         2
 20;     7.50600000;     0.625000;  0.005692;     0.000;   0.00000;         3
 21;     7.50400000;     0.625000;  0.025148;     0.000;   0.00000;         3
 22;     7.50300000;     0.625000;  0.031792;     0.000;   0.00000;         3
 23;     7.50200000;     0.625000;  0.024200;     0.000;   0.00000;         3
 24;     7.49900000;     1.250000;  0.032740;     0.000;   0.00000;         3
 25;     7.49000000;     0.625000;  0.005220;     0.000;   0.00000;         3
 26;     7.48900000;     1.250000;  0.041280;     0.000;   0.00000;         3
 27;     7.48700000;     1.250000;  0.125741;     0.000;   0.00000;         3
 28;     7.48400000;     1.250000;  0.024672;     0.000;   0.00000;         3
 29;     7.47500000;     1.250000;  0.018980;     0.000;   0.00000;         3
 30;     7.47300000;     0.625000;  0.013284;     0.000;   0.00000;         3
 31;     7.47200000;     0.625000;  0.025624;     0.000;   0.00000;         3
 32;     7.47100000;     0.625000;  0.018980;     0.000;   0.00000;         3
 33;     7.46900000;     0.625000;  0.006644;     0.000;   0.00000;         3
-------------------------------------------------------------------------------------------------

=================================================================================================
```

A7.1.2 dmtp.fvf

```
! Fit performed by francesco on 03/11/2024 at 21:08:01
```

```
        Region;     Intensity
-------------------------------------------------------------------------------------------------
   14.014:-2.015; 1.00000000e+01

   #;            u;          fwhm;   Rel. I.;     Phase;      Beta;     Group
-------------------------------------------------------------------------------------------------
   1;     8.07200000;     1.101000;  0.408226;     0.000;   0.22700;         1
   2;     8.07000000;    11.429000;  0.006033;     0.000;   0.53200;         1
   3;     3.87800000;     1.399000;  0.585741;     0.000;   0.66600;         0
-------------------------------------------------------------------------------------------------

=================================================================================================
```

A7.1.3 dmso.fvf

```
! Fit performed by francesco on 03/11/2024 at 21:08:01

        Region;     Intensity
-------------------------------------------------------------------------------------------------
   14.014:-2.015; 6.00000000e+00

   #;            u;          fwhm;   Rel. I.;     Phase;      Beta;     Group
-------------------------------------------------------------------------------------------------
   1;     2.49600000;     0.880000;  0.032030;     0.000;   1.00000;         2
   2;     2.49200000;     1.822000;  0.360630;     0.000;   0.00000;         2
   3;     2.48800000;     1.386000;  0.238270;     0.000;   0.61200;         2
   4;     2.48500000;     1.708000;  0.222080;     0.000;   0.47600;         2
   5;     2.48100000;     1.618000;  0.146990;     0.000;   0.00000;         2
-------------------------------------------------------------------------------------------------

=================================================================================================
```

A7.1.4 EC.fvf

```
! Fit performed by francesco on 01/11/2024 at 09:48:52

        Region;     Intensity
-------------------------------------------------------------------------------------------------
   15.994:-1.995; 4.00000000e+00

   #;            u;          fwhm;   Rel. I.;     Phase;      Beta;     Group
-------------------------------------------------------------------------------------------------
   1;     4.46511811;     1.125000;  1.000000;     0.000;   0.38100;         0
-------------------------------------------------------------------------------------------------

=================================================================================================
```

A7.1.5 TCNB.fvf

```
! Fit performed by francesco on 01/11/2024 at 10:18:29

        Region;     Intensity
-------------------------------------------------------------------------------------------------
   14.014:-2.014; 1.00000000e+00

   #;            u;          fwhm;   Rel. I.;     Phase;      Beta;     Group
-------------------------------------------------------------------------------------------------
   1;     7.74800000;     0.913000;  1.000000;     0.000;   0.32400;         0
-------------------------------------------------------------------------------------------------

=================================================================================================
```

A7.1.6 ochratoxin-a.fvf

```
! Fit performed by francesco on 01/11/2024 at 10:18:29

        Region;     Intensity
-----------------------------------------------------------------------------------------------
   14.014:-2.014; 1.70000000e+01

   #;            u;          fwhm;   Rel. I.;     Phase;      Beta;      Group
-----------------------------------------------------------------------------------------------
   1;     4.73000000;      2.000000;  0.001929;     0.000;   0.00000;         1
   2;     4.73700000;      2.000000;  0.002894;     0.000;   0.00000;         1
   3;     4.74300000;      2.000000;  0.008040;     0.000;   0.00000;         1
   4;     4.74900000;      2.000000;  0.008040;     0.000;   0.00000;         1
   5;     4.75500000;      2.000000;  0.008040;     0.000;   0.00000;         1
   6;     4.76300000;      2.000000;  0.008040;     0.000;   0.00000;         1
   7;     4.76600000;      2.000000;  0.008681;     0.000;   0.00000;         1
   8;     4.77300000;      2.000000;  0.008681;     0.000;   0.00000;         1
   9;     4.77900000;      2.000000;  0.008681;     0.000;   0.00000;         1
  10;     4.78600000;      2.000000;  0.008681;     0.000;   0.00000;         1
  11;     4.79200000;      2.000000;  0.002253;     0.000;   0.00000;         1
  12;     4.79900000;      2.000000;  0.002253;     0.000;   0.00000;         1
  13;     8.43000000;      2.000000;  0.072003;     0.000;   0.00000;         0
  14;    12.69000000;      2.000000;  0.083586;     0.000;   0.00000;         0
  15;     8.49500000;      2.000000;  0.016703;     0.000;   0.00000;         4
  16;     8.50500000;      2.000000;  0.015739;     0.000;   0.00000;         4
  17;     5.03500000;      2.000000;  0.009005;     0.000;   0.00000;         5
  18;     5.04800000;      2.000000;  0.020579;     0.000;   0.00000;         5
  19;     5.04400000;      2.000000;  0.009005;     0.000;   0.00000;         5
  20;     5.05800000;      2.000000;  0.020579;     0.000;   0.00000;         5
  21;     5.06200000;      2.000000;  0.009005;     0.000;   0.00000;         5
  22;     5.07400000;      2.000000;  0.009005;     0.000;   0.00000;         5
  23;     1.59500000;      2.000000;  0.100289;     0.000;   0.00000;         8
  24;     1.61000000;      2.000000;  0.092573;     0.000;   0.00000;         8
  25;     7.33300000;      0.813000;  0.001350;     0.000;   0.00000;        11
  26;     7.33000000;      1.725000;  0.010530;     0.000;   0.00000;        11
  27;     7.33600000;      9.399000;  0.000000;     0.000;   0.00000;        11
  28;     7.32700000;      0.790000;  0.001499;     0.000;   0.00000;        11
  29;     7.32000000;      1.331000;  0.006023;     0.000;   0.00000;        11
  30;     7.31400000;      1.360000;  0.014966;     0.000;   0.00000;        11
  31;     7.30500000;      1.041000;  0.005420;     0.000;   0.00000;        11
  32;     7.30200000;      1.224000;  0.021567;     0.000;   0.00000;        11
  33;     7.29874775;      1.053000;  0.004201;     0.000;   0.00000;        11
  34;     7.29424326;      0.824000;  0.000535;     0.000;   0.00000;        11
  35;     7.28200000;      0.763000;  0.001789;     0.000;   0.00000;        12
  36;     7.27900000;      1.373000;  0.009902;     0.000;   0.00000;        12
  37;     7.27600000;      1.262000;  0.008762;     0.000;   0.00000;        12
  38;     7.27200000;      0.703000;  0.015146;     0.000;   0.00000;        12
  39;     7.26500000;      1.261000;  0.014893;     0.000;   0.00000;        12
  40;     7.26000000;      1.527000;  0.033653;     0.000;   0.00000;        12
  41;     7.25700000;      1.514000;  0.024075;     0.000;   0.00000;        12
  42;     7.25200000;      0.625000;  0.002596;     0.000;   0.00000;        12
  43;     7.24700000;      1.613000;  0.011639;     0.000;   0.00000;        12
  44;     7.24400000;      1.676000;  0.020628;     0.000;   0.00000;        12
  45;     7.24100000;      1.180000;  0.007560;     0.000;   0.00000;        12
  46;     3.38200000;      1.694000;  0.009250;     0.000;   0.00000;         1
  47;     3.37100000;      1.633000;  0.009233;     0.000;   0.00000;         1
  48;     3.35300000;      1.712000;  0.016081;     0.000;   0.00000;         1
  49;     3.34300000;      1.816000;  0.016055;     0.000;   0.00000;         1
  50;     3.31400000;      1.112000;  0.009917;     0.000;   0.00000;         1
  51;     3.30700000;      1.133000;  0.010101;     0.000;   0.00000;         1
  52;     3.27900000;      1.130000;  0.014915;     0.000;   0.00000;         1
  53;     3.27200000;      1.175000;  0.015020;     0.000;   0.00000;         1
  54;     3.25000000;      1.663000;  0.016107;     0.000;   0.00000;         1
  55;     3.23600000;      1.630000;  0.016037;     0.000;   0.00000;         1
  56;     3.22200000;      1.562000;  0.009329;     0.000;   0.00000;         1
  57;     3.20800000;      1.635000;  0.009452;     0.000;   0.00000;         1
  58;     2.89400000;      1.350000;  0.022542;     0.000;   0.00000;         3
  59;     2.87100000;      1.356000;  0.021579;     0.000;   0.00000;         3
  60;     2.85900000;      1.364000;  0.019018;     0.000;   0.00000;         3
  61;     2.83600000;      1.339000;  0.018869;     0.000;   0.00000;         3
  62;     7.31639262;      0.937500;  0.015474;     0.000;   0.00000;         0
-----------------------------------------------------------------------------------------------
```

```
================================================================================================
```

A7.1.7 formic_acid.fvf

```
! Fit performed by francesco on 31/10/2024 at 15:05:10

         Region;     Intensity
------------------------------------------------------------------------------------------------
   14.709:-5.321; 1.00000000e+00

   #;             u;          fwhm;   Rel. I.;     Phase;      Beta;     Group
------------------------------------------------------------------------------------------------
   1;     8.38783386;     1.000000;  1.000000;     0.000;   0.00000;         0
------------------------------------------------------------------------------------------------

================================================================================================
```

A7.1.8 acetic_acid.fvf

```
! Fit performed by francesco on 31/10/2024 at 15:05:10

         Region;     Intensity
------------------------------------------------------------------------------------------------
   14.709:-5.321; 3.00000000e+00

   #;             u;          fwhm;   Rel. I.;     Phase;      Beta;     Group
------------------------------------------------------------------------------------------------
   1;     1.85120657;     1.000000;  1.000000;     0.000;   0.00000;         0
------------------------------------------------------------------------------------------------

================================================================================================
```

A7.1.9 citrate.fvf

```
! Fit performed by francesco on 31/10/2024 at 15:05:10

         Region;     Intensity
------------------------------------------------------------------------------------------------
   14.709:-5.321; 4.00000000e+00

   #;             u;          fwhm;   Rel. I.;     Phase;      Beta;     Group
------------------------------------------------------------------------------------------------
   1;     2.57900000;     1.000000;  0.309570;     0.000;   0.00000;         1
   2;     2.60400000;     1.000000;  0.193250;     0.000;   0.00000;         1
   3;     2.45600000;     1.000000;  0.184480;     0.000;   0.00000;         2
   4;     2.48000000;     1.000000;  0.312700;     0.000;   0.00000;         2
------------------------------------------------------------------------------------------------

================================================================================================
```

A7.1.10 lactate.fvf

```
! Fit performed by francesco on 31/10/2024 at 15:05:10

         Region;     Intensity
------------------------------------------------------------------------------------------------
   14.709:-5.321; 4.00000000e+00

   #;             u;          fwhm;   Rel. I.;     Phase;      Beta;     Group
------------------------------------------------------------------------------------------------
```

```
   1;     1.25609397;      0.687500;  0.383333;     0.000;    0.00000;         1
   2;     1.26753245;      0.531250;  0.283333;     0.000;    0.00000;         1
   3;     4.03100423;      1.000000;  0.041667;     0.000;    0.00000;         2
   4;     4.04258888;      1.000000;  0.125000;     0.000;    0.00000;         2
   5;     4.05421839;      1.000000;  0.125000;     0.000;    0.00000;         2
   6;     4.06580304;      1.000000;  0.041667;     0.000;    0.00000;         2
-----------------------------------------------------------------------------------------------

===============================================================================================
```

A7.1.11 alanine.fvf

```
! Fit performed by francesco on 31/10/2024 at 15:05:10

         Region;     Intensity
-----------------------------------------------------------------------------------------------
   14.709:-5.321; 4.00000000e+00

   #;             u;           fwhm;    Rel. I.;     Phase;      Beta;      Group
-----------------------------------------------------------------------------------------------
   1;     3.69831654;      1.000000;  0.011848;     0.000;    0.00000;         1
   2;     3.70731654;      1.000000;  0.035545;     0.000;    0.00000;         1
   3;     3.71531654;      1.000000;  0.035545;     0.000;    0.00000;         1
   4;     3.72431654;      1.000000;  0.011848;     0.000;    0.00000;         1
   5;     1.40597935;      1.000000;  0.464455;     0.000;    0.00000;         2
   6;     1.41799091;      1.000000;  0.440758;     0.000;    0.00000;         2
-----------------------------------------------------------------------------------------------

===============================================================================================
```

A7.1.12 glutamine.fvf

```
! Fit performed by francesco on 31/10/2024 at 15:05:10

         Region;     Intensity
-----------------------------------------------------------------------------------------------
   14.709:-5.321; 9.00000000e+00

   #;             u;           fwhm;    Rel. I.;     Phase;      Beta;      Group
-----------------------------------------------------------------------------------------------
   1;     2.02000000;      1.000000;  0.001918;     0.000;    0.00000;         1
   2;     2.03100000;      1.000000;  0.001918;     0.000;    0.00000;         1
   3;     2.03300000;      1.000000;  0.001911;     0.000;    0.00000;         1
   4;     2.03500000;      1.000000;  0.001911;     0.000;    0.00000;         1
   5;     2.04500000;      1.000000;  0.013397;     0.000;    0.00000;         1
   6;     2.05300000;      1.000000;  0.013397;     0.000;    0.00000;         1
   7;     2.34400000;      1.000000;  0.004225;     0.000;    0.00000;         2
   8;     2.36900000;      1.000000;  0.029374;     0.000;    0.00000;         2
   9;     2.35600000;      1.000000;  0.004841;     0.000;    0.00000;         2
  10;     2.35800000;      1.000000;  0.003616;     0.000;    0.00000;         2
  11;     2.38000000;      1.000000;  0.025075;     0.000;    0.00000;         2
  12;     2.38200000;      1.000000;  0.024548;     0.000;    0.00000;         2
  13;     2.38300000;      1.000000;  0.022699;     0.000;    0.00000;         3
  14;     2.39200000;      1.000000;  0.025684;     0.000;    0.00000;         3
  15;     2.41700000;      1.000000;  0.004841;     0.000;    0.00000;         3
  16;     2.42000000;      1.000000;  0.004225;     0.000;    0.00000;         3
  17;     2.40600000;      1.000000;  0.023842;     0.000;    0.00000;         3
  18;     2.43200000;      1.000000;  0.003616;     0.000;    0.00000;         3
  19;     3.69600000;      1.000000;  0.028055;     0.000;    0.00000;         4
  20;     3.70700000;      1.312000;  0.051724;     0.000;    0.00000;         4
  21;     3.71700000;      1.250000;  0.031659;     0.000;    0.00000;         4
  22;     6.81100000;      5.000000;  0.222165;     0.000;    0.00000;         0
  23;     7.54100000;      5.000000;  0.222165;     0.000;    0.00000;         0
  24;     2.05600000;      1.172000;  0.022973;     0.000;    0.00000;         0
  25;     2.06000000;      1.172000;  0.017225;     0.000;    0.00000;         0
  26;     2.06300000;      0.625000;  0.011479;     0.000;    0.00000;         0
  27;     2.06400000;      0.625000;  0.009576;     0.000;    0.00000;         0
```

```
  28;     2.06700000;      0.938000;  0.022973;     0.000;    0.00000;         0
  29;     2.07100000;      1.016000;  0.024883;     0.000;    0.00000;         0
  30;     2.07400000;      0.938000;  0.015315;     0.000;    0.00000;         0
  31;     2.07700000;      0.625000;  0.009576;     0.000;    0.00000;         0
  32;     2.07900000;      0.781000;  0.011479;     0.000;    0.00000;         0
  33;     2.08200000;      0.547000;  0.007658;     0.000;    0.00000;         0
  34;     2.08900000;      1.797000;  0.026801;     0.000;    0.00000;         0
  35;     2.10000000;      1.094000;  0.001911;     0.000;    0.00000;         0
  36;     2.10000000;      1.562000;  0.001911;     0.000;    0.00000;         0
  37;     2.10300000;      0.781000;  0.001911;     0.000;    0.00000;         0
  38;     2.11300000;      0.938000;  0.001911;     0.000;    0.00000;         0
  39;     2.39400000;      0.781000;  0.015392;     0.000;    0.00000;         0
  40;     2.39500000;      0.938000;  0.024006;     0.000;    0.00000;         0
  41;     2.38800000;      1.875000;  0.006215;     0.000;    0.00000;         0
-------------------------------------------------------------------------------------------------

=================================================================================================
```

A7.1.13 creatinine.fvf

```
! Fit performed by francesco on 31/10/2024 at 15:05:10

         Region;     Intensity
-------------------------------------------------------------------------------------------------
   14.709:-5.321; 5.00000000e+00

   #;             u;           fwhm;    Rel. I.;     Phase;      Beta;      Group
-------------------------------------------------------------------------------------------------
   1;     2.97804068;      1.000000;  0.600000;     0.000;    0.00000;         0
   2;     3.98804068;      1.000000;  0.400000;     0.000;    0.00000;         0
-------------------------------------------------------------------------------------------------

=================================================================================================
```

A7.1.14 a-glucose_exp.fvf

```
! Fit performed by francesco on 31/10/2024 at 15:05:10

         Region;     Intensity
-------------------------------------------------------------------------------------------------
   14.709:-5.321; 7.00000000e+00

   #;             u;           fwhm;    Rel. I.;     Phase;      Beta;      Group
-------------------------------------------------------------------------------------------------
   1;     5.16656714;      1.000000;  0.055911;     0.000;    0.00000;         1
   2;     5.17285368;      1.000000;  0.050829;     0.000;    0.00000;         1
   3;     3.46037612;      1.000000;  0.037022;     0.000;    0.00000;         2
   4;     3.47650745;      1.000000;  0.037022;     0.000;    0.00000;         2
   5;     3.46637612;      1.000000;  0.037022;     0.000;    0.00000;         2
   6;     3.48277011;      1.000000;  0.037022;     0.000;    0.00000;         2
   7;     3.63384790;      1.000000;  0.045001;     0.000;    0.00000;         3
   8;     3.64922996;      1.000000;  0.045001;     0.000;    0.00000;         3
   9;     3.65505049;      1.000000;  0.045001;     0.000;    0.00000;         3
  10;     3.66622996;      1.000000;  0.045001;     0.000;    0.00000;         3
  11;     3.33237612;      1.000000;  0.020893;     0.000;    0.00000;         4
  12;     3.34896123;      1.000000;  0.020893;     0.000;    0.00000;         4
  13;     3.34796711;      1.000000;  0.020893;     0.000;    0.00000;         4
  14;     3.36437612;      1.000000;  0.022983;     0.000;    0.00000;         4
  15;     3.75002158;      1.000000;  0.016098;     0.000;    0.00000;         5
  16;     3.75835299;      1.000000;  0.006441;     0.000;    0.00000;         5
  17;     3.75373504;      1.000000;  0.016098;     0.000;    0.00000;         5
  18;     3.76693763;      1.000000;  0.019317;     0.000;    0.00000;         5
  19;     3.76902736;      1.000000;  0.019317;     0.000;    0.00000;         5
  20;     3.77649915;      1.000000;  0.016098;     0.000;    0.00000;         5
  21;     3.77188120;      1.000000;  0.016098;     0.000;    0.00000;         5
  22;     3.78011709;      1.000000;  0.003222;     0.000;    0.00000;         5
  23;     3.78828639;      1.000000;  0.019311;     0.000;    0.00000;         6
```

```
  24;     3.76217353;      1.000000;  0.016093;     0.000;    0.00000;         6
  25;     3.79228639;      1.000000;  0.016093;     0.000;    0.00000;         6
  26;     3.77047163;      1.000000;  0.032187;     0.000;    0.00000;         6
  27;     3.72662497;      0.625000;  0.100531;     0.000;    0.00000;         7
  28;     3.71441221;      0.687500;  0.092598;     0.000;    0.00000;         7
  29;     3.73868140;      1.000000;  0.045001;     0.000;    0.00000;         7
  30;     3.70239925;      1.000000;  0.045001;     0.000;    0.00000;         7
-----------------------------------------------------------------------------------------------

===============================================================================================
```

A7.1.15 b-glucose_exp.fvf

```
! Fit performed by francesco on 31/10/2024 at 15:05:10

         Region;     Intensity
-----------------------------------------------------------------------------------------------
   14.709:-5.321; 7.00000000e+00

   #;             u;           fwhm;    Rel. I.;     Phase;      Beta;     Group
-----------------------------------------------------------------------------------------------
   1;     4.57354226;      1.000000;  0.089455;     0.000;    0.00000;         1
   2;     4.58754226;      1.000000;  0.089455;     0.000;    0.00000;         1
   3;     3.16439610;      1.000000;  0.044721;     0.000;    0.00000;         2
   4;     3.18016021;      1.000000;  0.044721;     0.000;    0.00000;         2
   5;     3.17787366;      1.000000;  0.044721;     0.000;    0.00000;         2
   6;     3.19363777;      1.000000;  0.044721;     0.000;    0.00000;         2
   7;     3.41059869;      1.000000;  0.037616;     0.000;    0.00000;         3
   8;     3.42559869;      1.000000;  0.036049;     0.000;    0.00000;         3
   9;     3.42578972;      1.000000;  0.034482;     0.000;    0.00000;         3
  10;     3.44117177;      1.000000;  0.031347;     0.000;    0.00000;         3
  11;     3.32254226;      1.000000;  0.023510;     0.000;    0.00000;         4
  12;     3.33916021;      1.000000;  0.031347;     0.000;    0.00000;         4
  13;     3.33811534;      1.000000;  0.031347;     0.000;    0.00000;         4
  14;     3.35435123;      1.000000;  0.054858;     0.000;    0.00000;         4
  15;     3.38554226;      1.000000;  0.020380;     0.000;    0.00000;         5
  16;     3.39554226;      1.000000;  0.021947;     0.000;    0.00000;         5
  17;     3.38954226;      1.000000;  0.023515;     0.000;    0.00000;         5
  18;     3.39896918;      1.000000;  0.020380;     0.000;    0.00000;         5
  19;     3.40192431;      1.000000;  0.009408;     0.000;    0.00000;         5
  20;     3.41135123;      1.000000;  0.017245;     0.000;    0.00000;         5
  21;     3.40554226;      1.000000;  0.012543;     0.000;    0.00000;         5
  22;     3.41554226;      1.000000;  0.012543;     0.000;    0.00000;         5
  23;     3.84436280;      1.000000;  0.026644;     0.000;    0.00000;         6
  24;     3.82398074;      1.000000;  0.031347;     0.000;    0.00000;         6
  25;     3.84798074;      1.000000;  0.025077;     0.000;    0.00000;         6
  26;     3.82740766;      1.000000;  0.031347;     0.000;    0.00000;         6
  27;     3.66598074;      1.000000;  0.030281;     0.000;    0.00000;         7
  28;     3.64598074;      1.000000;  0.026331;     0.000;    0.00000;         7
  29;     3.67598074;      1.000000;  0.026331;     0.000;    0.00000;         7
  30;     3.65559869;      1.000000;  0.026331;     0.000;    0.00000;         7
-----------------------------------------------------------------------------------------------

===============================================================================================
```

A7.1.16 urea.fvf

```
! Fit performed by francesco on 31/10/2024 at 15:05:10

         Region;     Intensity
-----------------------------------------------------------------------------------------------
   14.709:-5.321; 4.00000000e+00

   #;             u;           fwhm;    Rel. I.;     Phase;      Beta;     Group
-----------------------------------------------------------------------------------------------
   1;     5.70865197;     35.000000;  1.000000;     0.000;    0.00000;         0
-----------------------------------------------------------------------------------------------
```

```
================================================================================================
```

A7.1.17 DSS.fvf

```
! Fit performed by francesco on 31/10/2024 at 15:05:10

         Region;     Intensity
------------------------------------------------------------------------------------------------
   14.709:-5.321; 1.50000000e+01

   #;             u;          fwhm;   Rel. I.;     Phase;      Beta;     Group
------------------------------------------------------------------------------------------------
   1;   - 0.05609082;     0.938000;  0.478011;     0.000;   0.00000;         0
   2;     0.58790918;     1.925000;  0.038933;     0.000;   0.49500;         0
   3;     0.57790918;     2.622000;  0.020057;     0.000;   0.02200;         0
   4;     0.57090918;     2.519000;  0.049603;     0.000;   0.01500;         0
   5;     0.56490918;     2.515000;  0.019839;     0.000;   0.01400;         0
   6;     0.55390918;     1.938000;  0.038227;     0.000;   0.43600;         0
   7;     1.73390918;     1.748000;  0.013566;     0.000;   0.00000;         0
   8;     1.72390918;     1.902000;  0.009055;     0.000;   0.44100;         0
   9;     1.71790918;     1.942000;  0.023002;     0.000;   0.00100;         0
  10;     1.70990918;     2.745000;  0.027953;     0.000;   0.00000;         0
  11;     1.70090918;     2.191000;  0.030063;     0.000;   1.00000;         0
  12;     1.69190918;     3.006000;  0.030623;     0.000;   0.00000;         0
  13;     1.68490918;     2.123000;  0.023800;     0.000;   0.00000;         0
  14;     1.67890918;     1.779000;  0.008393;     0.000;   0.57000;         0
  15;     1.66790918;     1.916000;  0.014904;     0.000;   0.00000;         0
  16;     2.86790918;     1.825000;  0.040340;     0.000;   0.49500;         0
  17;     2.85790918;     2.360000;  0.009774;     0.000;   0.47500;         0
  18;     2.85290918;     2.219000;  0.069229;     0.000;   0.01000;         0
  19;     2.84590918;     1.824000;  0.008281;     0.000;   0.98600;         0
  20;     2.83690918;     1.857000;  0.046346;     0.000;   0.37700;         0
------------------------------------------------------------------------------------------------

================================================================================================
```

A7.2 BzAc – DMTP – DMSO

A7.2.1 Input file

```
BASE_FILENAME
new2

MIX_PATH
DMTP_BCBL1702V/DMTP-1_20201111_01/DMTP-1_PROTON_nt1_28C_01.fid, spect='varian'

COMP_PATH
../comp/bzac_auto.fvf, 5H
../comp/dmso.fvf, 6H
../comp/dmtp.fvf, 10H

FIT_KWS
method='leastsq', tol=1e-4, max_nfev=2000

FIT_BDS
utol=0.1
utol_sg=0.01
stol=10
ktol=0.01

PLT_OPTS
ext=png, dpi=600

FIT_LIMITS
```

```
  8.095,  8.043
  7.965,  7.907
  7.638,  7.577
  7.525,  7.452
  3.884,  3.870
  2.509,  2.467

I0
1.8500, 0.6500, 0.7000
```

A7.2.2 Output file

```
! Fit performed by francesco on 30/10/2024 at 18:00:50

Mixture spectrum: /home/francesco/Programma/_on_dev/test_pyihm/4Ltz_articolo/aligned/DMTP_BCBL1702V/DMTP-1
     _20201111_01/DMTP-1_PROTON_nt1_28C_01.fid/DMTP-1_PROTON_nt1_28C_01

Absolute intensity correction: I = 3.64326e+01

Relative intensities:
Component  1:   59.81807% | Rel:  3.01798
Component  2:   19.82056% | Rel:  1.00000
Component  3:   20.36136% | Rel:  1.02728



Component 1 fitted parameters:
         Region;     Intensity
-------------------------------------------------------------------------------------------------
     8.095:2.467;  1.08966393e+02

   #;            u;          fwhm;   Rel. I.;     Phase;      Beta;     Group
-------------------------------------------------------------------------------------------------
   1;     7.94168302;     1.414687;  0.060252;     0.000;   0.36450;         0
   2;     7.94419562;     0.991711;  0.057234;     0.000;   0.20791;         0
   3;     7.92484364;     0.932616;  0.064793;     0.000;   0.20357;         0
   4;     7.92895327;     0.714024;  0.011120;     0.000;   0.25050;         0
   5;     7.94763481;     1.030958;  0.004476;     0.000;   0.53138;         0
   6;     7.93964969;     2.051173;  0.047743;     0.000;   0.81999;         0
   7;     7.93146720;     1.133605;  0.020318;     0.000;   0.05195;         0
   8;     7.94309427;     0.522211;  0.006470;     0.000;   0.74482;         0
   9;     7.92761209;     1.205186;  0.094559;     0.000;   0.43735;         0
  10;     7.92330654;     1.427579;  0.018338;     0.000;   0.42770;         0
  11;     7.93805555;     0.573401;  0.002915;     0.000;   0.02492;         0
  12;     7.94167423;     0.613084;  0.012282;     0.000;   0.29368;         0
  13;     7.60729528;     0.833030;  0.008924;     0.000;   0.69884;         0
  14;     7.59149878;     0.954468;  0.014836;     0.000;   0.16845;         0
  15;     7.62649443;     0.945538;  0.007985;     0.000;   0.36096;         0
  16;     7.61069291;     0.839439;  0.012269;     0.000;   0.21950;         0
  17;     7.59420479;     1.013937;  0.036460;     0.000;   0.00131;         0
  18;     7.62122763;     1.011896;  0.011514;     0.000;   0.01446;         0
  19;     7.60553139;     1.065489;  0.017334;     0.000;   0.10576;         0
  20;     7.60901807;     1.003339;  0.047352;     0.000;   0.28527;         0
  21;     7.59687913;     0.902399;  0.012440;     0.000;   0.42546;         0
  22;     7.61261700;     1.002279;  0.011695;     0.000;   0.15079;         0
  23;     7.62387169;     0.977521;  0.019922;     0.000;   0.13788;         0
  24;     7.50187055;     1.055682;  0.030177;     0.000;   0.26349;         0
  25;     7.46946312;     1.355409;  0.014438;     0.000;   0.41376;         0
  26;     7.48414808;     1.304971;  0.030676;     0.000;   0.15684;         0
  27;     7.49964800;     1.117415;  0.026061;     0.000;   0.00054;         0
  28;     7.48683234;     1.277303;  0.101110;     0.000;   0.51132;         0
  29;     7.47232347;     0.646004;  0.013115;     0.000;   0.04750;         0
  30;     7.47555858;     0.924991;  0.011958;     0.000;   0.17489;         0
  31;     7.50601067;     0.881318;  0.003845;     0.000;   0.89583;         0
  32;     7.48955633;     1.528225;  0.047340;     0.000;   0.64394;         0
  33;     7.47315771;     1.067108;  0.025586;     0.000;   0.00562;         0
  34;     7.48899813;     0.515440;  0.003662;     0.000;   0.08480;         0
  35;     7.50360698;     0.984296;  0.039870;     0.000;   0.13932;         0
  36;     7.48812467;     0.697059;  0.013401;     0.000;   0.65375;         0
  37;     7.50278261;     0.707630;  0.018067;     0.000;   0.21126;         0
  38;     7.47137737;     0.902802;  0.019467;     0.000;   0.89280;         0
```

```
-------------------------------------------------------------------------------------------------
=================================================================================================


Component 2 fitted parameters:
         Region;     Intensity
-------------------------------------------------------------------------------------------------
     8.095:2.467;  4.33268821e+01

   #;            u;          fwhm;    Rel. I.;     Phase;      Beta;     Group
-------------------------------------------------------------------------------------------------
   1;     2.49597556;      1.330890;  0.070415;     0.000;    0.86814;         0
   2;     2.49219502;      1.526405;  0.292513;     0.000;    0.25421;         0
   3;     2.48851018;      1.379664;  0.287276;     0.000;    0.74763;         0
   4;     2.48492775;      1.417845;  0.214389;     0.000;    0.75464;         0
   5;     2.48128408;      1.530572;  0.135408;     0.000;    0.29247;         0
-------------------------------------------------------------------------------------------------

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


Component 3 fitted parameters:
         Region;     Intensity
-------------------------------------------------------------------------------------------------
     8.095:2.467;  7.41817435e+01

   #;            u;          fwhm;    Rel. I.;     Phase;      Beta;     Group
-------------------------------------------------------------------------------------------------
   1;     3.87857658;      1.086769;  0.587324;     0.000;    0.17156;         0
   2;     8.07263216;      0.944862;  0.407631;     0.000;    0.01770;         0
   3;     8.06065086;     21.429000;  0.005045;     0.000;    0.00005;         0
-------------------------------------------------------------------------------------------------

=================================================================================================
```

A7.2.3 Histogram of the residuals

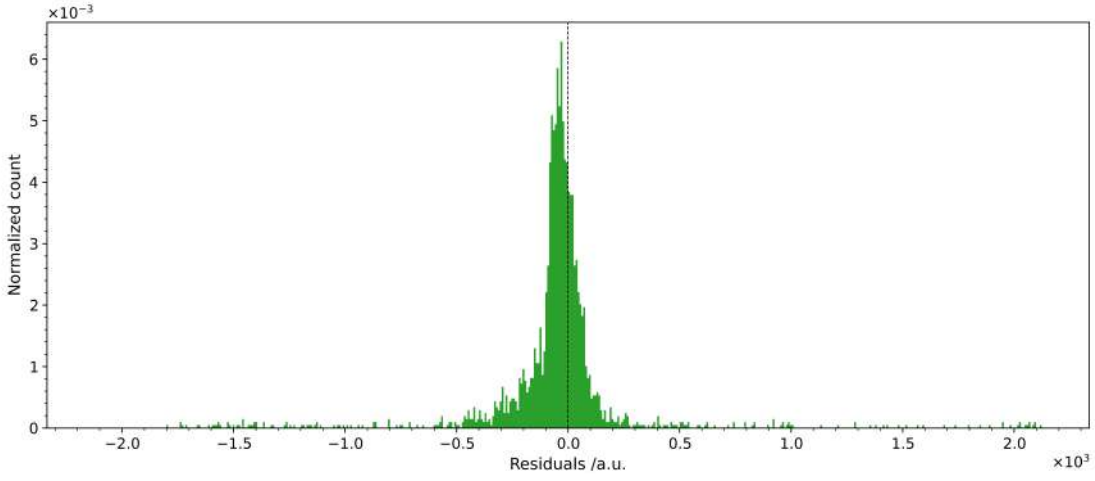


Figure A7.1: Histogram of the residuals of the `pyIHM` deconvolution of the BzAc − DMTP − DMSO mixture.

A7.3 BzAc − EC − DMSO

A7.3.1 Input file

```
BASE_FILENAME
new

MIX_PATH
/media/francesco/Volume/NMR/BAM/NMR_DATA/EC_HPCStd_803959/Ethylencarbonat_803959-1_20220128_01/
     Ethylencarbonat_803959-1_PROTON_nt16_27C_01.fid, spect='varian'

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

FIT_KWS
method='nelder', tol=1e-3, max_nfev=10000
method='leastsq', tol=1e-5, max_nfev=20000

FIT_BDS
utol=0.2
utol_sg=0.01
stol=10
ktol=0.01

PLT_OPTS
ext=png, dpi=300

FIT_LIMITS
  7.998,  7.893
  7.652,  7.575
  7.534,  7.452
  4.512,  4.439
  2.529,  2.469

I0
1.3000, 0.5000, 1.4000
```

A7.3.2 Output file

```
! Fit performed by francesco on 01/11/2024 at 16:44:07

Mixture spectrum: /media/francesco/Volume/NMR/BAM/NMR_DATA/EC_HPCStd_803959/Ethylencarbonat_803959-1_20220128_01/
     Ethylencarbonat_803959-1_PROTON_nt16_27C_01.fid/Ethylencarbonat_803959-1_PROTON_nt16_27C_01

Absolute intensity correction: I = 6.10741e+04

Relative intensities:
Component  1:   49.46903% | Rel:  3.81773
Component  2:   12.95770% | Rel:  1.00000
Component  3:   37.57327% | Rel:  2.89969



Component 1 fitted parameters:
         Region;     Intensity
-------------------------------------------------------------------------------------------------
     7.998:2.469;  1.51063738e+05

   #;            u;          fwhm;   Rel. I.;     Phase;      Beta;     Group
-------------------------------------------------------------------------------------------------
   1;     7.95264790;     1.149675;  0.053033;     0.000;   0.99983;         0
   2;     7.95589002;     1.404773;  0.013228;     0.000;   0.39838;         0
   3;     7.95026076;     1.011163;  0.088632;     0.000;   0.00000;         0
   4;     7.94788480;     1.684384;  0.040301;     0.000;   0.67888;         0
   5;     7.94012805;     1.037207;  0.025339;     0.000;   0.00198;         0
   6;     7.93640373;     1.394226;  0.108041;     0.000;   0.75984;         0
   7;     7.93232780;     1.626981;  0.032482;     0.000;   0.59417;         0
   8;     7.93351110;     0.694046;  0.038373;     0.000;   0.72419;         0
   9;     7.63211317;     0.856136;  0.009666;     0.000;   0.04185;         0
  10;     7.62947065;     0.840685;  0.020339;     0.000;   0.00000;         0
  11;     7.62682061;     0.781624;  0.009494;     0.000;   0.06132;         0
  12;     7.61829374;     0.859920;  0.010496;     0.000;   0.33638;         0
```

```
  13;     7.61624197;      0.812806;  0.013923;     0.000;    0.18593;         0
  14;     7.61460272;      0.917371;  0.051350;     0.000;    0.00000;         0
  15;     7.61266666;      0.960277;  0.014973;     0.000;    0.00000;         0
  16;     7.61098528;      0.651603;  0.008073;     0.000;    0.24468;         0
  17;     7.60240272;      0.949695;  0.020299;     0.000;    0.05121;         0
  18;     7.59981114;      0.814692;  0.025158;     0.000;    0.52972;         0
  19;     7.59720674;      0.910809;  0.017286;     0.000;    0.17771;         0
  20;     7.51228175;      0.905283;  0.007680;     0.000;    0.39989;         0
  21;     7.51027200;      1.182999;  0.024614;     0.000;    0.05025;         0
  22;     7.50944019;      0.882551;  0.030267;     0.000;    0.69751;         0
  23;     7.50829623;      1.041309;  0.024355;     0.000;    0.99916;         0
  24;     7.50595236;      1.083615;  0.030841;     0.000;    0.00025;         0
  25;     7.49719027;      1.092223;  0.007244;     0.000;    0.28551;         0
  26;     7.49563727;      1.461042;  0.043421;     0.000;    0.68896;         0
  27;     7.49333068;      1.492681;  0.124145;     0.000;    0.62212;         0
  28;     7.49026166;      0.997840;  0.022746;     0.000;    0.00002;         0
  29;     7.48170734;      0.985645;  0.017046;     0.000;    0.18403;         0
  30;     7.47934891;      0.813559;  0.014191;     0.000;    0.71916;         0
  31;     7.47828387;      0.694036;  0.026566;     0.000;    0.03555;         0
  32;     7.47721759;      0.778344;  0.020984;     0.000;    0.00946;         0
  33;     7.47536852;      0.635252;  0.005413;     0.000;    0.12764;         0
-----------------------------------------------------------------------------------------------

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


Component 2 fitted parameters:
         Region;     Intensity
-----------------------------------------------------------------------------------------------
     7.998:2.469;  4.74827699e+04

   #;            u;          fwhm;    Rel. I.;     Phase;      Beta;     Group
-----------------------------------------------------------------------------------------------
   1;     2.50309697;      0.858469;  0.033747;     0.000;    1.00000;         0
   2;     2.49927959;      1.663629;  0.362904;     0.000;    0.01160;         0
   3;     2.49556968;      1.256402;  0.237000;     0.000;    0.87646;         0
   4;     2.49206487;      1.351996;  0.220783;     0.000;    0.56352;         0
   5;     2.48844342;      1.599199;  0.145567;     0.000;    0.00028;         0
-----------------------------------------------------------------------------------------------

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


Component 3 fitted parameters:
         Region;     Intensity
-----------------------------------------------------------------------------------------------
     7.998:2.469;  9.17900882e+04

   #;            u;          fwhm;    Rel. I.;     Phase;      Beta;     Group
-----------------------------------------------------------------------------------------------
   1;     4.47484148;      0.726193;  1.000000;     0.000;    0.00044;         0
-----------------------------------------------------------------------------------------------

===============================================================================================
```

A7.3.3 Histogram of the residuals

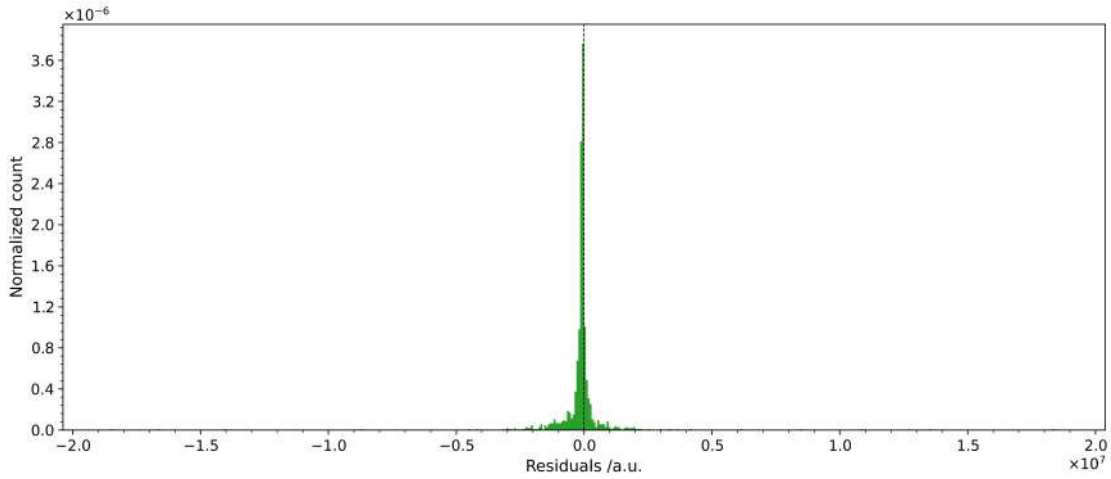


Figure A7.2: Histogram of the residuals of the `pyIHM` deconvolution of the BzAc − EC − DMSO mixture.

A7.4 Purity assessment of Ochratoxin-A

A7.4.1 Input file

```
BASE_FILENAME
ochratox_nuovo

MIX_PATH
../spettri/Ochratoxin_A_OC045_2_PROTON_nt32_27C_01.fid, spect='varian'

COMP_PATH
../comp/ochratoxin-a.fvf, 17H
../comp/TCNB.fvf, 1H

FIT_KWS
method='leastsq', tol=1e-6, max_nfev=5000

FIT_BDS
utol=0.20
utol_sg=0.1
stol=20
ktol=0.01

PLT_OPTS
ext=png, dpi=300
```

A7.4.2 Output file

```
! Fit performed by francesco on 01/11/2024 at 14:24:41

Mixture spectrum: /home/francesco/Programma/_on_dev/test_pyihm/4Ltz_articolo/spettri/
     Ochratoxin_A_OC045_2_PROTON_nt32_27C_01.fid/Ochratoxin_A_OC045_2_PROTON_nt32_27C_01

Absolute intensity correction: I = 6.00344e+04

Relative intensities:
Component  1:   46.84084% | Rel:  1.00000
Component  2:   53.15916% | Rel:  1.13489
```

Component 1 fitted parameters:

Region;	Intensity
12.782:1.555;	4.04721881e+05

#;	u;	fwhm;	Rel. I.;	Phase;	Beta;	Group
1;	8.43515441;	1.332615;	0.082785;	0.000;	0.08955;	0
2;	12.75762810;	1.770240;	0.096212;	0.000;	0.00000;	0
3;	7.31687018;	1.197980;	0.018620;	0.000;	0.92588;	0
4;	3.38172767;	1.916283;	0.011405;	0.000;	0.57102;	0
5;	3.37110182;	1.775110;	0.011385;	0.000;	0.49614;	0
6;	3.35347152;	1.935750;	0.019323;	0.000;	0.39849;	0
7;	3.34288344;	2.019493;	0.019293;	0.000;	0.35768;	0
8;	3.31402426;	1.188121;	0.012178;	0.000;	0.46235;	0
9;	3.30716880;	1.207678;	0.012391;	0.000;	0.47738;	0
10;	3.27921358;	1.298124;	0.017971;	0.000;	0.18213;	0
11;	3.27231293;	1.296194;	0.018093;	0.000;	0.12801;	0
12;	3.25030908;	1.906229;	0.019353;	0.000;	0.41513;	0
13;	3.23592562;	1.825675;	0.019023;	0.000;	0.41187;	0
14;	3.22205110;	1.761811;	0.011496;	0.000;	0.55201;	0
15;	3.20774028;	1.728530;	0.011638;	0.000;	0.46413;	0
16;	2.89393065;	1.876560;	0.025449;	0.000;	0.00593;	0
17;	2.87089962;	1.845617;	0.024333;	0.000;	0.00037;	0
18;	2.85906338;	1.846489;	0.021364;	0.000;	0.00000;	0
19;	2.83600475;	1.846440;	0.021191;	0.000;	0.00106;	0
20;	8.49914954;	3.301411;	0.020044;	0.000;	0.99405;	0
21;	8.51270787;	3.206935;	0.018927;	0.000;	0.99989;	0
22;	1.59575975;	1.595538;	0.115574;	0.000;	0.02964;	0
23;	1.60841872;	1.602572;	0.106630;	0.000;	0.23009;	0
24;	7.33350538;	1.033471;	0.002247;	0.000;	1.00000;	0
25;	7.33039597;	1.530990;	0.012888;	0.000;	0.00289;	0
26;	7.34175844;	29.398998;	0.000443;	0.000;	0.99979;	0
27;	7.32691166;	1.421124;	0.002420;	0.000;	0.00000;	0
28;	7.31870099;	2.179645;	0.007664;	0.000;	1.00000;	0
29;	7.31409876;	1.472612;	0.018031;	0.000;	0.68904;	0
30;	7.30456754;	1.373436;	0.006965;	0.000;	0.99988;	0
31;	7.30190227;	1.251652;	0.025683;	0.000;	0.68036;	0
32;	7.29945881;	1.323001;	0.005552;	0.000;	1.00000;	0
33;	7.30000170;	0.501940;	0.001302;	0.000;	1.00000;	0
34;	7.28171968;	1.023946;	0.002756;	0.000;	0.00055;	0
35;	7.27887261;	1.463965;	0.012160;	0.000;	0.39327;	0
36;	7.27516226;	2.243506;	0.010839;	0.000;	0.92964;	0
37;	7.27252475;	0.813467;	0.018239;	0.000;	0.75445;	0
38;	7.26485854;	1.410629;	0.017946;	0.000;	0.51589;	0
39;	7.26018317;	1.570804;	0.039693;	0.000;	0.55905;	0
40;	7.25703809;	1.453860;	0.028590;	0.000;	0.85652;	0
41;	7.25326598;	1.514649;	0.003691;	0.000;	1.00000;	0
42;	7.24771101;	2.418794;	0.014174;	0.000;	0.70633;	0
43;	7.24393719;	1.569469;	0.024594;	0.000;	0.50379;	0
44;	7.24137076;	1.298516;	0.009446;	0.000;	0.84210;	0

Component 2 fitted parameters:

Region;	Intensity
12.782:1.555;	3.19137865e+04

#;	u;	fwhm;	Rel. I.;	Phase;	Beta;	Group
1;	7.74855943;	0.902832;	1.000000;	0.000;	0.26162;	0

A7.4.3 Histogram of the residuals

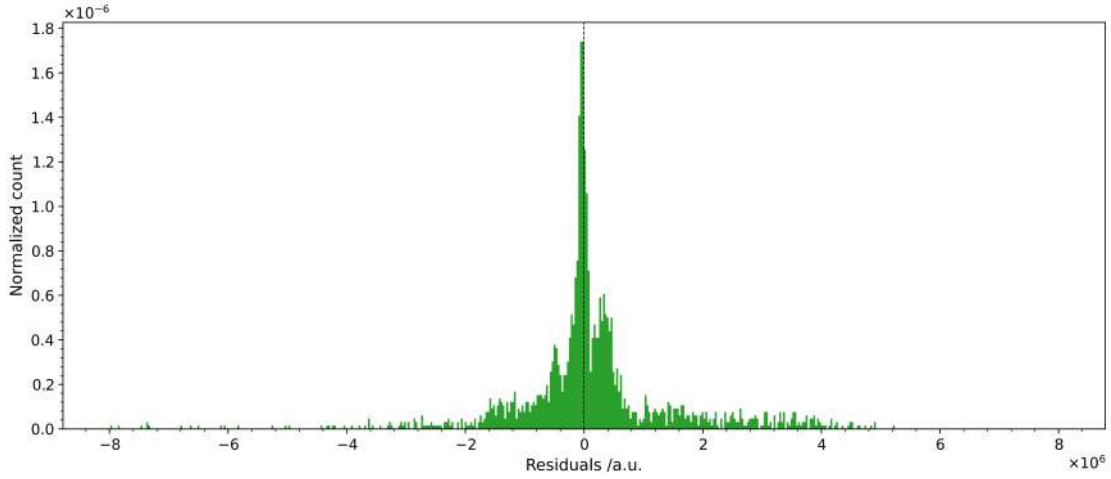


Figure A7.3: Histogram of the residuals of the `pyIHM` deconvolution of the ochratoxin-a − TCNB mixture.

A7.5 Mock urine

A7.5.1 Input file

```
BASE_FILENAME
urine_nuovo

MIX_PATH
/media/francesco/Volume/NMR/frans/fake-urine-140217/8/

MIX_SPECTRUM_TXT
r4

COMP_PATH
./components/formic_acid.fvf, 1H
./components/acetic_acid.fvf, 3H
./components/citrate.fvf, 4H
./components/lactate.fvf, 4H
./components/alanine.fvf, 4H
./components/glutamine.fvf, 9H
./components/creatinine.fvf, 5H
./components/a-glucose_exp.fvf, 7H
./components/b-glucose_exp.fvf, 7H
./components/urea.fvf, 4H
./components/DSS.fvf, 15H

FIT_KWS
method='nelder', tol=1e-4, max_nfev=10000
method='leastsq', tol=1e-5, max_nfev=20000
method='leastsq', tol=1e-5, max_nfev=20000
method='leastsq', tol=1e-5, max_nfev=20000

FIT_BDS
utol=0.10
utol_sg=0.01
stol=20
ktol=0.01

PLT_OPTS
ext=png, dpi=300

FIT_LIMITS
  8.396,  8.372
  5.769,  5.651
  5.186,  5.151
```

```
  4.598,  4.566
  4.071,  4.024
  4.004,  3.978
  3.855,  3.816
  3.800,  3.623
  3.491,  3.315
  3.205,  3.148
  2.986,  2.964
  2.875,  2.829
  2.616,  2.562
  2.495,  2.333
  2.117,  2.016
  1.863,  1.836
  1.731,  1.665
  1.427,  1.399
  1.278,  1.244
  0.594,  0.550
 -0.042, -0.071

I0
8.4000, 5.0000, 5.0000, 0.7000, 2.8000, 1.8000, 3.5000, 1.8000, 1.9000, 0.2000, 0.5000
```

A7.5.2 Output file

```
! Fit performed by francesco on 31/10/2024 at 20:07:54

Mixture spectrum: /media/francesco/Volume/NMR/frans/fake-urine-140217/8/

Absolute intensity correction: I = 1.09283e+03

Relative intensities:
Component  1:   16.65816% | Rel: 21.67644
Component  2:   13.91932% | Rel: 18.11252
Component  3:   15.71735% | Rel: 20.45220
Component  4:    3.12794% | Rel:  4.07023
Component  5:    9.63414% | Rel: 12.53643
Component  6:   12.05437% | Rel: 15.68576
Component  7:   13.79853% | Rel: 17.95534
Component  8:    6.04480% | Rel:  7.86580
Component  9:    7.06277% | Rel:  9.19043
Component 10:    0.76849% | Rel:  1.00000
Component 11:    1.21413% | Rel:  1.57989



Component 1 fitted parameters:
         Region;     Intensity
-----------------------------------------------------------------------------------------------
    8.396:-0.071;  1.82045228e+02

   #;            u;          fwhm;    Rel. I.;     Phase;      Beta;     Group
-----------------------------------------------------------------------------------------------
   1;     8.38770883;      0.567737;  1.000000;     0.000;    0.21318;         0
-----------------------------------------------------------------------------------------------

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



Component 2 fitted parameters:
         Region;     Intensity
-----------------------------------------------------------------------------------------------
    8.396:-0.071;  4.56343062e+02

   #;            u;          fwhm;    Rel. I.;     Phase;      Beta;     Group
-----------------------------------------------------------------------------------------------
   1;     1.85164021;      0.641793;  1.000000;     0.000;    0.32259;         0
-----------------------------------------------------------------------------------------------

===============================================================================================
```

Component 3 fitted parameters:

Region;	Intensity
8.396:-0.071;	6.87054827e+02

#;	u;	fwhm;	Rel. I.;	Phase;	Beta;	Group
1;	2.57882801;	0.868601;	0.307145;	0.000;	0.11144;	0
2;	2.60401921;	0.812547;	0.195729;	0.000;	0.14884;	0
3;	2.45631465;	0.857385;	0.186960;	0.000;	0.29892;	0
4;	2.48153489;	0.881611;	0.310167;	0.000;	0.08176;	0

Component 4 fitted parameters:

Region;	Intensity
8.396:-0.071;	1.36732009e+02

#;	u;	fwhm;	Rel. I.;	Phase;	Beta;	Group
1;	1.25584461;	0.761241;	0.384490;	0.000;	0.10843;	0
2;	1.26740829;	0.752658;	0.288576;	0.000;	0.64205;	0
3;	4.03116302;	0.996955;	0.040038;	0.000;	0.00000;	0
4;	4.04269862;	0.977127;	0.123676;	0.000;	0.01361;	0
5;	4.05422565;	0.990178;	0.123676;	0.000;	0.01208;	0
6;	4.06573697;	1.014722;	0.039543;	0.000;	0.00037;	0

Component 5 fitted parameters:

Region;	Intensity
8.396:-0.071;	4.21138688e+02

#;	u;	fwhm;	Rel. I.;	Phase;	Beta;	Group
1;	3.69560757;	0.827502;	0.014276;	0.000;	0.00806;	0
2;	3.70631394;	1.207209;	0.037855;	0.000;	0.08123;	0
3;	3.71491564;	0.987894;	0.037855;	0.000;	0.11473;	0
4;	3.72393821;	2.403318;	0.009323;	0.000;	0.01302;	0
5;	1.40599556;	0.893041;	0.459647;	0.000;	0.00789;	0
6;	1.41806941;	0.896064;	0.441044;	0.000;	0.10722;	0

Component 6 fitted parameters:

Region;	Intensity
8.396:-0.071;	6.58805486e+02

#;	u;	fwhm;	Rel. I.;	Phase;	Beta;	Group
1;	2.05616956;	1.509403;	0.039071;	0.000;	0.62369;	0
2;	2.06000701;	1.255920;	0.028784;	0.000;	0.00000;	0
3;	2.06291445;	0.934584;	0.018552;	0.000;	0.12146;	0
4;	2.06450801;	0.991708;	0.019075;	0.000;	0.47559;	0
5;	2.06718757;	1.177831;	0.042427;	0.000;	0.23756;	0
6;	2.07094654;	1.269713;	0.043748;	0.000;	0.33057;	0
7;	2.07445058;	1.063305;	0.026399;	0.000;	0.01893;	0
8;	2.07674285;	0.912912;	0.017165;	0.000;	0.41197;	0
9;	2.07874336;	1.421284;	0.022458;	0.000;	0.98525;	0
10;	2.08204251;	1.120395;	0.015663;	0.000;	1.00000;	0
11;	2.08855435;	1.896697;	0.047874;	0.000;	0.00045;	0

```
  12;     2.10053477;      0.767624;  0.001471;     0.000;    1.00000;         0
  13;     2.09929794;      0.848393;  0.002005;     0.000;    0.53657;         0
  14;     2.10290552;      1.278288;  0.004402;     0.000;    0.04751;         0
  15;     2.11283790;      1.019512;  0.002727;     0.000;    0.34304;         0
  16;     2.39403832;      1.126330;  0.029076;     0.000;    0.10058;         0
  17;     2.39521137;      1.246138;  0.044858;     0.000;    0.46534;         0
  18;     2.38816716;      1.958265;  0.009747;     0.000;    0.00000;         0
  19;     2.02067803;      1.003056;  0.002533;     0.000;    0.17547;         0
  20;     2.03143701;      1.166967;  0.002802;     0.000;    0.00009;         0
  21;     2.03295874;      1.040784;  0.003317;     0.000;    0.00023;         0
  22;     2.03514004;      0.861968;  0.001483;     0.000;    0.67022;         0
  23;     2.04502475;      1.256202;  0.025884;     0.000;    0.03633;         0
  24;     2.05301950;      1.075056;  0.022069;     0.000;    0.02005;         0
  25;     2.34414491;      1.183907;  0.008779;     0.000;    0.00220;         0
  26;     2.36980611;      1.176813;  0.054454;     0.000;    0.08787;         0
  27;     2.35550336;      1.181040;  0.008449;     0.000;    0.00859;         0
  28;     2.35776554;      0.973555;  0.005435;     0.000;    0.00010;         0
  29;     2.37951909;      1.236682;  0.046700;     0.000;    0.23013;         0
  30;     2.38142768;      1.168107;  0.045833;     0.000;    0.00240;         0
  31;     2.38311000;      1.085016;  0.038625;     0.000;    0.24315;         0
  32;     2.39191818;      1.210215;  0.047476;     0.000;    0.17991;         0
  33;     2.41727355;      1.303252;  0.008706;     0.000;    0.00100;         0
  34;     2.42028888;      1.094471;  0.005707;     0.000;    0.32237;         0
  35;     2.40585398;      1.325950;  0.044573;     0.000;    0.40023;         0
  36;     2.43169677;      1.294013;  0.007684;     0.000;    0.00002;         0
  37;     3.69627883;      1.025185;  0.052099;     0.000;    0.43712;         0
  38;     3.70670475;      1.274129;  0.094377;     0.000;    0.67016;         0
  39;     3.71680501;      1.110566;  0.057515;     0.000;    0.52442;         0
-------------------------------------------------------------------------------------------

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


Component 7 fitted parameters:
         Region;     Intensity
-------------------------------------------------------------------------------------------
    8.396:-0.071;  7.53971515e+02

   #;            u;          fwhm;   Rel. I.;     Phase;      Beta;     Group
-------------------------------------------------------------------------------------------
   1;     2.97754886;      1.206022;  0.600768;     0.000;    0.59542;         0
   2;     3.99126659;      1.353676;  0.399232;     0.000;    0.54173;         0
-------------------------------------------------------------------------------------------

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


Component 8 fitted parameters:
         Region;     Intensity
-------------------------------------------------------------------------------------------
    8.396:-0.071;  4.62415152e+02

   #;            u;          fwhm;   Rel. I.;     Phase;      Beta;     Group
-------------------------------------------------------------------------------------------
   1;     5.16668320;      1.429061;  0.056641;     0.000;    0.71613;         0
   2;     5.17293364;      1.447857;  0.051621;     0.000;    0.89797;         0
   3;     3.46018080;      1.072118;  0.035160;     0.000;    0.00050;         0
   4;     3.47652927;      0.914292;  0.035160;     0.000;    0.01018;         0
   5;     3.46646438;      1.025314;  0.035160;     0.000;    0.00065;         0
   6;     3.48282391;      0.901670;  0.035160;     0.000;    0.06386;         0
   7;     3.63374791;      1.207183;  0.043041;     0.000;    0.10635;         0
   8;     3.64943868;      1.477339;  0.045864;     0.000;    0.99999;         0
   9;     3.65517293;      2.912376;  0.043041;     0.000;    0.00003;         0
  10;     3.66641533;      1.025253;  0.043041;     0.000;    0.16613;         0
  11;     3.33279214;      1.156747;  0.022050;     0.000;    0.01111;         0
  12;     3.34943609;      1.945652;  0.022050;     0.000;    0.78513;         0
  13;     3.34828950;      1.851947;  0.022050;     0.000;    0.92813;         0
  14;     3.36476178;      1.377820;  0.024114;     0.000;    0.69937;         0
  15;     3.75018051;      1.519931;  0.017226;     0.000;    0.60219;         0
  16;     3.75835450;      1.606396;  0.007740;     0.000;    0.41094;         0
  17;     3.75390403;      1.403791;  0.015722;     0.000;    0.73088;         0
  18;     3.76693443;      1.386522;  0.020493;     0.000;    0.67272;         0
  19;     3.76909596;      1.015044;  0.020493;     0.000;    0.25747;         0
```

```
  20;     3.77639197;      1.252358;  0.016226;     0.000;    0.53741;         0
  21;     3.77259871;      1.612870;  0.016256;     0.000;    0.00967;         0
  22;     3.78012723;      1.336114;  0.004590;     0.000;    0.34203;         0
  23;     3.78846410;      1.132652;  0.020470;     0.000;    0.31688;         0
  24;     3.76209037;      1.436562;  0.017289;     0.000;    0.75117;         0
  25;     3.79223922;      1.106610;  0.015728;     0.000;    0.27430;         0
  26;     3.77075912;      1.105379;  0.033201;     0.000;    0.39796;         0
  27;     3.72663020;      0.766315;  0.099819;     0.000;    0.33261;         0
  28;     3.71451237;      0.704050;  0.092881;     0.000;    0.04818;         0
  29;     3.73869531;      0.948879;  0.043041;     0.000;    0.00181;         0
  30;     3.70248616;      0.862254;  0.044677;     0.000;    0.00027;         0
-----------------------------------------------------------------------------------------------

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


Component 9 fitted parameters:
         Region;     Intensity
-----------------------------------------------------------------------------------------------
    8.396:-0.071;  5.40287710e+02

   #;             u;           fwhm;    Rel. I.;     Phase;       Beta;      Group
-----------------------------------------------------------------------------------------------
   1;     4.57556960;      1.120470;  0.085457;     0.000;    0.00004;         0
   2;     4.58882798;      1.136908;  0.085432;     0.000;    0.00001;         0
   3;     3.16456273;      0.844515;  0.044646;     0.000;    0.07311;         0
   4;     3.18017718;      0.850473;  0.044738;     0.000;    0.31296;         0
   5;     3.17779250;      0.849504;  0.044738;     0.000;    0.16094;         0
   6;     3.19338612;      0.856506;  0.044738;     0.000;    0.34496;         0
   7;     3.41035449;      0.717974;  0.037850;     0.000;    0.00163;         0
   8;     3.42549878;      0.944406;  0.036331;     0.000;    0.00487;         0
   9;     3.42599695;      0.954339;  0.034812;     0.000;    0.97541;         0
  10;     3.44103976;      0.948293;  0.031772;     0.000;    0.38478;         0
  11;     3.32289220;      0.966743;  0.024176;     0.000;    0.28992;         0
  12;     3.33924197;      1.020496;  0.031773;     0.000;    0.90822;         0
  13;     3.33782074;      0.870968;  0.031773;     0.000;    0.28080;         0
  14;     3.35423924;      0.980125;  0.054564;     0.000;    0.50218;         0
  15;     3.38561240;      0.994295;  0.021141;     0.000;    0.34359;         0
  16;     3.39526219;      0.936028;  0.022660;     0.000;    0.28373;         0
  17;     3.38930564;      0.906113;  0.024180;     0.000;    0.01728;         0
  18;     3.39896118;      0.949733;  0.021141;     0.000;    0.22183;         0
  19;     3.40188481;      1.041648;  0.010505;     0.000;    0.00001;         0
  20;     3.41111976;      1.132396;  0.018102;     0.000;    1.00000;         0
  21;     3.40571471;      0.988766;  0.013544;     0.000;    0.01726;         0
  22;     3.41523946;      0.946148;  0.013544;     0.000;    0.00556;         0
  23;     3.84434376;      1.117019;  0.027212;     0.000;    0.57736;         0
  24;     3.82384918;      1.075083;  0.031773;     0.000;    0.54873;         0
  25;     3.84797931;      1.083196;  0.025695;     0.000;    0.48344;         0
  26;     3.82750737;      1.051639;  0.031773;     0.000;    0.46044;         0
  27;     3.66535643;      1.280223;  0.027970;     0.000;    0.24282;         0
  28;     3.64596649;      0.991729;  0.026910;     0.000;    0.57495;         0
  29;     3.67608691;      1.000500;  0.026910;     0.000;    0.66298;         0
  30;     3.65565945;      1.087758;  0.024141;     0.000;    0.00035;         0
-----------------------------------------------------------------------------------------------

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


Component 10 fitted parameters:
         Region;     Intensity
-----------------------------------------------------------------------------------------------
    8.396:-0.071;  3.35931951e+01

   #;             u;           fwhm;    Rel. I.;     Phase;       Beta;      Group
-----------------------------------------------------------------------------------------------
   1;     5.70910483;     48.195708;  1.000000;     0.000;    0.23890;         0
-----------------------------------------------------------------------------------------------

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


Component 11 fitted parameters:
```

```
        Region;     Intensity
-------------------------------------------------------------------------------------------------
    8.396:-0.071;  1.96326041e+02

   #;            u;          fwhm;   Rel. I.;     Phase;      Beta;      Group
-------------------------------------------------------------------------------------------------
   1;   - 0.05453742;     0.663997;  0.487909;     0.000;    0.16679;         0
   2;     0.58496160;     1.292283;  0.039185;     0.000;    0.00007;         0
   3;     0.57659157;     2.438165;  0.019871;     0.000;    0.00691;         0
   4;     0.57082420;     2.594565;  0.051147;     0.000;    0.00000;         0
   5;     0.56551024;     2.566677;  0.019587;     0.000;    0.11728;         0
   6;     0.55668799;     1.404314;  0.038838;     0.000;    0.00283;         0
   7;     1.72595839;     1.075715;  0.008550;     0.000;    0.07529;         0
   8;     1.71722811;     2.276197;  0.022939;     0.000;    0.00000;         0
   9;     1.70849276;     5.366332;  0.028379;     0.000;    0.99957;         0
  10;     1.69864640;     1.960678;  0.029963;     0.000;    0.48856;         0
  11;     1.69113680;     3.275392;  0.031077;     0.000;    0.05722;         0
  12;     1.68476281;     1.880277;  0.024011;     0.000;    0.00257;         0
  13;     1.67991301;     1.533601;  0.007875;     0.000;    0.99980;         0
  14;     1.67125650;     1.493898;  0.014512;     0.000;    0.00054;         0
  15;     2.86653441;     1.202112;  0.040439;     0.000;    0.00763;         0
  16;     2.85860919;     1.102907;  0.010176;     0.000;    1.00000;         0
  17;     2.85358501;     2.764507;  0.070569;     0.000;    0.00000;         0
  18;     2.84835183;     1.565814;  0.007761;     0.000;    0.66310;         0
  19;     2.84018823;     1.405388;  0.047211;     0.000;    0.00005;         0
-------------------------------------------------------------------------------------------------

=================================================================================================
```

A7.5.3 Histogram of the residuals

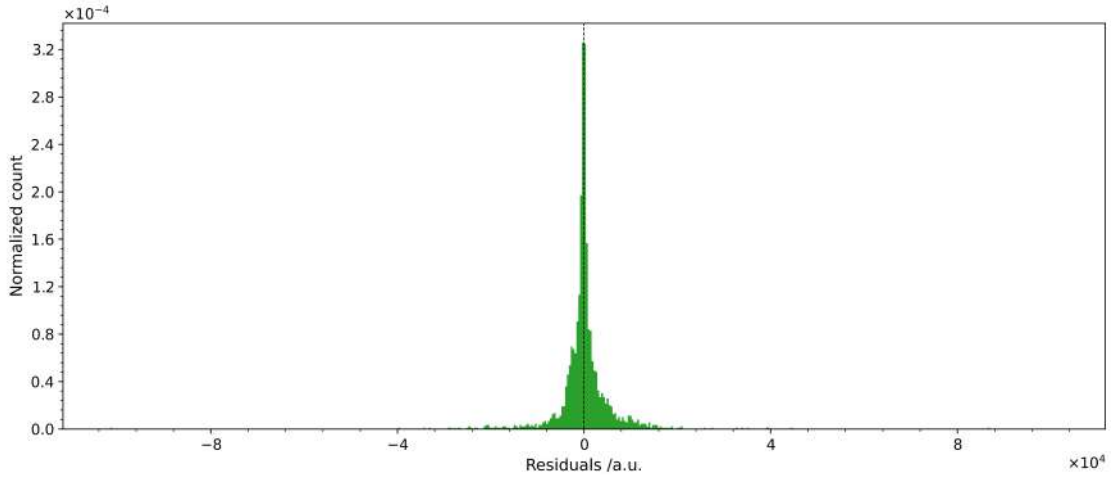


Figure A7.4: Histogram of the residuals of the `pyIHM` deconvolution of the synthetic urine sample.

References

[1] Harald Schwalbe et al. "The future of integrated structural biology". In: *Structure* 32.10 (2024), pp. 1563–1580.

[2] Lucia Banci et al. "Biomolecular NMR at 1.2 GHz". In: *arXiv preprint arXiv:1910.07462* (2019).

[3] Harald Schwalbe. "new 1.2 GHz NMR spectrometers-new horizons?" In: *Angewandte Chemie (International ed. in English)* 56.35 (2017), pp. 10252–10253.

[4] Ewen Lescop, Thomas Kern, and Bernhard Brutscher. "Guidelines for the use of band-selective radiofrequency pulses in hetero-nuclear NMR: example of longitudinal-relaxation-enhanced BEST-type 1H–15N correlation experiments". In: *Journal of Magnetic Resonance* 203.1 (2010), pp. 190–198.

[5] Paul Schanda, Ēriks Kupče, and Bernhard Brutscher. "SOFAST-HMQC experiments for recording two-dimensional deteronuclear correlation spectra of proteins within a few seconds". In: *Journal of biomolecular NMR* 33 (2005), pp. 199–211.

[6] Mateusz Urbańczyk, Wiktor Koźmiński, and Krzysztof Kazimierczuk. "Accelerating diffusion-ordered NMR spectroscopy by joint sparse sampling of diffusion and time dimensions". In: *Angewandte Chemie International Edition* 53.25 (2014), pp. 6464–6467.

[7] Md Sharif Ullah et al. "Ultrafast transverse relaxation exchange NMR spectroscopy". In: *Physical Chemistry Chemical Physics* 24.36 (2022), pp. 22109–22114.

[8] EO Brigham. *The fast Fourier transform and its applications*. 1988.

[9] James A Cadzow. "Signal enhancement-a composite property mapping algorithm". In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 36.1 (1988), pp. 49–62.

[10] J-P Antoine. "Wavelets and wavelet frames on the 2-sphere". In: *Contemporary problems in mathematical physics*. World Scientific, 2006, pp. 344–362.

[11] Gabriel Vivó-Truyols and Peter J Schoenmakers. "Automatic selection of optimal Savitzky-Golay smoothing". In: *Analytical chemistry* 78.13 (2006), pp. 4598–4608.

[12] Lionel Chiron et al. "Efficient denoising algorithms for large experimental datasets and their applications in Fourier transform ion cyclotron resonance mass spectrometry". In: *Proceedings of the National Academy of Sciences* 111.4 (2014), pp. 1385–1390.

[13] KR Metz, MM Lam, and AG Webb. "Reference deconvolution: a simple and effective method for resolution enhancement in nuclear magnetic resonance spectroscopy". In: *Concepts in Magnetic resonance* 12.1 (2000), pp. 21–42.

[14] Guillaume Laurent et al. "Denoising applied to spectroscopies–part I: concept and limits". In: *Applied Spectroscopy Reviews* 54.7 (2019), pp. 602–630.

[15] Francesco Bruno et al. "Multivariate Curve Resolution for 2D solid-state NMR spectra". In: *Analytical chemistry* 92.6 (2020), pp. 4451–4458.

[16] Frank Delaglio et al. “NMRPipe: a multidimensional spectral processing system based on UNIX pipes”. In: *Journal of biomolecular NMR* 6.3 (1995), pp. 277–293.

[17] 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* 55.4 (2013), pp. 355–367.

[18] Laura Castañar et al. “The GNAT: A new tool for processing NMR data”. In: *Magnetic Resonance in Chemistry* 56.6 (2018), pp. 546–558.

[19] Tanja Beyer, Bernd Diehl, and Ulrike Holzgrabe. “Quantitative NMR spectroscopy of biologically active substances and excipients”. In: *Bioanalytical Reviews* 2 (2010), pp. 1–22.

[20] E Kriesten et al. “Identification of unknown pure component spectra by indirect hard modeling”. In: *Chemometrics and Intelligent Laboratory Systems* 93.2 (2008), pp. 108–119.

[21] 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.

[22] John Cavanagh. *Protein NMR spectroscopy: principles and practice*. Academic press, 1996.

[23] Halfdan Grage and Mikael Akke. “A statistical analysis of NMR spectrometer noise”. In: *Journal of magnetic resonance* 162.1 (2003), pp. 176–188.

[24] AF Mehlkopf et al. “Sources of t1 noise in two-dimensional NMR”. In: *Journal of Magnetic Resonance (1969)* 58.2 (1984), pp. 315–323.

[25] Caroline Brissac, Thérèse E Malliavin, and Marc A Delsuc. “Use of the Cadzow procedure in 2D NMR for the reduction of t1 noise”. In: *Journal of Biomolecular NMR* 6.4 (1995), pp. 361–365.

[26] William H Press and Brian P Flannery. *Numerical recipes*. Citeseer.

[27] A Heuer. “A new algorithm for automatic phase correction by symmetrizing lines”. In: *Journal of Magnetic Resonance (1969)* 91.2 (1991), pp. 241–253.

[28] Richard R Ernst. “Numerical Hilbert transform and automatic phase correction in magnetic resonance spectroscopy”. In: *Journal of Magnetic Resonance (1969)* 1.1 (1969), pp. 7–26.

[29] Edward C Craig and Alan G Marshall. “Automated phase correction of FT NMR spectra by means of phase measurement based on dispersion versus absorption relation (DISPA)”. In: *Journal of Magnetic Resonance (1969)* 76.3 (1988), pp. 458–475.

[30] Li Chen et al. “An efficient algorithm for automatic phase correction of NMR spectra based on entropy minimization”. In: *Journal of Magnetic Resonance* 158.1-2 (2002), pp. 164–168.

[31] Stanislav Sokolenko et al. “Robust 1D NMR lineshape fitting using real and imaginary data in the frequency domain”. In: *Journal of Magnetic Resonance* 298 (2019), pp. 91–100.

[32] Paul HC Eilers. “A perfect smoother”. In: *Analytical chemistry* 75.14 (2003), pp. 3631–3636.

[33] Ioannis-Angelos Giapitzakis et al. “Investigation of the influence of macromolecules and spline baseline in the fitting model of human brain spectra at 9.4 T”. In: *Magnetic resonance in medicine* 81.2 (2019), pp. 746–758.

[34] Mathias Sawall et al. “Multi-objective optimization for an automated and simultaneous phase and baseline correction of NMR spectral data”. In: *Journal of Magnetic Resonance* 289 (2018), pp. 132–141.

[35] Vincent Mazet et al. “Background removal from spectra by designing and minimising a non-quadratic cost function”. In: *Chemometrics and intelligent laboratory systems* 76.2 (2005), pp. 121–133.

[36] Antonio Cicone and Haomin Zhou. *Numerical Analysis for Iterative Filtering with New Efficient Implementations Based on FFT*. arXiv:1802.01359 [math]. Oct. 2018. DOI: 10.48550/arXiv.1802.01359. URL: http://arxiv.org/abs/1802.01359 (visited on 05/20/2024).

[37] Antonio Cicone. “Iterative filtering as a direct method for the decomposition of nonstationary signals”. en. In: *Numerical Algorithms* 85.3 (Nov. 2020), pp. 811–827. ISSN: 1572-9265. DOI: 10.1007/s11075-019-00838-z. URL: https://doi.org/10.1007/s11075-019-00838-z (visited on 05/20/2024).

[38] Norden E. Huang et al. “The empirical mode decomposition and the Hilbert spectrum for nonlinear and non-stationary time series analysis”. In: *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* 454.1971 (Mar. 1998). Publisher: Royal Society, pp. 903–995. DOI: 10.1098/rspa.1998.0193. URL: https://royalsocietypublishing.org/doi/10.1098/rspa.1998.0193 (visited on 05/20/2024).

[39] Marl A Delsuc. “Spectral representation of 2D NMR spectra by hypercomplex numbers”. In: *Journal of Magnetic Resonance (1969)* 77.1 (1988), pp. 119–124.

[40] Reza Ghanati et al. “Filtering and parameter estimation of surface-NMR data using singular spectrum analysis”. In: *Journal of Applied Geophysics* 130 (2016), pp. 118–130.

[41] Pascal P Man, Christian Bonhomme, and Florence Babonneau. “Denoising NMR time-domain signal by singular-value decomposition accelerated by graphics processing units”. In: *Solid State Nuclear Magnetic Resonance* 61 (2014), pp. 28–34.

[42] Francesco Bruno et al. “Lysozyme is Sterically Trapped Within the Silica Cage in Bioinspired Silica–Lysozyme Composites: A Multi-Technique Understanding of Elusive Protein–Material Interactions”. In: *Langmuir* 38.26 (2022), pp. 8030–8037.

[43] Willem Windig and Jean Guilment. “Interactive self-modeling mixture analysis”. In: *Analytical chemistry* 63.14 (1991), pp. 1425–1432.

[44] F Cuesta Sánchez and DL Massart. “Application of SIMPLISMA for the assessment of peak purity in liquid chromatography with diode array detection”. In: *Analytica chimica acta* 298.3 (1994), pp. 331–339.

[45] Isabella C Felli and Roberta Pierattelli. “13C direct detected NMR for challenging systems”. In: *Chemical Reviews* 122.10 (2022), pp. 9468–9496.

[46] Francesco Bruno, Letizia Fiorucci, and Enrico Ravera. “Sensitivity considerations on denoising series of spectra by singular value decomposition”. In: *Magnetic Resonance in Chemistry* 61.6 (2023), pp. 373–379.

[47] Anna de Juan and Roma Tauler. “Multivariate Curve Resolution: 50 years addressing the mixture analysis problem–A review”. In: *Analytica Chimica Acta* 1145 (2021), pp. 59–78.

[48] Henri P Gavin. “The Levenberg-Marquardt algorithm for nonlinear least squares curve-fitting problems”. In: *Department of Civil and Environmental Engineering Duke University August* 3 (2019).

[49] Carl T Kelley. *Iterative methods for optimization*. SIAM, 1999.

[50] Constantino Tsallis and Daniel A Stariolo. "Generalized simulated annealing". In: *Physica A: Statistical Mechanics and its Applications* 233.1-2 (1996), pp. 395–406.

[51] Olivier Cappé, Simon J Godsill, and Eric Moulines. "An overview of existing methods and recent advances in sequential Monte Carlo". In: *Proceedings of the IEEE* 95.5 (2007), pp. 899–924.

[52] Samantha Caputo et al. "Study and application of graphene oxide in the synthesis of 2, 3-disubstituted quinolines via a Povarov multicomponent reaction and subsequent oxidation". In: *RSC advances* 12.25 (2022), pp. 15834–15847.

[53] Andrea Brunetti et al. "Graphene-Oxide Mediated Chemodivergent Ring-Opening of Cyclobutanols". In: *Chinese Journal of Chemistry* 41.11 (2023), pp. 1333–1340.

[54] M Villarruel Dujovne et al. "Introducing NMR strategies to define water molecules that drive metal binding in a transcriptional regulator". In: *Journal of Magnetic Resonance Open* 16 (2023), p. 100114.

[55] Panteleimon G Takis et al. "Mapping of 1H NMR chemical shifts relationship with chemical similarities for the acceleration of metabolic profiling: Application on blood products". In: *Magnetic Resonance in Chemistry* 61.12 (2023), pp. 759–769.

[56] Maxwell Venetos et al. "Deconvolution and Analysis of 1H NMR Spectra of Crude Reaction Mixtures". In: (2023).

[57] Simon Kern et al. "Mathematical and statistical tools for online NMR spectroscopy in chemical processes". In: *Advanced Mathematical and Computational Tools in Metrology and Testing XI*. World Scientific, 2019, pp. 229–234.

[58] Florian Fricke et al. "Artificial Intelligence for Mass Spectrometry and Nuclear Magnetic Resonance Spectroscopy Using a Novel Data Augmentation Method". In: *IEEE Transactions on Emerging Topics in Computing* 10.1 (Jan. 2022). Conference Name: IEEE Transactions on Emerging Topics in Computing, pp. 87–98. ISSN: 2168-6750. DOI: 10.1109/TETC.2021.3131371. URL: https://ieeexplore.ieee.org/abstract/document/9638378 (visited on 04/22/2024).

[59] Yu.V. Rakitin, G.M. Larin, and V.V. Minin. *Interpretatsiya spektrov EPR koordinatsionnykh soedinenii (Interpretation of EPR Spectra of Coordination Compounds)*. Russian. Moskow: Nauka, 1993.

[60] Hasan Husein Mor, Høgni Weihe, and Jesper Bendix. "Fitting of EPR spectra: The importance of a flexible bandwidth". In: *Journal of Magnetic Resonance* 207.2 (2010), pp. 283–286.

[61] Stefan Stoll. "Computational Modeling and Least-Squares Fittingof EPR Spectra". In: *Multifrequency electron paramagnetic resonance: Data and techniques* (2014), pp. 69–138.

[62] Frans AA Mulder, Leonardo Tenori, and Claudio Luchinat. "Fast and quantitative NMR metabolite analysis afforded by a paramagnetic co-solute". In: *Angewandte Chemie* 131.43 (2019), pp. 15427–15430.

[63] David Marks and Shimon Vega. "A theory for cross-polarization NMR of nonspinning and spinning samples". In: *Journal of Magnetic Resonance, Series A* 118.2 (1996), pp. 157–172.

[64] Isabella C Felli and Roberta Pierattelli. “Spin-state-selective methods in solution-and solid-state biomolecular 13C NMR”. In: *Progress in Nuclear Magnetic Resonance Spectroscopy* 84 (2015), pp. 1–13.

La borsa di dottorato è stata cofinanziata con risorse del
Programma Operativo Nazionale Ricerca e Innovazione 2014-2020, risorse FSE REACT-EU
Azione IV.4 “Dottorati e contratti di ricerca su tematiche dell’innovazione”
e Azione IV.5 “Dottorati su tematiche Green”