The xpy3-tools tutorial

Oleg V. Petrov | KFNT MFF | Univerzita Karlova

Version 1.6

31 October 2025

Table of Contents

1 Introduction

`xpy3-tools` is a collection of Python scripts for NMR data processing, intended to be suitable for use with Bruker's Topspin. As such, they rely on Bruker's Python API available since Topspin 4.1.3 [1]. With later versions of Topspin, you can run the scripts directly from command line by typing `xpy3 <script name>`, in a similar manner to conventional Jython scripts and Bruker's AU programs. The scripts solve the tasks from the author's own practice but hopefully can be of interest to other NMR practitioners. The current version of `xpy3-tools` comprises 15 scripts (see Table 1), which number is meant to grow given an ever-populated to-do list.

Using Topspin environment means, among other things, a direct access to its GUI. Yet a few scripts in `xpy3-tools` have their own GUI – pop-up windows for interactive parameter optimization and visual control. The graphical functionality of `xpy3-tools` is implemented by means of `matplotlib` / `Tkinter` methods, while data I/O is based on a module `BrukerIO` from a `JTutils` package (https://github.com/jtrebosc/JTutils).

It should be noted that `xpy3-tools` will not modify in place `fid`, `ser`, `acqu(123)` or other files of raw data. When those files get overwritten by design, as *e.g.* in de-noising or linear prediction scripts, it is done after first copying entire dataset to a new `EXPNO`, thus keeping original data intact. In other cases data are processed in-place, the result being saved in a `pdata` folder under the same `EXPNO` from whence a script is being called.

2 List of scripts

Script name	*Function*	*Comments*
`baseline`	Baseline correction on 1D data.	Uses a rolling standard deviation distribution method to identify baseline points [4] and a Whittaker method for their interpolation [5].
`baseline2`	`baseline` run on individual spectra in a pseudo-2D data set.	Allows navigation across a 2D data set for a manual baseline adjustment on the spectra.
`cpmg_fp`	Whole-echo processing for CPMG multi-echo trains.	Relies on zero points in a CPMG signal to split it into individual echoes. To have such zeros, consider a `op_qcpmg` pulse program.
`cpmg_recursive`	T_2 relaxation analysis of a CPMG echo train using a mono-exponential recursive formula.	Quantifies individual echoes, plots their cumulative sum vs time and fits with a mono-exponential recursive model [6].
`cpmg_relaxation`	T_2 relaxation analysis of a CPMG echo train using mono- or stretched-exponential model.	Plot individual echo intensity vs echo time and fits the curve with mono- or stretched exponential.
`denoise`	Low-rank SVD approximation of a 1D signal in time domain (a Cadzow's de-noising scheme).	Uses a Toepitz matrix format to represent the FID in 2D. An optimum SVD threshold value is computed according to [7].
`denoise2`	Approximation of a 2D signal in time domain using noise-adaptive principal components.	Employs NUS randomization of the acquired data to perturb the principal components and thus to average their noise-related content prior to use them in data approximation [8].
`icoshift`	Peak alignment over a series of spectra – either stacked in a pseudo-2D data set or linked via Topspin's multi display.	A number of peak regions can be selected for individual alignment; or you can choose one particular region to align the entire spectrum.
`lp`	Forward linear prediction on 1D data	Employs an original PCA-based projection method of LP (see below).
`phase_fid`	Zero-order phasing of 1D time-domain signals, including CPMG type signals.	Handy for maximizing the Re-part of the signal; has the option to nullify the Im-part.
`rotsync`	Construction of a rotor-synchronized MAS spectrum from an arbitrary sampled FID.	Enables quantitation of distinct chemical shift components directly through integration; useful in line shape analysis.
`rotsync_plus`	Same as `rotsync` but more involved rotary-echo quantitation.	In theory should provide better SNR than `rotsync`.
`sino_fid`	Signal-to-noise ratio (SNR) calculator for time-domain signals in 1D.	SNR is defined here as the ratio of the FID amplitude to 2 * std of noise per channel.
`symmetrize`	MAS sideband symmetrization for quadrupolar spectra of half-integer spins	Facilitates the fit of the 1st-order quadrupolar model to MAS spinning sidebands; produces good-looking spectra for presentations.

`t1t2`	Basic relaxation analysis module	Allows interactive integration over pseudo-2D spectra with optional correction for baseline shifts; currently includes mono-, two- and stretched exponential fit.
`utils`	Contains common utility functions and classes.	Ever growing collection of `def`s.

3 Installation

3.1 Python environment setup

Topspin 4.1.4+ comes with a preinstalled Python 3 environment in `<topspin>/python/lib/python3.x/` that includes Bruker Python API and few packages necessary for running demo scripts. To use another Python environment, you need to specify a path to its prefix. In Topspin 4.2+, you can do that via Topspin's Preferences going `Preferences` → `Python 3+` → `Select Python 3+ Environment`. In Topspin 4.1.x, you are bound to use the prefix `<topspin>/python/...` unless you backport a Jython script `xpy3.py` from Topspin 4.2+ where you can hard-code the path to Python of your choice. Backporting `xpy3.py` will also render `xpy3-tools` callable from Topspin 4.1.x command line, otherwise you can run them as regular console applications (using CLI or Windows' CMD) or from within a dedicated IDE such as Spyder or VSCode.

For instance, imagine that you have chosen to use a Python 3.x interpreter from `/usr/local/bin` on your Linux machine. Imagine also that your package installer is `pip`, namely `pip3.x`. To proceed to installation, open the terminal (CLI), change the current directory to `/usr/local/bin` and run the command

```
python3.x pip3.x install -r path/to/requirements.txt
```

with the configuration file `requirements.txt` shipped with `xpy3-tools`. This will install the required dependencies. Next, provide a search path for the Bruker Python API modules stored a `site-package` (or `dist-package`) folder in a Topspin's installation prefix. It can be done using a `.pth` file with a text line containing the path to the prefix, *e.g.*

```
/opt/topspin4.2.0/python/lib/python3.10/site-package/
```

The `.pth` file needs to be placed in a `site-package` directory of the chosen Python 3.x environment. Alternatively, you may re-install the Bruker Python API modules directly to the place using `.whl` files shipped with `<topspin>/python/examples`, as follows:

```
python3.x pip3.x install -r path/to/ts_remote_api*.whl
python3.x pip3.x install -r path/to/bruker_nmr_api*.whl
```

A special care is required for the module `BrukerIO`. The package `JTutils` where this module comes from cannot be installed via `pip`. Hence copy the `brukerIO.py` file directly from the `JTutils` repository at `https://github.com/jtrebosc/JTutils/blob/master/CpyLib/` and put it in the folder where `xpy3-tools` scripts are kept (see the next paragraph).

Installation on a Windows machine is supposed to be similar (not tested though).

3.2 Placement of scripts

Running `xpy3-tools` scripts directly from Topspin's command line requires a proper script location. The search path for Python 3 scripts is hard-coded in a `xpy3.py` file as `<topspin>/python/examples`. A simple workaround will be, therefore, to move the `xpy3-tools` files, including `brukerIO.py`, into `<topspin>/python/examples`. Should you prefer another location, add a path as follows. In a text editor,

open the file `<topspin>/exp/stan/nmr/py/xpy3.py` (you might need administrator's privileges for that), then find and comment out the line

```
return getParfileDirs(PYTHON_3_INDEX)
```

and put instead the lines (mind indents):

```
    defaultDir = getParfileDirs(PYTHON_3_INDEX)
    extras = ['/my/path/to/scripts']
    for x in extras: defaultDir.append(x)
return defaultDir
```

Here `'/my/path/to/scripts'` is where you want to keep your Python scripts for Topspin. You may add as many paths in `extras` as you wish, separated with commas. As mentioned in paragraph 3.1, `xpy3-tools` can be run as regular console applications, *i.e.* `python3.x <script name>`, or be launched from within a dedicated IDE. All you need in this case is open a target NMR dataset in Topspin and make sure that the network service is running – either through `Preferences` → `Python 3+` → `Manage TopSpin Network Interface` in Topspin 4.2+ or by executing command `start_rest_interface` when using Topspin 4.1.3. The examples below presume that Topspin 4.2+ is used.

4 Usage examples

4.1 baseline

This script allows interactive baseline correction of 1D spectra. For a baseline classification part, the script employs a function `std_distribution()` from the package `pybaselines` [2], which identifies baseline points by analyzing a rolling standard deviation distribution across a spectrum. This part is controlled by two parameters – `half_window` and `num_std` – adjusting, respectively, the number of points involved in the rolling standard deviation calculation and a number of standard deviations to be used as a baseline threshold. For a baseline interpolation part, it uses a Whittaker smoother `ws2d()` from the package `vam.whittaker` [3]. It gives the third adjustable parameter – `lambda` – to control the interpolation smoothness. Running `xpy3 baseline` in Topspin's command line (Fig. 1) brings about a two-graph window (Fig. 2). The upper graph shows the classified baseline points and baseline interpolation overlapping the original spectrum. The lower graph shows the spectrum after subtracting the estimated baseline.

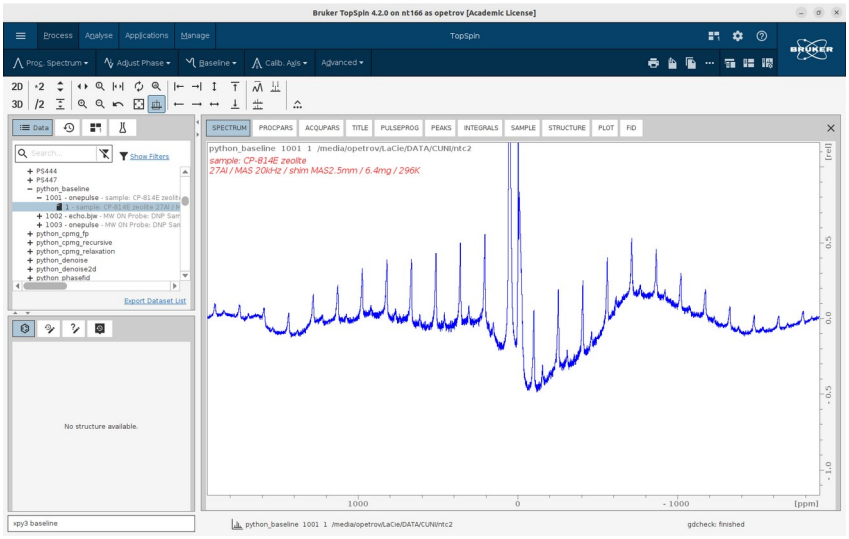


Fig. 1

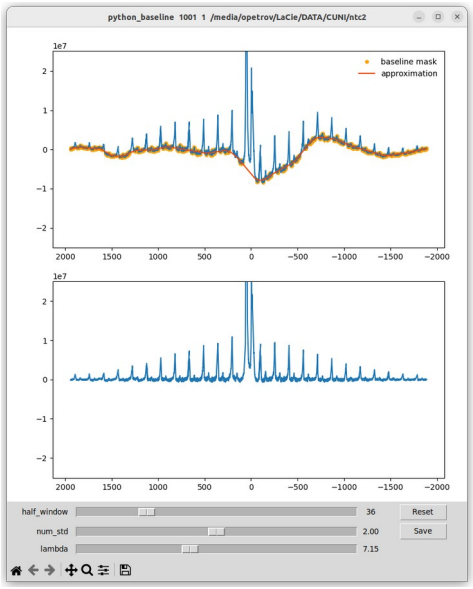


Fig. 2

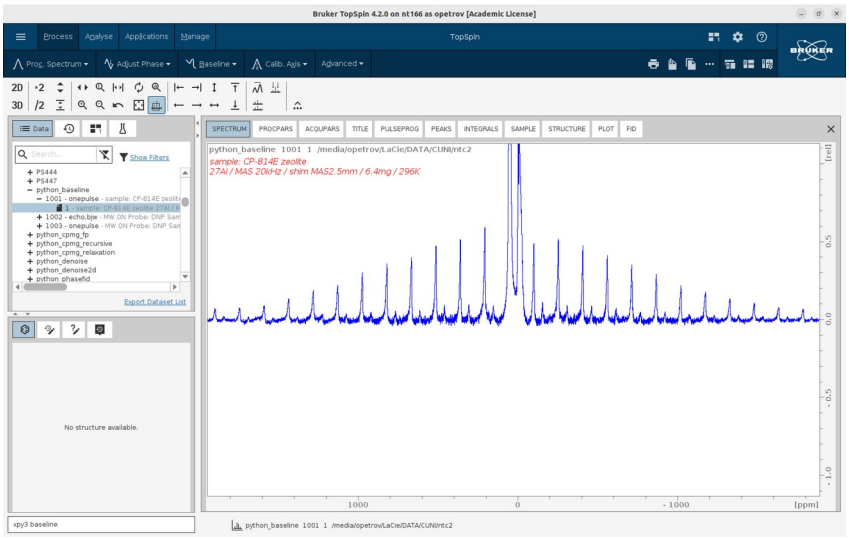


Fig. 3

The baseline estimate can be improved manually using sliders. Clicking `Reset` restores the automatically estimated baseline. Clicking `Save` writes the spectrum shown on the bottom graph to the `pdata` folder and updates Topspin's window accordingly (Fig. 3). You may run the script few times for additive corrections.

The script checks modification times of `1r` and `1i` files, and when they coincide (meaning that in-built baseline corrections have never been applied, hence Re and Im parts of processed data are still coherent), the estimated baseline is subtracted from both Re- and Im-part of the spectrum. This is handy if you plan to process the spectrum any further, and the processing involves both of the parts (*e.g.* phasing, conversion to magnitude mode, inverse FT). The script is applicable to datasets with `1i` files missing, *i.e.* to magnitude spectra.

4.2 baseline2

This script is used to correct baselines of individual spectra stored in pseudo-2D data sets – that is, in `2rr` and `2ii` files (but not in `2ir` or `2ri` files). As an example, Fig. 4 shows a series of 24 spectra from a T_1 inversion-recovery experiment on methane compressed inside MOF powder.

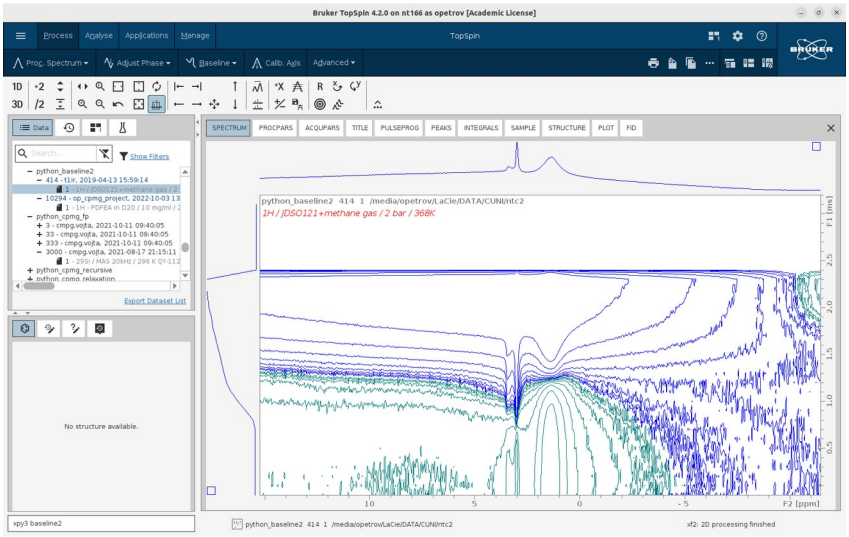


Fig. 4

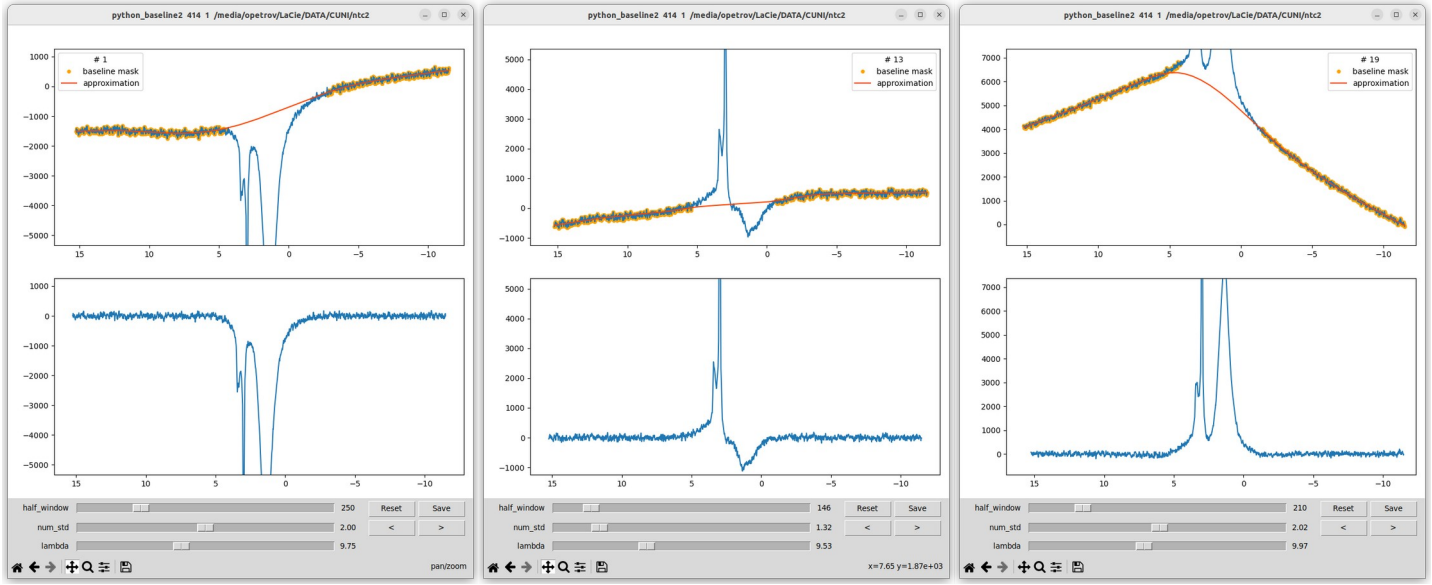


Fig. 5

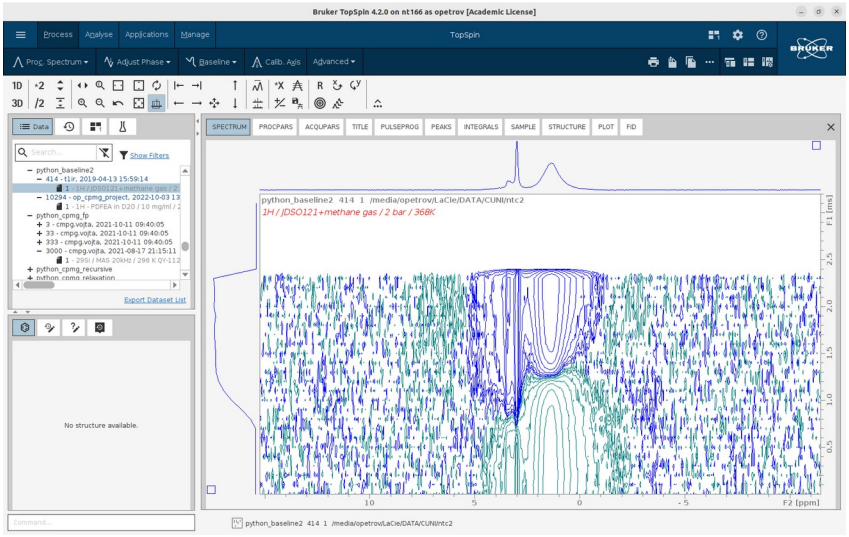


Fig. 6

Running the `xpy3 baseline2` command brings about a pop-up window which is identical to Fig. 2 but with extra buttons, ">" and "<", to navigate through the spectra. The first spectrum of a series is displayed by default. Once you inspect and optimize a baseline estimate for this spectrum, you switch to another, and so on (Fig. 5). Eventually you click `Save` to write the entire baseline-corrected series to the disk (Fig. 6).

Tip: If you find matplotlib too slow at interactive re-drawing (zooming and such), you can try and improve it by increasing a line segment simplification parameter `path.simplify_threshold` in a matplotlib's configuration file `[PYTHON]/.../matplotlib/mpl-data/matplotlibrc`.

4.3 cpmg_fp

This script is intended for `qcpmg` type pulse programs that generate continuously sampled echo decays (a.k.a. echo trains). Processing the echo train as a regular FID – *i.e.* with the command `fp` – will result in a discrete 'spikelet' spectrum. To get a conventional continuous spectrum, you need to collapse the train into a single echo (sum of echoes) and follow whole-echo processing steps, which comprise splitting the echo in the middle, swapping the halves and only then `fp`. The `cpmg_fp` script automates these particular steps; besides it does the first-order phase correction to maximize the Re part of the spectrum.

Fig. 7 shows a train of 256 spin echoes from a zeolite sample (^{29}Si NMR). Typing `xpy3 cpmg_fp` brings about a two-graph window (Fig. 8) with the top graph showing the echo train (same as in Topspin's FID window) and the bottom graph the average echo (sum of echoes) after splitting the train into individual echo segments. By default, the average echo is calculated with all segments included, which can be changed to include only a number of leading echoes using the slider `# echoes`. This will affect not only SNR but also the line shape of the spectrum provided that it has differently relaxing components. Once you are certain about `# echoes`, click `OK` to get to the spectrum (Fig. 9).

For the script to work, the echo segments in the train must be separated by at least one zero point. A pulse program `op_qcpmg` shipped with `xpy3-tools` is a variant of a standard `qcpmg` that is intended for having such zeroes in the signal.

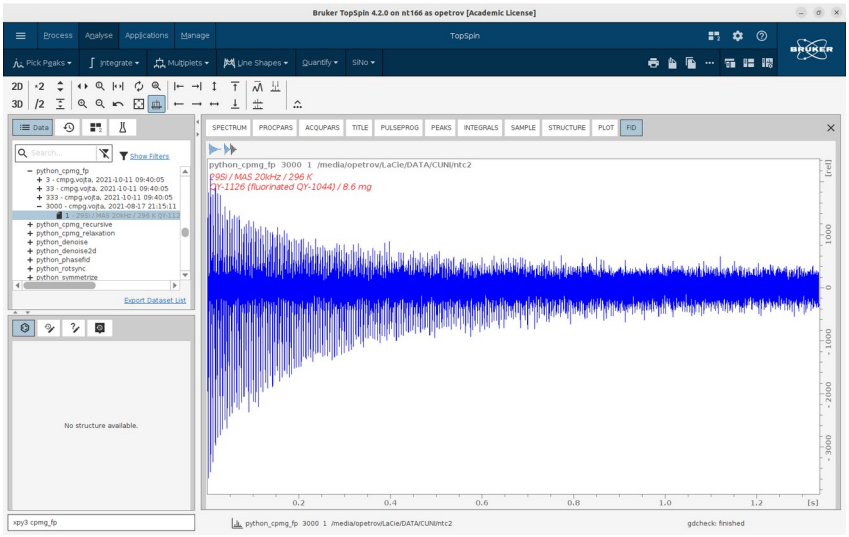


Fig. 7

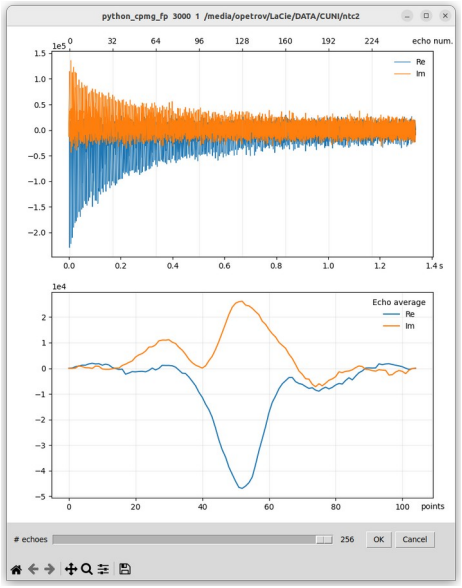


Fig. 8

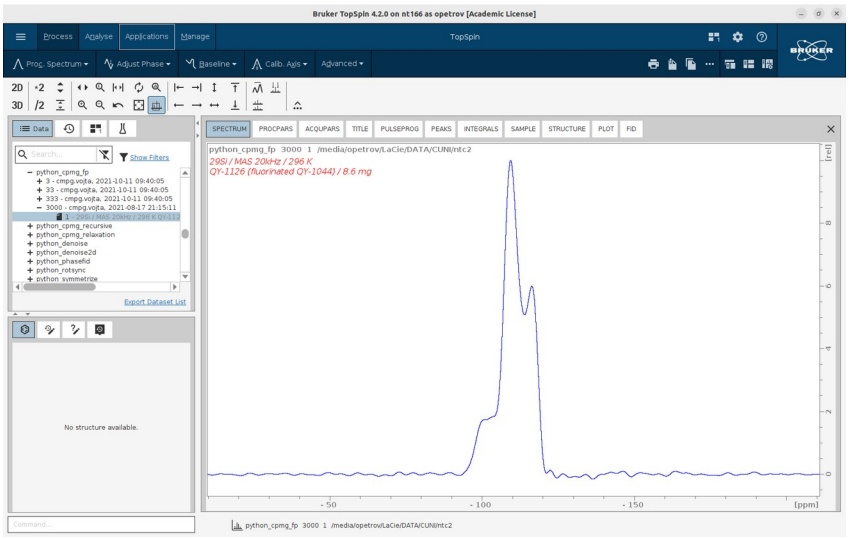


Fig. 9

4.4 cpmg_relaxation

This script is again relevant to the `qcpmg` type pulse programs generating continuous trains of zero-separated echoes. Individual echoes are quantifies using either midpoints (by default) or integral intensities. The resultant echo-decay curve – the echo signal *vs.* echo time – is fitted with either mono or stretched exponential model.

Fig. 10 shows an example echo decay for a fluorine-containing zeolite (^{19}F NMR) acquired with the pulse program `op_qcpmg` (included to the package). Running `xpy3 cpmg_relaxation` brings about a two-graph window (Fig. 11). The top graph shows the experimental signal, with detected echo positions marked with

asterisks, and the bottom graph the echo decay curve, on a semi-log scale, measured from echo midpoints (with the option to switch to integrals). Clicking `Fit Me!` fits a mono-exponential model to the curve, with the best-fit parameters reported in as annotation box (Fig. 12). For non-exponential decays, there is an option to use a stretched exponential fit, in which case a mean relaxation time is reported (Fig. 13). Clicking `Save&Quit` saves the decay curve in a two-column text file in `<EXPNO>/pdata/1/ct1t2.txt`, for further processing (*e.g.* for Inverse Laplace transformation). Note that the annotation text with best-fit model parameters is not saved – so mind that you write them down before clicking `Save&Quit`.

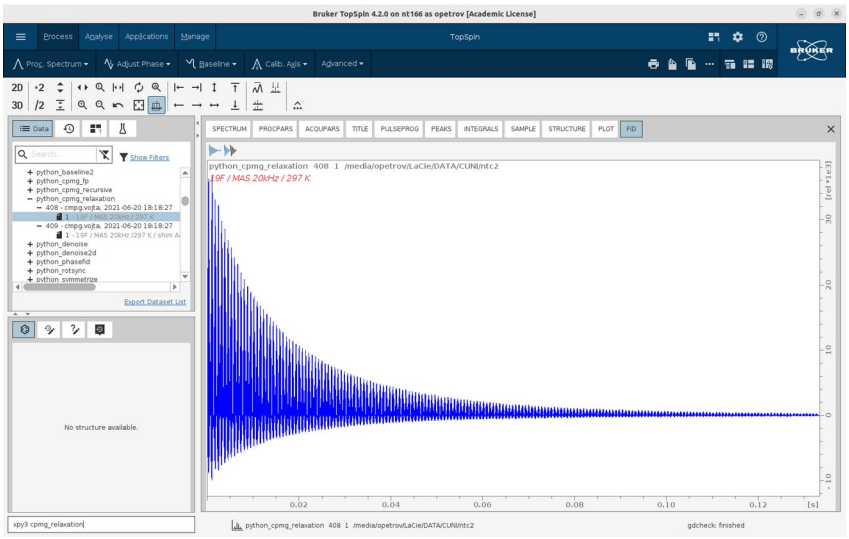


Fig. 10

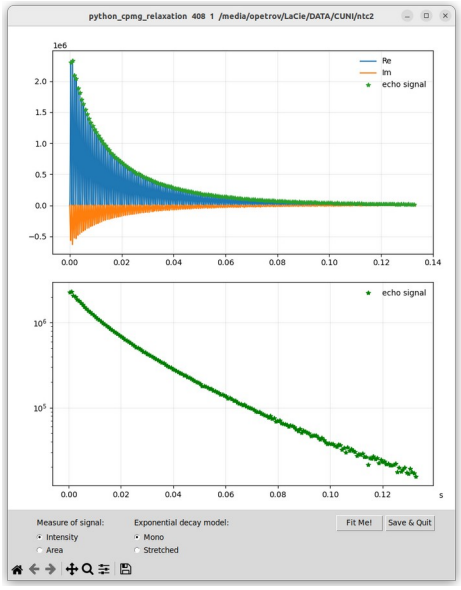


Fig. 11

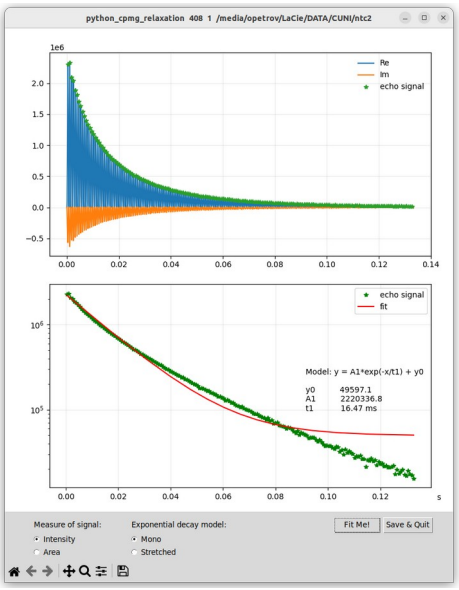


Fig. 12

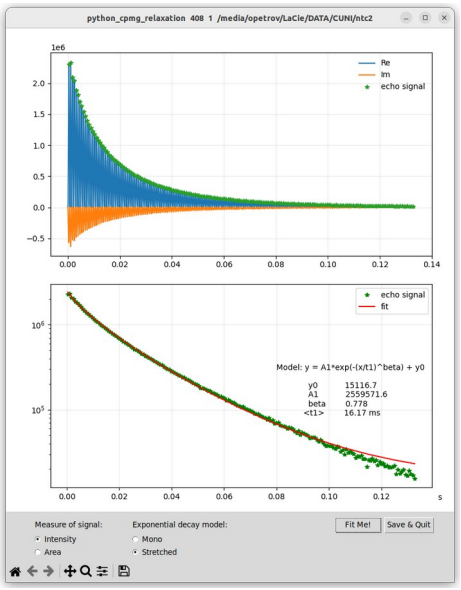


Fig. 13

4.5 cpmg_recursive

The function of `cpmg_recursive` is identical to `cpmg_relaxation` except for its using a cumulative sum of consecutive echoes intensity instead of individual echo intensity as a measure of relaxation. The cumulative

sum is fitted with a recursive mono-exponential relaxation model given in [6]. The summation decreases a scatter in points on longer relaxation times and it renders the best-fit parameters less sensitive to fast relaxation of few initial points, thereby better capturing an overall relaxation behavior. Fig. 14 shows the performance of `cpmg_recursive` on the same data as in Fig. 10. The GUI's controls are analogous to `cpmg_relaxation`.

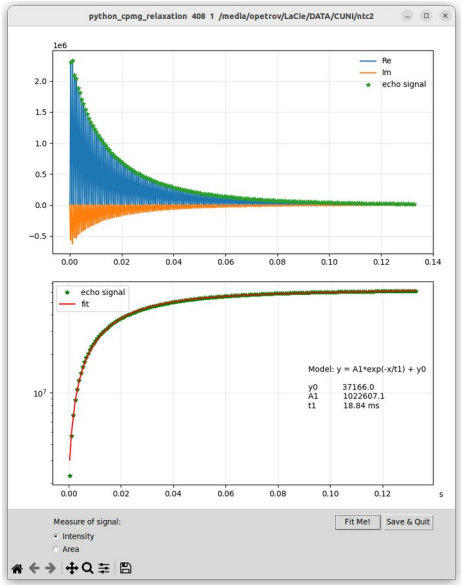


Fig. 14

4.6 denoise

This script implements a Cadzow's denoising of a 1D time-domain signal [9]. The method consists of a low-rank decomposition of a data matrix constructed from the FID – of either a Hankel or Toeplitz format – where each row is a lagged (one-point shifted) sample of FID. The data matrix is subjected to a singular value decomposition (SVD), where only a few most significant singular eigenvectors are retained. The retained terms are then employed to reconstruct (approximate) the data matrix, which is finally re-arranged back into a 1D signal. To avoid dealing with group delay's points that precede the actual FID points, the signal is automatically converted to an AMX ('analog filter') type by Topspin's `convdta`.

The script does not have a dedicated GUI. There are a couple of things that may help to optimize the performance. Firstly, you may try and truncate the FID signal by setting the Topspin's proc. parameter `TDeff` somewhere between the actual `TD` and the value at which truncation artifacts ('sinc wiggles') become visible. Secondly, you might want to change the number of eigenvectors used in data approximation. By default, this number is set automatically according to a Gavish-Donoho's (G.-D.) threshold formula [7] and can be adjusted by running the script with an external argument `offset` as

```
xpy3 denoise --offset n
```

Here `n` is an integer (either positive or negative) which will be added to the G.-D.'s threshold value.

Fig. 15 shows a ^{13}C spectrum of an organic precursor inside a zeolite sample (on the top) and the same spectrum after applying `denoise` (on the bottom), demonstrating a flawless performance of the method. Another example is a ^{13}C spectrum of alginate (Fig. 16), in which case `denoise` leaves two noisy spikes at 47 and 170

ppm. As a rule, such spikes appear ‘off-phase’ w.r.t. real peaks and therefore are readily recognizable as noise-related artifacts.

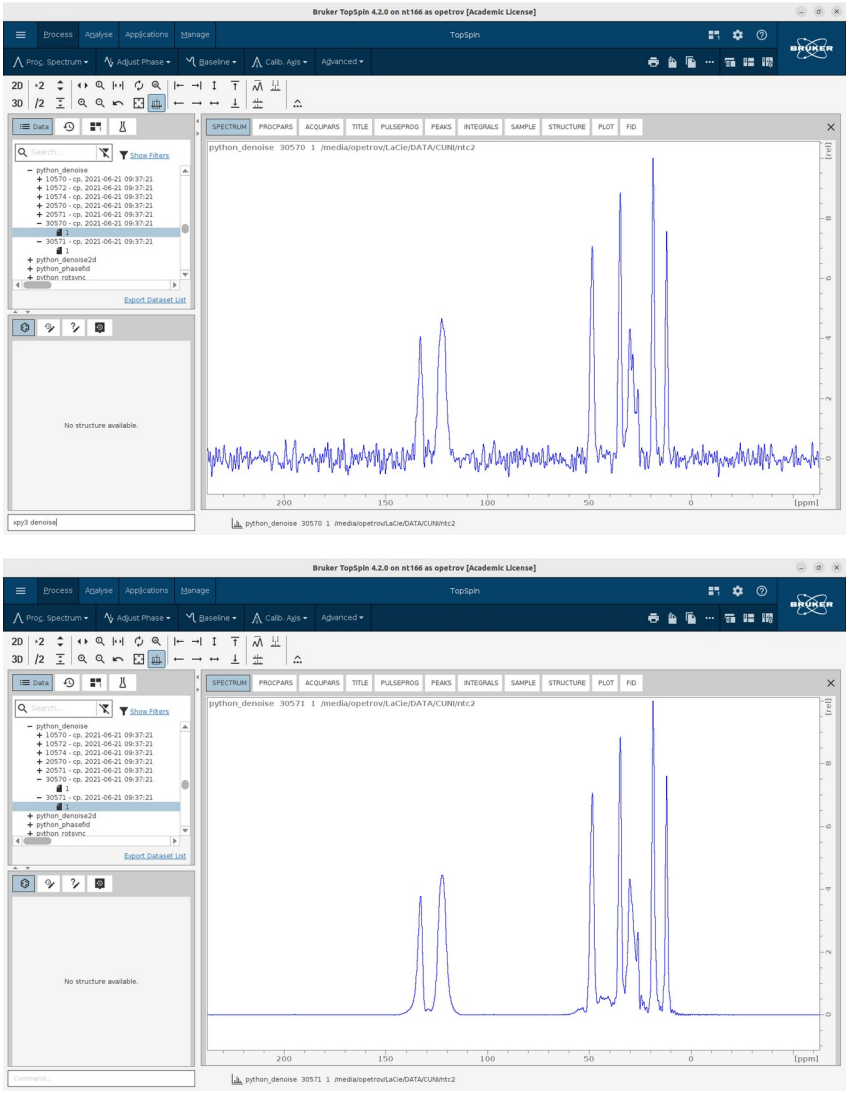


Fig. 15

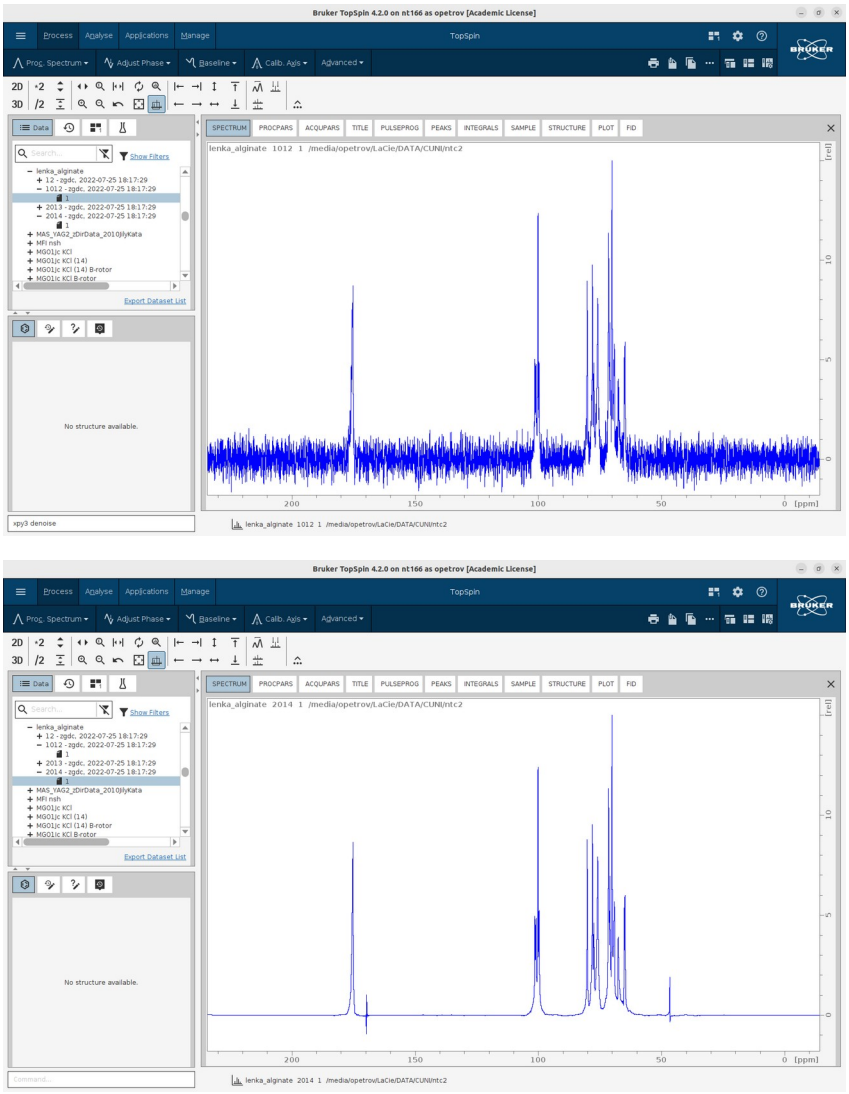


Fig. 16

4.7 denoise2

This script removes noise in 2D data using the original method described in [8]. In brief, it is a PCA-based de-noising method where the principal components (PCs) constituting a basis for data approximation are attenuated accordingly to their noise-related content. You can think of it as a regularized SVD approximation in contrast to the hard-threshold SVD approximation that is used in `denoise`. The PC attenuation is achieved through 'averaging' PCs of the same index in a repetitive SVD on randomly chosen rows of data. The number of rows is controlled by a parameter `nus` ranging from 0.25 to 1.0 and the number of repetitions by the parameter `samples`. Both parameters are set through external arguments as follows (the square brackets indicate that these arguments are optional):

```
xpy3 denoise2 [--nus nus] [--samples samples]
```

Given no arguments, the script estimates `nus` automatically and uses the default `samples` = `20`.

Fig. 17 shows a phase-sensitive 2D INADEQUATE spectrum of strychnine in $CDCl_3$ (taken from Bruker's example dataset `/examdata/exam_CMCse_3/`). Beneath it is the same spectrum but after applying `denoise2`, demonstrating a factor of 2.3 improvement in SNR.

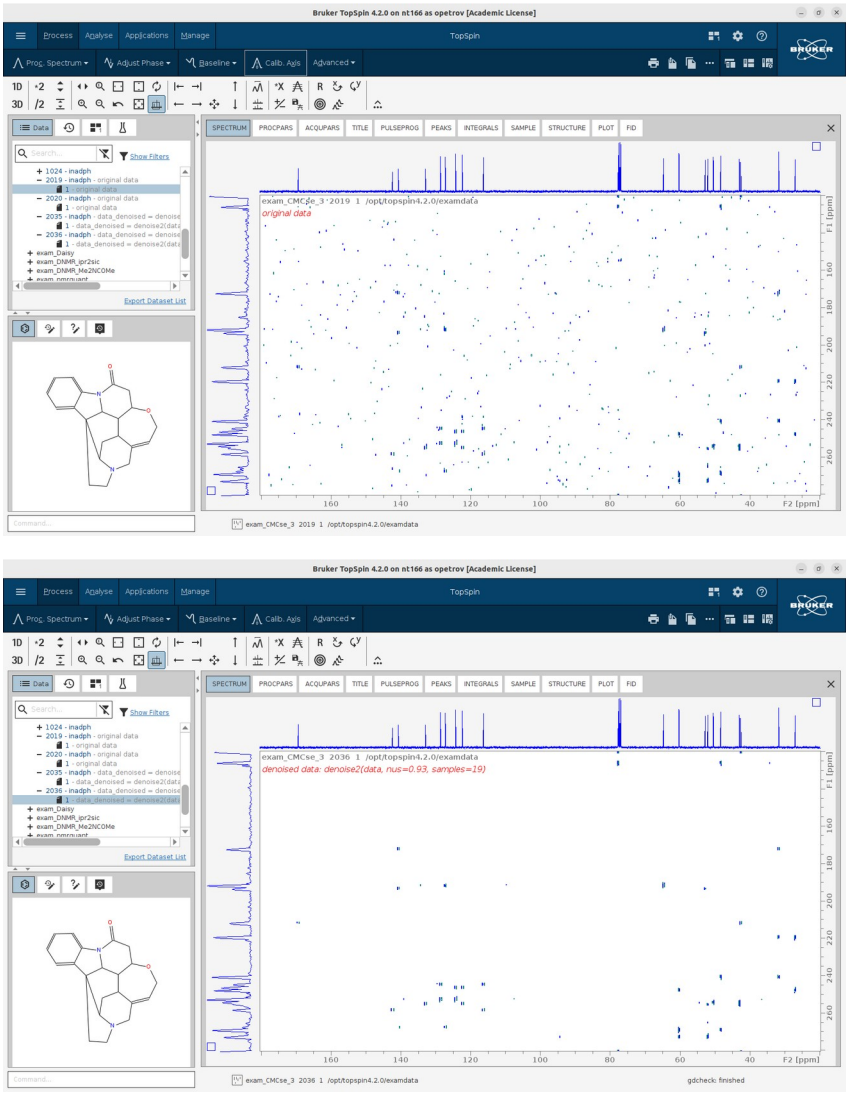


Fig. 17

Running `denoise2` on large datasets and with the number of samples greater than the default 20 may take longer than a minute, which feels annoying. Should you not want to wait that long, interrupt the script by closing the accompanying progress bar widget and restart it with less number of samples. Like its 1D counterpart `denoise`, `denoise2` copies the entire data set to a nearest available `EXPNO` as a first step.

4.8 icoshift

This is a front-end to the functions of `pyicoshift` [11] – a Python implementation of the *icoshift*, which stands for '*interval correlation optimized shifting'* algorithm for peak alignment. It is applicable to spectra with the same peak pattern where certain peak positions are prone to fluctuations in pH, ion content, temperature *etc.*, or where the whole spectrum is shifted due to instrumental factors. The script can be run either on a

pseudo-2D dataset or on 1D spectra linked to each other via multi-display (`md`) mode, provided they are of the same size.

Fig. 18 shows spectra of a polyacrylamide hydrogel in D_2O from a variable-temperature experiment, opened in a Topspin's multi-display window (`.md`). Typing `xpy3 icoshift` brings about a two-graph GUI (Fig. 19). The top graph replicates the content of the Topspin's window, thus enabling peak intervals selection. Omitting the selection means treating the whole spectrum as one interval. Note that once selected, the intervals cannot be edited, you can only clear them altogether with a `Clear` button and start over. In Figs. 20 and 21, one interval is chosen that encloses a single peak from a reference compound (TMMS). With `Align mode` options, you can choose whether to align only this selected peak, keeping the rest intact (the option `n_intervals`), or to apply the shifts which will be found for this peak to the whole spectra (the option `whole`). When `n_intervals` is chosen, the gaps in the shifted spectra are filled with adjacent values, whereas for the `whole` option, the spectra are completed with `nan`'s on either of their sides. With the radio-buttons `Target`, you choose a reference signal to which the spectra are shifted to match its position. The default target is `maxcorr` ("*the signal with the highest correlation with all input signals*" [10]), the others are `average` (arithmetic mean of spectra), `median` (median of spectra), `max` (spectrum of maximum intensity), and `average2` (uses the average target spectrum twice). For more information about the targets and which one it is better to use, see a tutorial to the Matlab version of `icoshift` [11]. In practice, you just go through all those targets and watch the difference it makes.

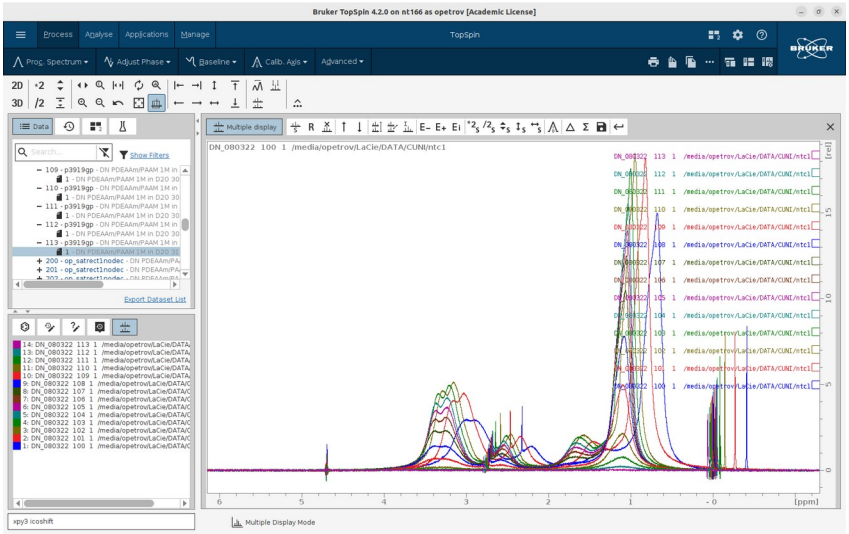


Fig. 18

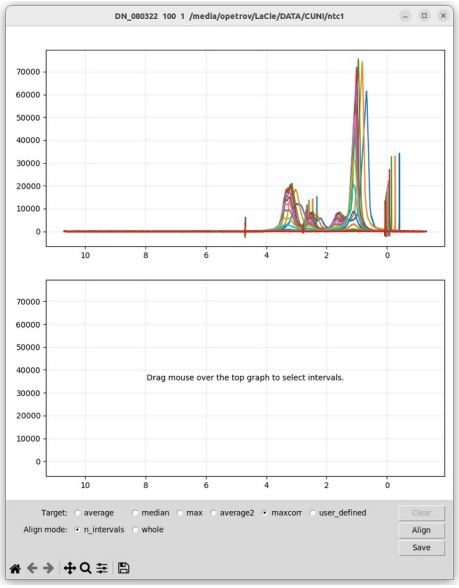


Fig. 19

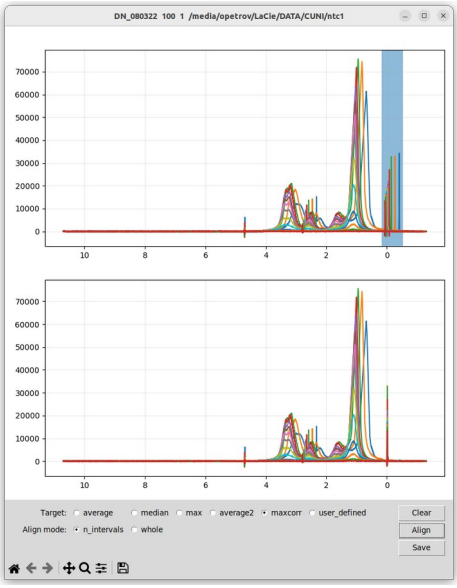


Fig. 20

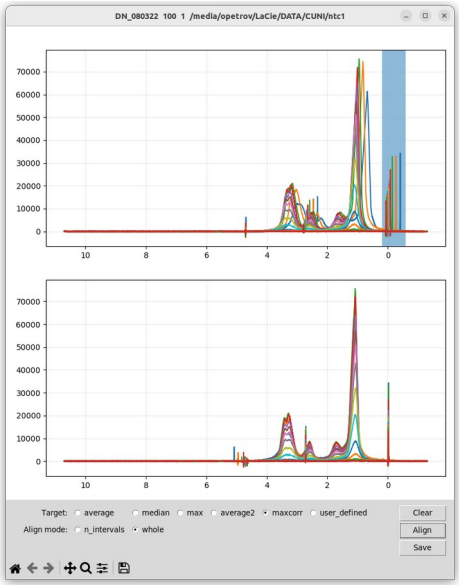


Fig. 21

4.9 lp

This plugin aims to provide an alternative method of *forward* linear prediction (LP) in 1D in case the built-in method will not work. What it does is finds principal components of a Toeplitz matrix of available data and projects them on an upper triangle matrix representing the extended part of FID with trailing zeros. The result of the projection is blended in with the available data (the diagonal averaging), then the projection/ averaging steps are repeated, until convergence. Because of the repetition, it takes longer than the built-in LP method, but in return it is free of problems inherent in recursive prediction based on LP coefficient calculation (which is presumably what the built-in LP method is about).

The script requires three parameters be specified. First is `ME_mod` – to define whether to use a plain FID (`ME_mod` = `LPfc`) or mirror-image FID (`ME_mod` = `LPmifc`) as input data. Second is `LPBIN` – to define the size of an extended FID (including original data). Both `ME_mod` and `LPBIN` are taken from Topspin's PROCPARS. The third parameter `lpblk` gives the number of simultaneously predicted points expressed as a fraction of the FID size and can be varied from 0.5 to 0.9. By default, it is set to 0.7 meaning that the FID is simultaneously extended by 70%, then by another 70% and so on until all `LPBIN` points are delivered. To adjust `lpblk`, say to 0.8, run the script with an external argument `--lpblk` followed by the value, *i.e.*

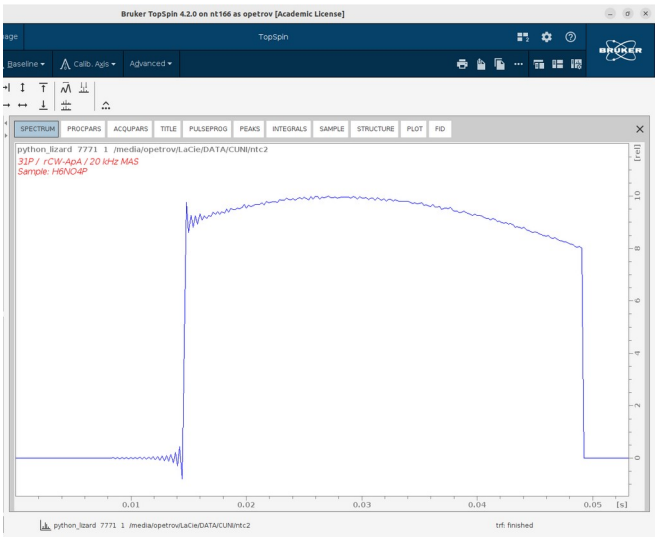
```
xpy3 lp --lpblk 0.8
```

Tips for `lpblk` and `ME_mod`: (i) Low `lpblk` (< 0.7) readily lead to unstable results, meaning that the calculated FID values skyrocket to several orders of magnitude instead of approaching zero. When this happens, increase `lpblk`. (ii) Apparently, `lp` with `ME_mod` = `LPmifc` is more stable than with `LPfc`. However, the former requires a specific FID structure to work (hard to tell which one exactly) while `LPfc` suits them all.

Topspin doesn't have a dedicated command for LP. To see the result of LP application in time domain, one has to disable FT by setting `FT_mod` = `no` and run a `trf` command which would knows, via `ME_mod`, that LP had been requested. The extrapolated FID is saved in this case in `~/pdata/.../1r` and `1i` files. In contrast, `lp`

saves the extrapolated FID to a `fid` file modifying relevant parameters – `TD`, `AQ` and `FIDRES` – according to its new size. Like all scripts aiming at FID modification, `lp` writes in `fid` after first copying the entire data set under a new EXPNO. Should another run of `lp` be necessary, don't forget switching back to a window containing the initial data set. Once you find the extended FID adequate, proceed to FT normally.

Fig. 22 shows a ^{31}P FID of H_6NO_4P from a CP MAS experiment with proton decoupling, which allowed max. 50 ms acquisition time to avoid problems with applying high-power decoupling pulses. (This is probably the only reason that FID is truncated during acquisition nowadays.) Consequently, FT has severe truncation artifacts. Figs. 23 shows this FID extrapolated up to 400 ms employing `lp` with `ME_mod=LPfc`, `LPBIN=4096`, and `lpblk=0.9`, as well as a new spectrum. For another example, Fig. 23 shows a truncated ^{1}H FID from a gelatin derivative in H_2O and the FID extrapolation with `ME_mod=LPmifc`, `LPBIN=11776`, and `lpblk=0.6` (only a part of the spectrum is shown).



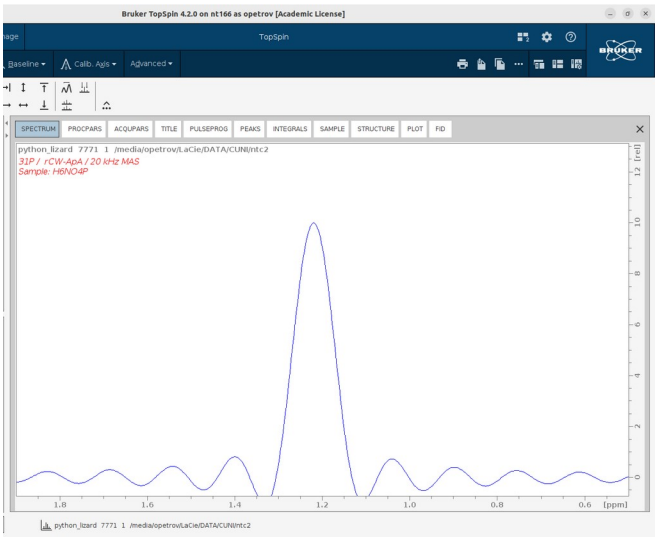


Fig. 22a

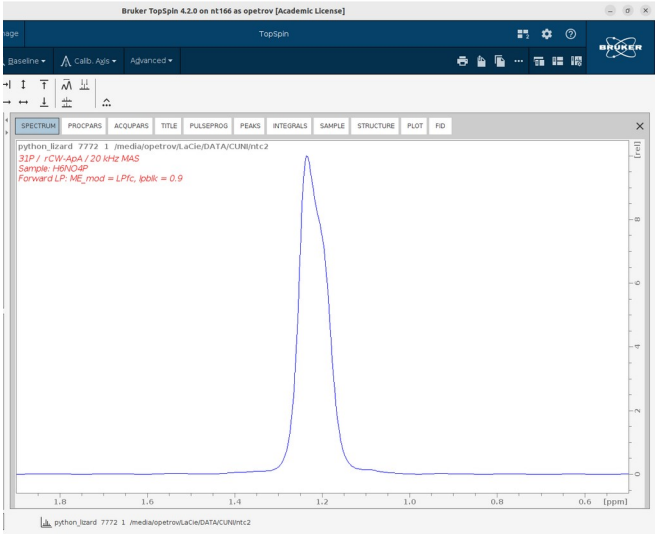


Fig. 22b

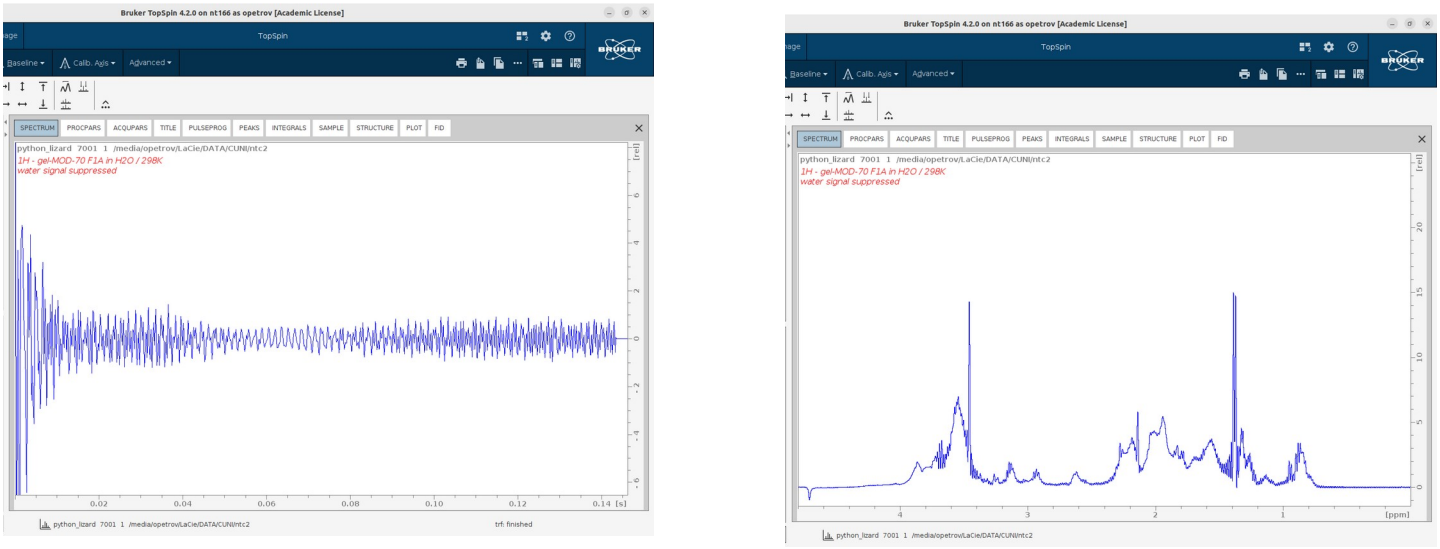


Fig. 23a

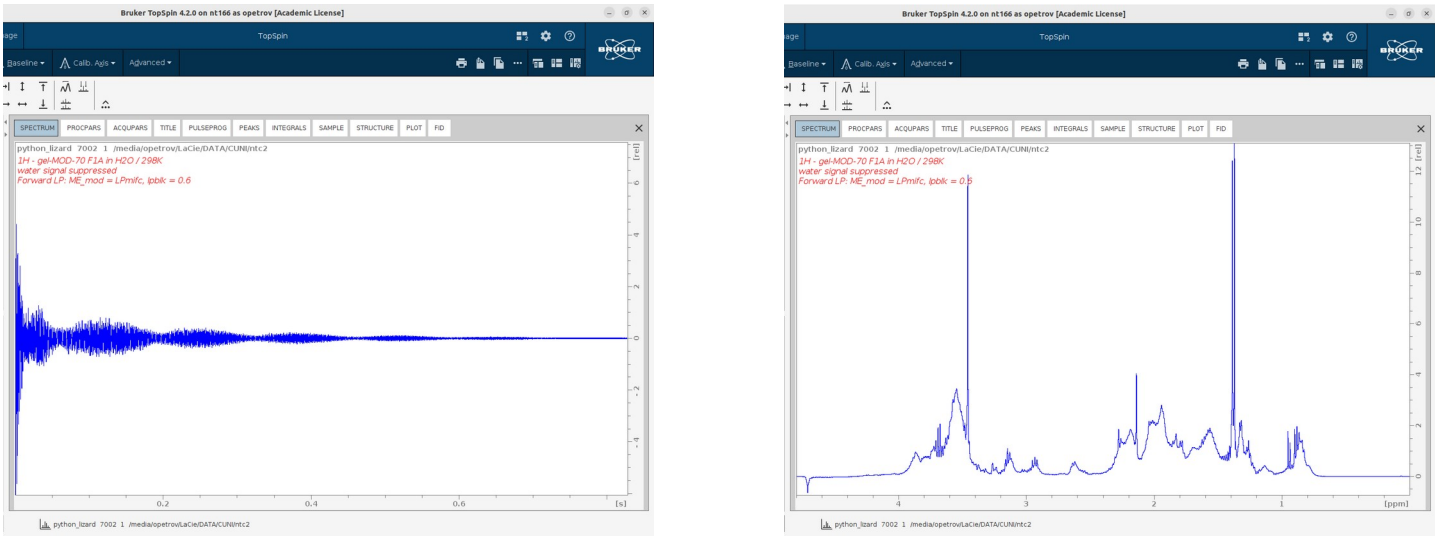


Fig. 23b

4.10 phase_fid

This script implements zero-order phasing (rotation) of a 1D time-domain signal, the first point of the signal being used as a pivot. It is primarily intended for relaxometry where such a rotation may be needed to maximize the Re-part of the signal (for quantitation purposes).

Figs. 25 shows rotation of a CPMG echo train from Fig. 24 in order to maximize the real part of the echoes. Like other scripts aiming to modify time-domain data, `phase_fid` saves the rotated signal under a new `EXPNO` (Fig. 26). The option `Zero Im` allows one to nullify the imaginary part of the rotated signal, which is a handy means to obtain a symmetric Fourier transform ans which is often used in ^{2}H line shape analysis. The value of the `PH_ref` slider may be written down and added to Topspin's PROCPARS for the next experiment.

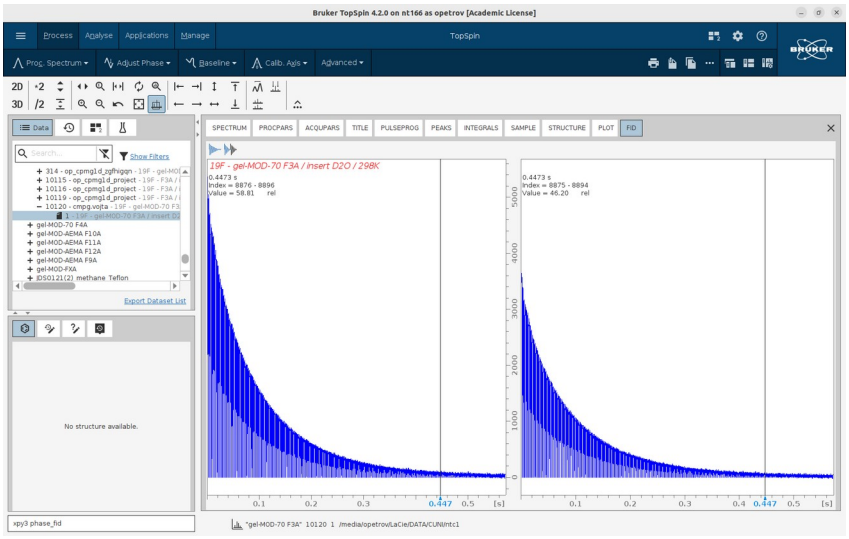


Fig. 24

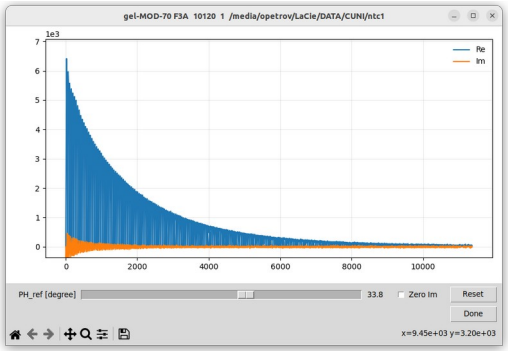


Fig. 25

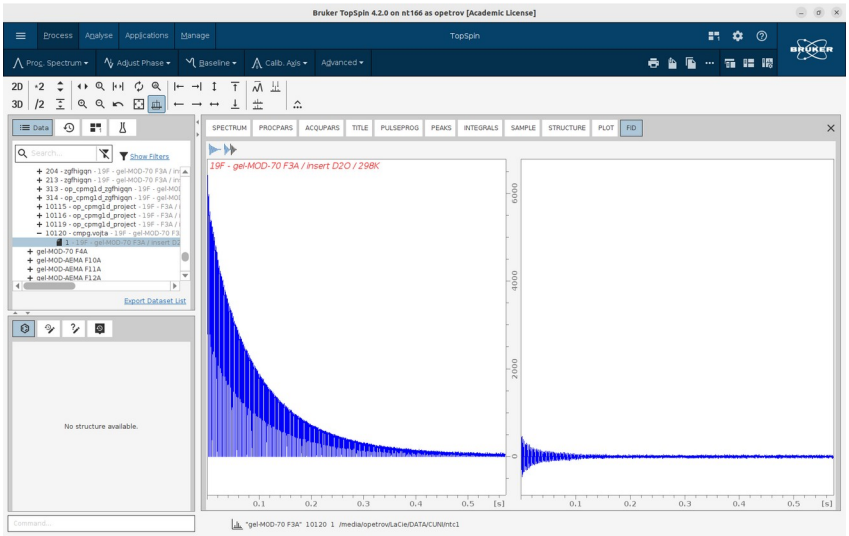


Fig. 26

4.11 rotsync

This tool is relevant to the magic angle spinning (MAS) NMR spectroscopy. What it does is positions rotary echoes in a MAS FID signal and re-samples it down to those positions only, thereby mimicking a rotor-synchronized FID acquisition. The corresponding FT spectrum should not have spinning sidebands (ssb), as all satellite transitions fall into one centerband where they superimpose the central transition signal. Therefore, when it comes to quantitation of different chemical shift components of the spectra, it can be done through direct integration. The line shape of the rotor-synchronized spectrum is equivalent to an infinite MAS rate spectrum, which is handy for line shape analysis.

Usually, the rotor synchronization is implemented on the fly during an actual signal acquisition. However it requires thorough adjustment of both the sampling period and the sampling offset as the acquisition parameters. If you find such an adjustment tedious and difficult to control, use this post-acquisition synchronization instead. Of course, it will only work if the rotary echoes are well defined (detectable), at least over a few rotor periods.

Fig. 27 shows the centerband of a MAS spectrum of $Al(acac)_3$ acquired under a 20 kHz MAS (^{27}Al NMR). Running `xpy3 rotsync` brings about a window (Fig. 28) with the given FID, in the magnitude mode, with rotary echo positions marked with asterisks. If needed, you can adjust these positions using the sliders `offset` and `period`. Clicking `OK` saves the echo positions as a new FID signal under nearest available `EXPNO` and process it with `efp` command (Fig. 29).

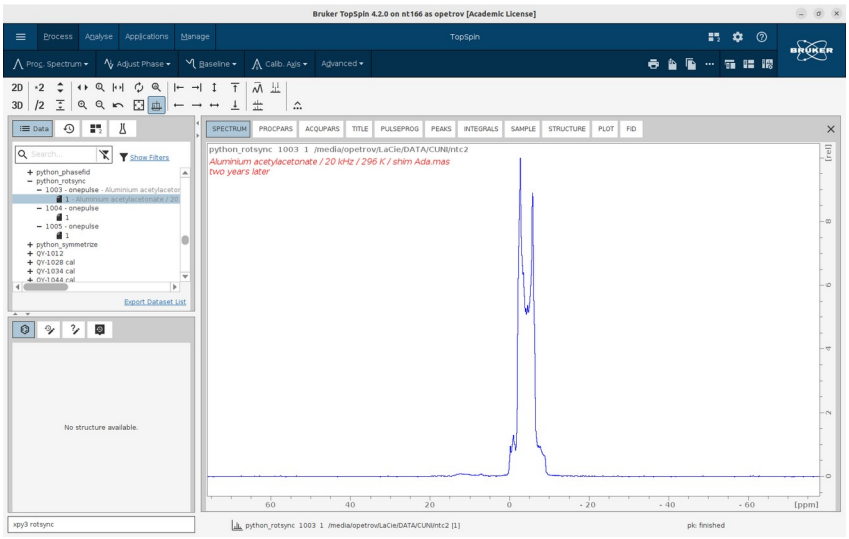


Fig. 27

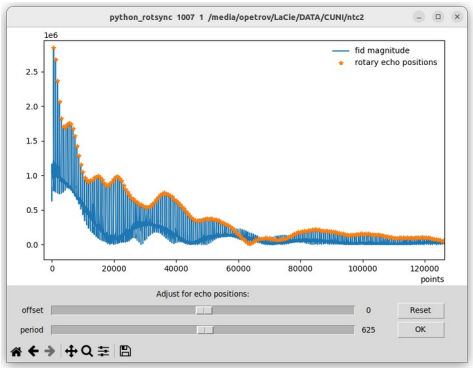


Fig. 28

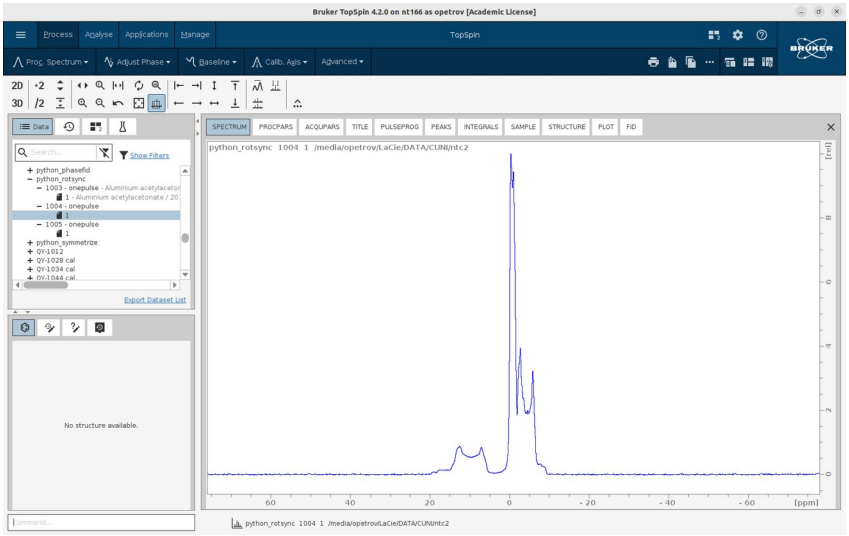


Fig. 29

4.12 rotsync_plus

Identical to `rotsync` but with a rather involved echo quantitation procedure, aiming to better SNR.

4.13 sino_fid

This is a variant of Topspin's `sino` but for the time domain. It calculates the ratio of the highest intensity in the FID to the noise level. The latter is defined as 2 times standard deviation of noise, in accord to the Topspin's

definition of SNR. The noise component is extracted from the FID through principal component analysis, the procedure being similar to that used in `denoise`. It practically means that there is no need to specify pure noise region to estimate the noise std. The SNR value is reported in a pop-up message.

4.14 symmetrize

This, again, is relevant to quadrupolar MAS spectra of half-integer spins. It is often that ssb manifolds in such spectra have asymmetric pattern with respect to the central band, even though the first-order quadrupolar interactions prevail, meaning good symmetry. Observed distributions of the ssb intensity can be skewed both positively and negatively, suggesting contribution of several competing factors – magic angle misadjustment, uneven probe response as a function of frequency, a noneligible second-order quadrupolar interaction, etc. Fig. 30 is an example of a positively skewed ssb distribution (^{27}Al NMR). If you are really interested in having it symmetric, *e.g.* to see how ssb manifolds from different samples compare, or to fit the first-order quadrupolar model to ssb, or simply present the spectrum at its best (*i.e.*, without instrument factors engaged), then use this script.

Typing `xpy3 symmetrize` brings about a two-graph window (Fig. 31). On the top graph you can see the original spectra with ssb's marked with asterisks. In red is shown the recognized centerband (cb) area that is kept intact upon symmetrization. You can re-adjust the cb limits with the sliders. The cb recognition relies on a Topspin's auto-integration method (see `int auto`), specifically on the integral regions listed in the `intrng` file. The bottom graph shows a symmetrized spectrum with opposite ssb's equalized using their average intensity. (The averaging is hard-coded in line 105 of the script for `mode='mean'`. If needed, it can be changed for `mode='maximum'` in which case the ssb of a greater intensity will be placed on either side of the spectrum.) Clicking `OK` will return the symmetrized spectrum to a Topspin's window (Fig. 32).

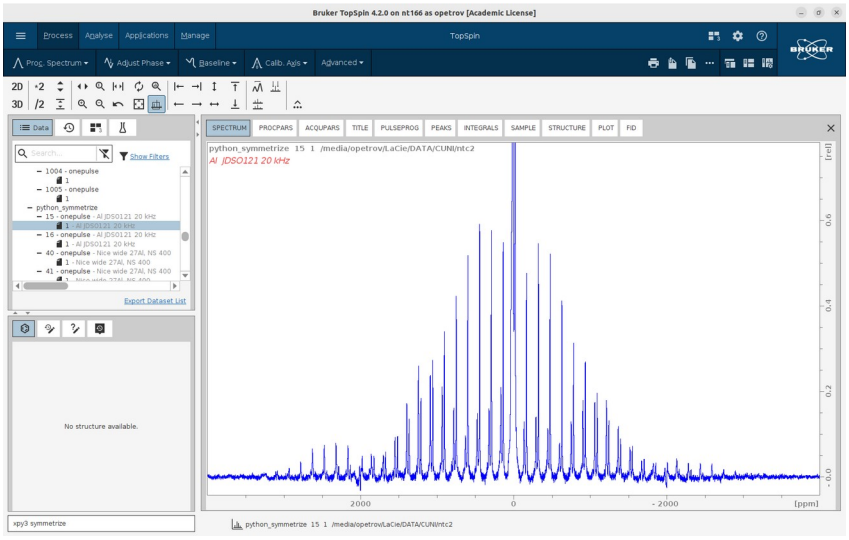


Fig. 30

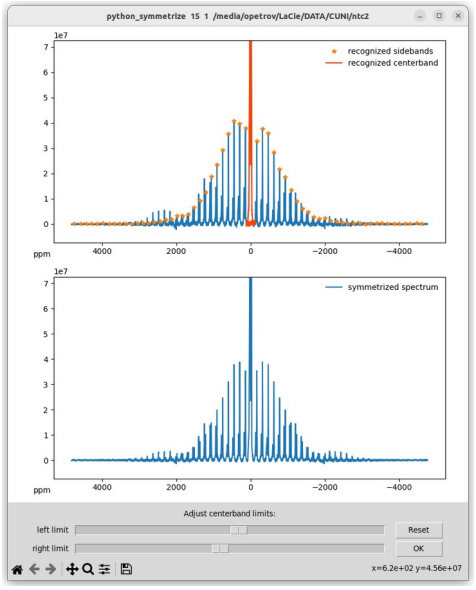


Fig. 31

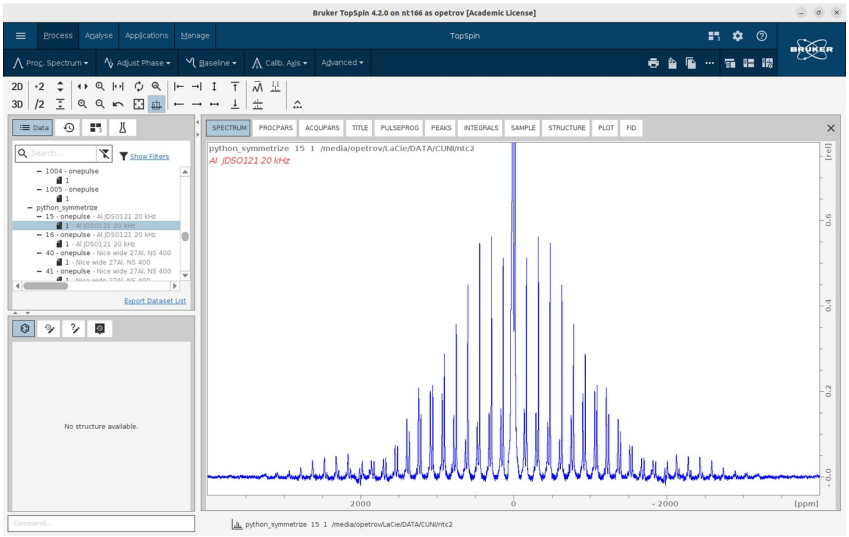


Fig. 32

4.15 t1t2

As the name suggests, `t1t2` is similar in functionality to Topspin's T1T2 relaxation toolbox. What it does is integrates a series of spectra measured at varying delays (pseudo-2D dataset) over an interactively selected region and fits an exponential model to the integral *vs* delay plot. Unlike the Topspin's T1T2, it works only with processed data, not with the FIDs. Moreover, it doesn't seek the biggest peak in the integration region, plotting only the integrals. The relaxation models used are mono-, two-, and stretched exponential ones. Relaxation delay values are read from a VDLIST file, after first being sorted in ascending order. If a VDLIST file is not found, the integrals are plotted against their indices, in which case the fitting functionality is disabled.

The script has an option for signal denoising through SVD thresholding, which is supposed to decrease a scatter in integrals. For spectra with non-zero baselines, there is an option to automatically subtract a baseline shift contribution from the integrals. (Apparently, this option needs more testing. Should it fail, use `baseline2` instead to correct baselines in the first place.)

Fig. 33 shows a two-graph GUI popped up after typing in a command `xpy3 t1t2` for the dataset shown in Fig. 4. These are data from a T_1 inversion-recovery experiment with 24 delays. The top graph displays superimposed spectra where the user can interactively select the desired region. After selection, a T_1 relaxation curve shows up on the lower graph (Fig. 34), with a live updating upon new selections. In this example, the intrinsic mono-exponential relaxation of the selected peak is obscured by a non-zero baseline shift. You can fix it checking a `Baseline corrector` checkbox, then fit a mono-exponential model normally via `Fit Me!` button (Fig. 35).

Clicking `Save&Quit` saves both the experimental and best-fit curves in a `<EXPNO>/pdata/1/ct1t2.001` text file. Note that the annotation text with best-fit model parameters is not saved, so mind writing down the needed values before quitting.

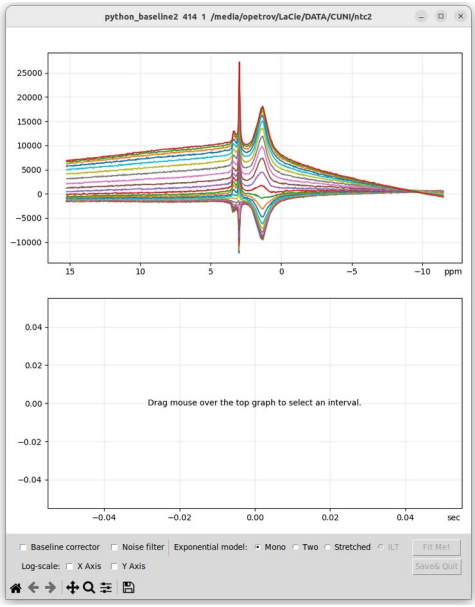


Fig. 33

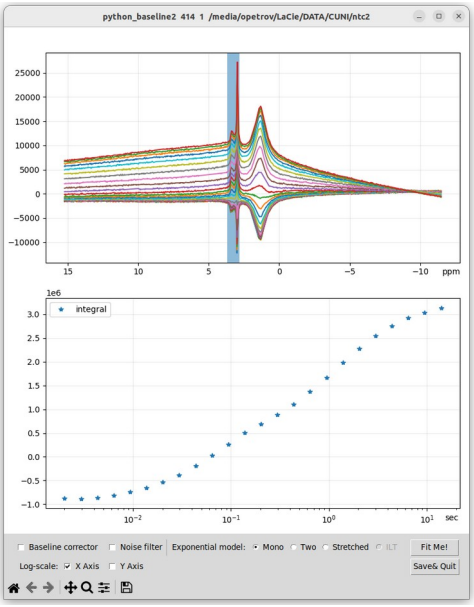


Fig. 34

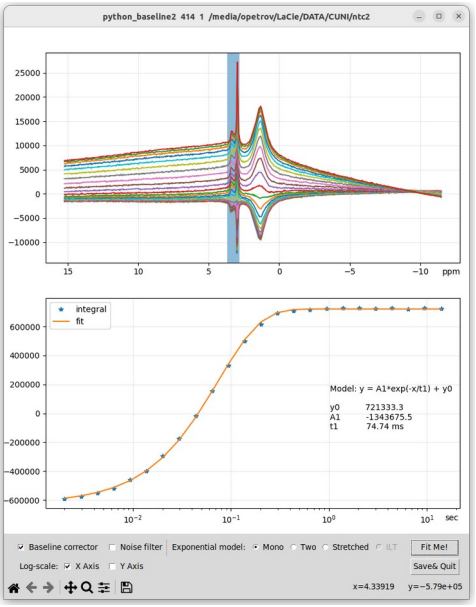


Fig. 35

5 Final remarks

The Python API introduced in Topspin 4.1.3 [1] is a tempting option for those who write their own Python scripts for NMR data processing. The `xpy3-tools` is a small collection of such scripts – formerly standalone, turned into Topspin plugins, decorated in one style. The tasks they solve may seem too diverse, or rather too specific, making impression that `xpy3-tools` will not find its use. But then conventional AU programs and Jython scripts from Topspin's library are diverse too, yet they are used by many. Anyway, `xpy3-tools` are gradually updated, with new scripts being added. Those who tend to write their own scripts might consider it a template / building blocks / complement to their own efforts.

6 References

[1] https://www.bruker.com/en/products-and-solutions/mr/nmr-software/topspin/topspin-python-interface.html

[2] https://github.com/derb12/pybaselines

[3] https://github.com/WFP-VAM/vam.whittaker/

[4] Wang, K.C. et al. Distribution-Based Classification Method for Baseline Correction of Metabolomic 1D Proton Nuclear Magnetic Resonance Spectra, *Analytical Chemistry* **85** (2013) 1231-1239.

[5] Eilers, P.H. A perfect smoother, *Anal. Chem.* **75** (2003) 3631–3636.

[6] Kolyagin, Y.G. *J. Phys. Chem. Lett.* **7** (2016), 1249–1253

[7] Gavish, M. and Donoho D. L. The Optimal Hard Threshold for Singular Values is 4/√3, *IEEE Trans. Inf. Theory* **60** (2014) 5040–5053

[8] Petrov, O.V., The Use of Self-Adaptive Principal Components in PCA-based Denoising, *J. Magn. Reson.* **371** (2025) 107824

[9] Laurent, G. et. al, Denoising Applied to Spectroscopies – Part I: Concept, *Appl. Spectrosc. Rev.* **54** (2019) 602–630

[10] https://github.com/sekro/pyicoshift

[11] https://doi.org/10.1039/9781849737531-00014