The xpy3-tools tutorial

Oleg V. Petrov | KFNT MFF | Univerzita Karlova

Version 1.1

13th December 2024

Table of Contents

1 Introduction

`xpy3-tools` is a collection of Python scripts for NMR data processing intended to be used as command line tools in Bruker Topspin software. As such, they rely on the Bruker Python API available since the Topspin version 4.1 [1]. The scripts have primarily targeted the author's needs and the scope of his own practice, but hopefully might be of interest to other NMR practitioners.The present version (v1.1) comprises 12 scripts (Table 1), which number is supposed to be extended in newer versions, as time goes on and the tasks tend to grow.

The scripts are run by typing the command `xpy3 <script name>` in the Topspin command line. In this respect, they are similar to conventional Jython scripts and Bruker AU programs. Sharing a workspace with Topspin means, among other things, an access to the Topspin's GUI. Yet most of `xpy3-tools` have their own GUIs popping up in a separate windows, – for interactive parameter optimization and visual control of the result before returning to Topspin. The GUI part is implemented using `matplotlib` / `Tkinter` functionality.

Input / output (I/O) functionality is provided by the Python module `BrukerIO` from the package `JTutils` (https://github.com/jtrebosc/JTutils), as I/O functions available in the Bruker Python API are still very basic. `xpy3-tools` do not modify in place raw data from `fid`, `ser`, `acqu(123)`, etc. files. If the raw data (time-domain signals) are programmed to change, a given data set is copied as a whole under a new `EXPNO` thus keeping the original data intact. Otherwise the data are processed in-place, the files with processed data being saved in `pdata` under the `EXPNO` from whence the script has been run.

2 List of scripts

Script name	*Function*	*Dependencies**	*Comments*
`baseline`	Baseline correction in 1D	pybaselines [2] vam.whittaker [3]	Uses rolling standard deviation distribution method to identify baseline points [4] and Whittaker method of interpolation [5].
`baseline2`	`baseline` applied to individual spectra in a pseudo-2D data set.	pybaselines [2] vam.whittaker [3]	Enables navigation across 2D dataset for baseline adjustment of individual spectra.
`cpmg_fp`	Whole-echo processing for CPMG echo trains.	pybaselines [2]	Relies on zero-point separators in a CPMG signal when splitting into echoes. To get such zeroes, consider using a `op_qcpmg` pulse program (see below).
`cpmg_recursive`	T_2 relaxation time measurement on CPMG data with mono-exponential recursive model.	pybaselines [2]	Quantifies individual echoes, plots their cumulative sum vs time and fits it with a recursive relaxation model [6].
`cpmg_relaxation`	T_2 relaxation time measurement on CPMG data using mono- or stretched-exponential model.	pybaselines [2]	Plot the individual echo intensity vs echo time and fits the curve with either mono- or stretched-exponential function.
`denoise`	Low-rank SVD approximation of 1D signals in time domain.		Uses a Toepitz matrix for a 2D representation of a signal and the SVD thresholding method by Gavish & Donoho [7].
`denoise2`	Approximation of 2D signals in time domain using noise-adaptive principal component analysis		Applies NUS to the acquired data to randomize their principal components in order to average out their noise-related content prior to data reconstruction [8].
`icoshift`	Peak alignment over a series of spectra stacked in a pseudo-2D dataset or a multi-display mode.	pybaselines [2] pyicoshift [9]	A number of peak regions can be selected for alignment, or you can choose one peak to shift the whole spectrum.
`phase_fid`	Zero-order phasing of 1D signals in time domain.		Useful to maximize the Re part of an FID or echo train, *e.g.* for a quantitation purpose.
`rotsync`	Reconstruction of 1D rotor-syncronized MAS spectra from arbitrarily sampled FIDs.		Useful in quantitation through integration of chemical shift components; facilitates the line shape analysis.

`rotsync_plus`	Includes a rather involved echo quantitation procedure compared to `rotsync`.	vam.whittaker [3]	In theory, provides better SNR than `rotsync`.
`symmetrize`	MAS sideband symmetrization for 1D quadrupolar spectra of half-integer spins.		Facilitates fitting of first-order quadrupolar model to MAS spinning sidebands; good for presentations.
`utils`	Contains common utility functions and classes.	contourpy [10]	Ever growing collection of `def`s.

*) Specific for given scripts; for all dependencies see `requirements.txt`.

3 Installation

3.1 Python environment setup

Topspin 4.2+ comes with a preinstalled Python 3 environment in `<topspin>/python/lib/python3.x/`, which includes Bruker API and a few packages used in example scripts. To use a Python environment running outside Topspin, you should specify the path to the chosen Python 3+ interpreter on the Topspin preferences page as `Preferences → Python 3+ → Select Python 3+ Environment`.
Imagine you have chosen to use a Python 3.x interpreter installed, *e.g.*, in `/usr/local/bin` on your Linux machine. Imagine also that the package installer for Python is `pip`, namely `pip3.x`. In the Linux command line, navigate to the Python interpreter folder, *e.g.* to `/usr/local/bin`, then run the command

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

to install the required Python packages. The configuration file `requirements.txt` is taken from the `xpy3-tools` project directory. The `.whl` files are taken from `<topspin>/python/examples` or, should you use Topspin 4.1, you can download these files from bruker.com. The procedure should work on a Windows machine too (not tested though).

A special care must be taken of the module `BrukerIO`. The package `JTutils` containing this module cannot be installed via `pip`, hence not included in `requirements.txt`. To work it out, copy the file `brukerIO.py` directly from the `JTutils` homepage at https://github.com/jtrebosc/JTutils/blob/master/CpyLib/ and place it in the folder where `xpy3-tools` scripts are kept (see next paragraph).

3.2 Placement of scripts

The syntax `xpy3 <script-name>` to run `xpy3-tools` scripts from the Topspin command line implies a proper script location. A search path for Python 3 scripts is hard-coded to `<topspin>/python/examples`. Therefore, the simplest way to make it work is to place `xpy3-tools` scripts in that folder. Should you prefer another location, add a path. 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)
```

Then add the following code below that line (mind indents):

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

You may add as many paths in `extras` as needed, separated by a comma. In fact, `xpy3-tools` scripts can be run from a console command line as a usual Python program (`python3.x <script name>`), or be launched from within a dedicated IDE such as Spider or VSCode. All you need is having the target NMR dataset opened in a Topspin window and not forgetting to run the embedded web service [1].

4 Usage examples

4.1 baseline

This script implements interactive baseline correction in 1D. For baseline classification, it employs a function `std_distribution()` from the package `pybaselines` [2]. The function identifies baseline points by analyzing a rolling standard deviation distribution across a spectrum. It takes two parameters – `half_window` and `num_std` – to define, respectively, the number of points involved in the rolling standard deviation calculation and the number of standard deviations for baseline points thresholding. The baseline interpolation is implemented by means of a Whittaker smoother `ws2d()` from the package `vam.whittaker` [3]. This adds the third adjustable parameter – `lambda` – to control the interpolation smoothness. Running `xpy3 baseline` in the command line (Fig. 1) brings about a two-graph window (Fig. 2), where the upper graph shows the classified baseline points and baseline interpolation overlapping the original spectrum. The lower graph shows the spectrum after subtracting the baseline thus estimated.

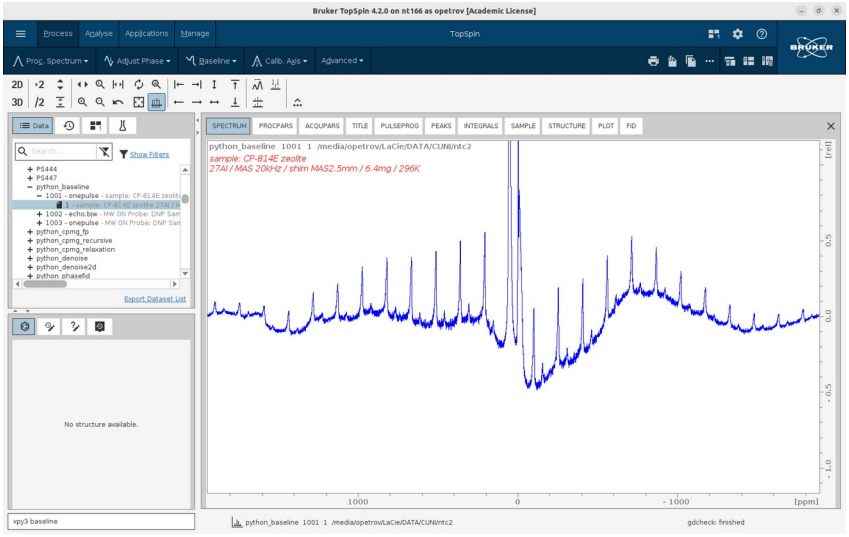

Fig. 1

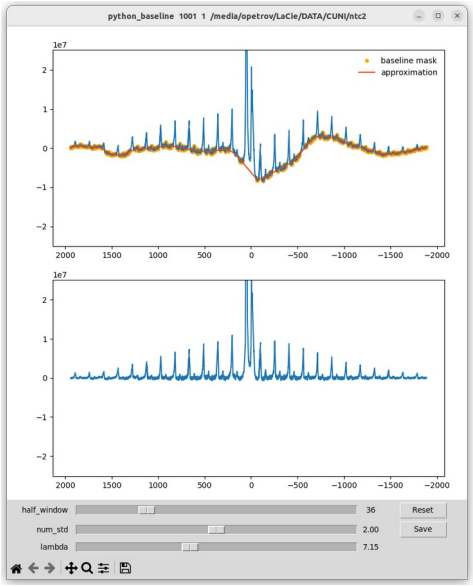

Fig. 2

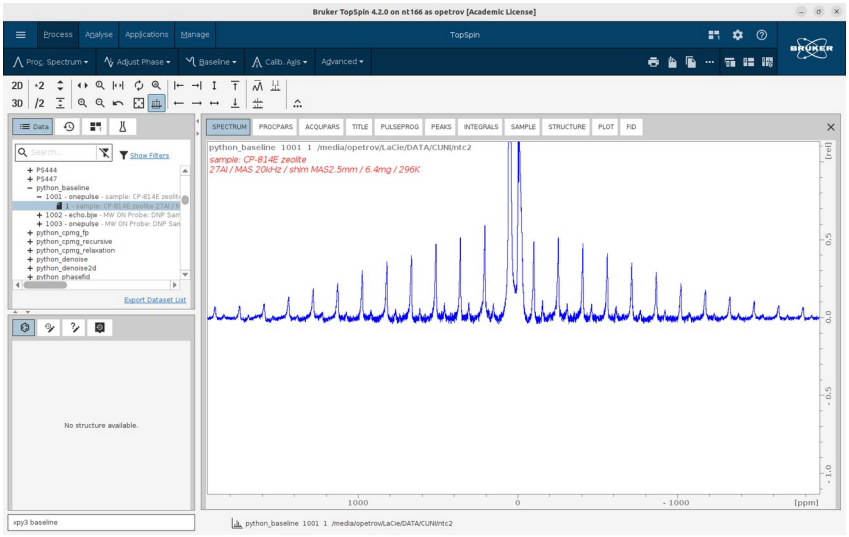

Fig. 3

You can try and improve the baseline estimate manually using sliders. Clicking `Reset` will restore the previous estimate. Clicking `Save` writes the spectrum shown on the bottom graph to the `pdata` folder and updates the Topspin window accordingly (Fig. 3). To exit without updating, close the GUI with ☒ or `Alt+F4`.

The baseline correction is applied to both Re and Im parts of the spectrum, provided that `1r` and `1i` file modification times coincide. It might be useful if you want to process the spectrum any further using both these parts (phase adjustment, conversion to magnitude mode, inverse FT, etc.). It also applies to data sets with `1i` file missing, *e.g.*, to a magnitude spectrum.

4.2 baseline2

This script applies baseline correction to a series of spectra stored in pseudo-2D datasets, that is, in `2rr` and `2ii` files (but not in `2ir` or `2ri` files). For instance, Fig. 4 shows a series of 24 spectra from a T_1 inversion-recovery experiment on methane compressed inside MOF powder.

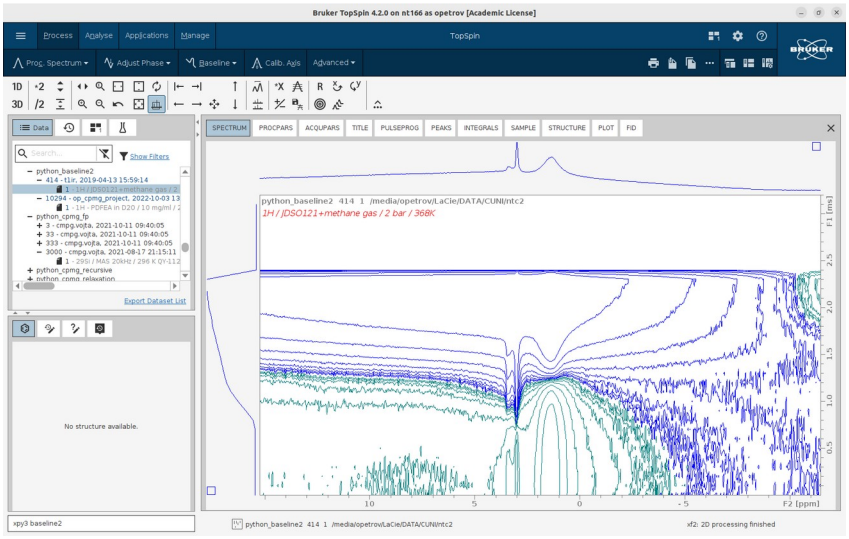

Fig. 4

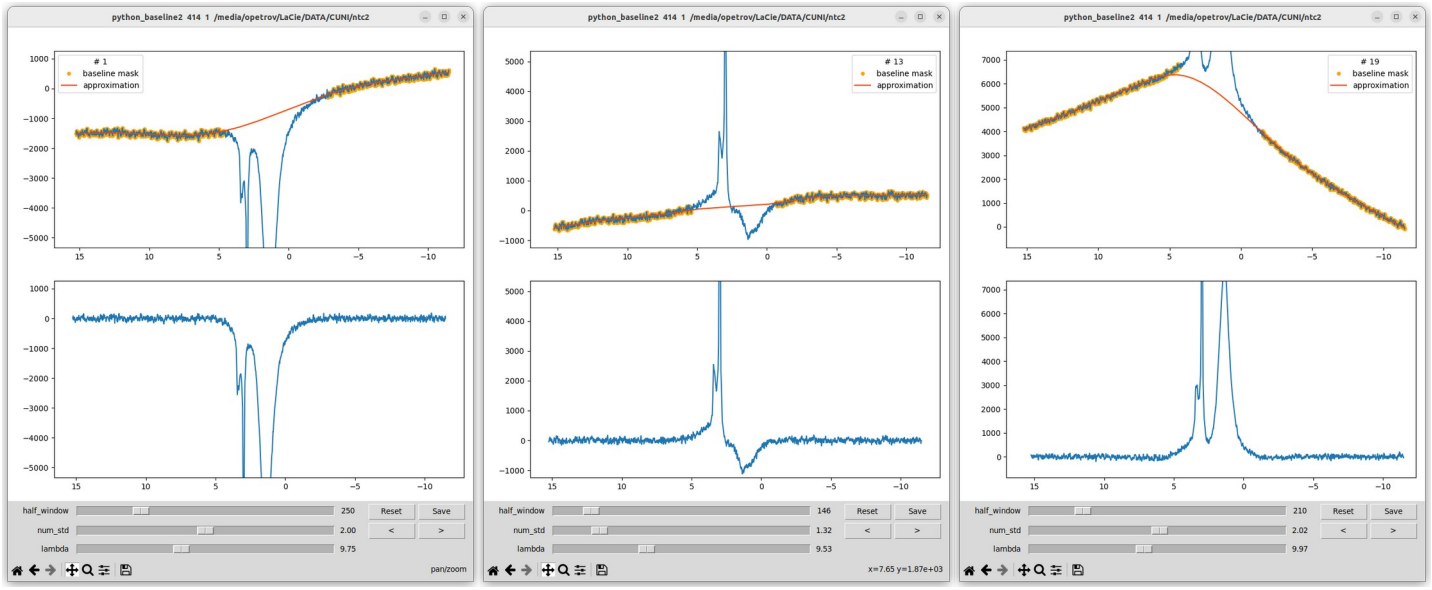

Fig. 5

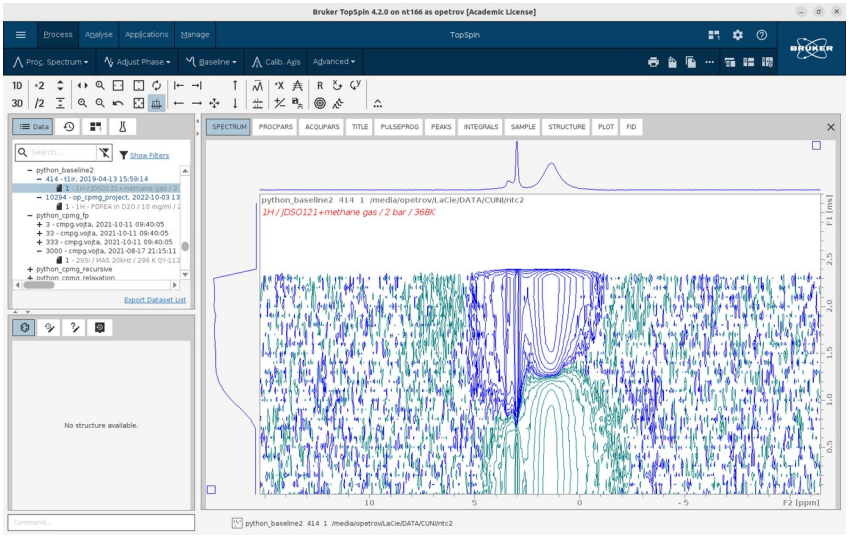

Fig. 6

Running the `xpy3 baseline2` command brings about a pop-up window identical to Fig. 2 but having an option to navigate across spectra with navigation buttons ">" and "<". By default, the first spectrum of a series is displayed. Once you inspect and optimize, if needed, the baseline estimate, you switch to another spectrum, and so on (Fig. 5). Eventually you click a `Save` button to write the entire series to the disk (Fig. 6).

Tip: If you find matplotlib too slow at re-drawing spectra in interactive mode (at zooming etc.), you can try and improve it by changing a line segment simplification parameter path.simplify_threshold in the matplotlib configuration file. Open the file [PYTHON]/.../matplotlib/mpl-data/matplotlibrc and change the value of path.simplify_threshold from the default 0.111111111111 to some greater value (up to 1.0).

4.3 cpmg_fp

This script is intended to be used with `qcpmg` type pulse programs which generate continuously sampled echo decays (aka echo trains). When processing an echo decay as a regular fid – *e.g.* with the command `fp` (FT + phase correction) – it results in a discrete 'spikelet' spectrum. To get a normal continuous complex spectrum, you ought to collapse the echo train into a single echo (sum of echoes) and apply the whole-echo processing steps: split the echo in the middle, swap the halves and FT. All these steps are implemented in `cpmg_fp`, plus auto-phasing of the spectrum as a bonus.

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 decay signal, same as in a Topspin window, and the bottom graph shows the average echo (sum of echoes) after first splitting the signal into individual echo segments. By default, all echo segments are included in the sum but you can discard later echoes using the slider. Manipulating with the number of echoes (`# echoes`) has an effect on SNR of the spectrum and also on the lineshape if the spectrum has of slow- and fast-relaxing components. Once you are certain about `# echoes`, click `OK` to get the spectrum, auto-phased (rotated) to have the maximum real part (Fig. 9).

For the script to work, echoes in the train must be separated by one or more zero data points. Attached to the package is a pulse program `op_qcpmg` which is a modification of `qcpmg` that ensures the echo train to include such zeroes.

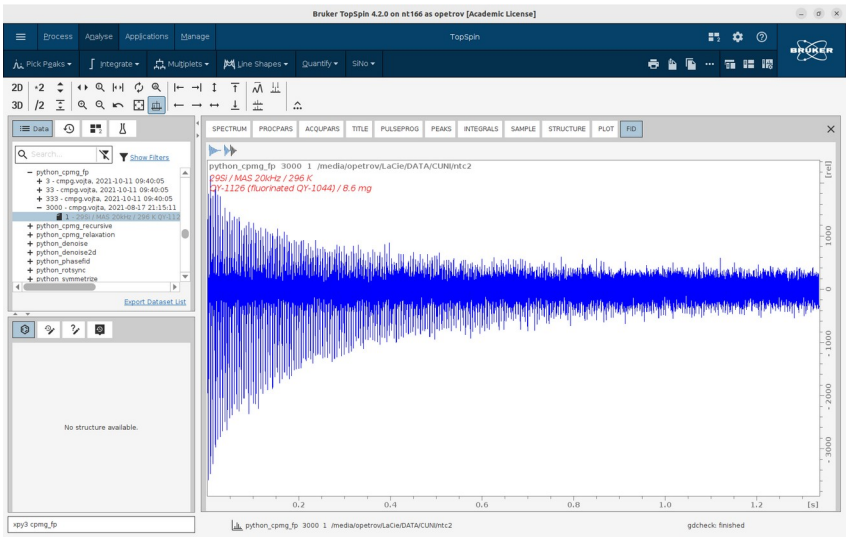

Fig. 7

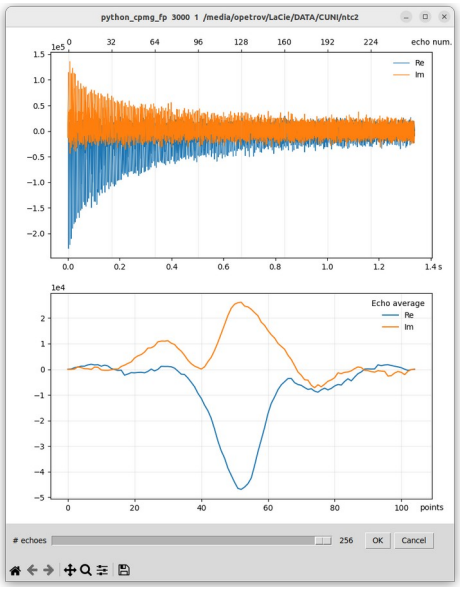

Fig. 8

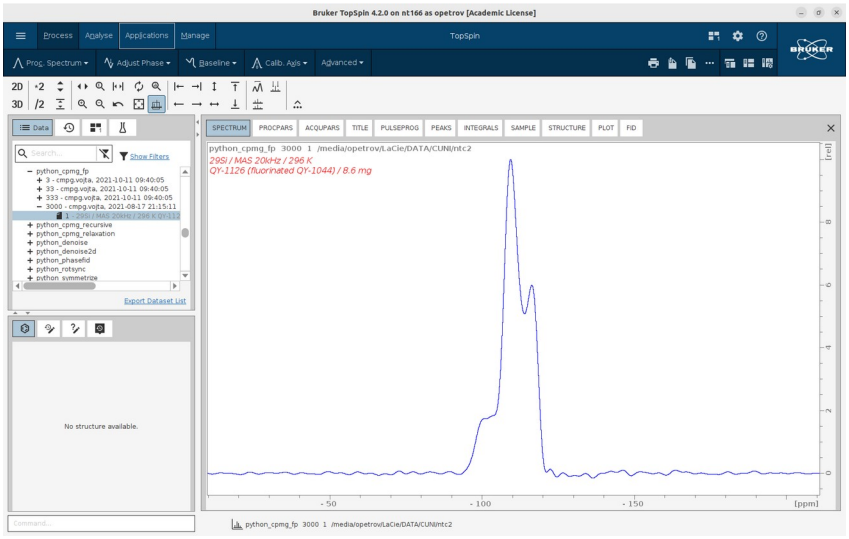

Fig. 9

4.4 cmpg_relaxation

This script again must be used in combination with `qcpmg` type pulse programs that generate continuous trains of zero-separated echoes. Given the zero positions, the script splits the train into individual echoes and quantifies them using either the midpoint intensity or integral intensity. The resultant echo decay curve – echo signal *vs.* echo time – can be 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 shows the echo decay curve, on a semi-log scale, measured by default from echo midpoints (with the option to switch to echo integrals). Clicking `Fit Me!` fits a mono-exponential model to the curve, the best-fit parameters being annotated (Fig. 12). In case of pronounced non-exponential decay, there is an option to use a stretched exponential model with a mean relaxation time reported (Fig. 13). Clicking `Save&Quit` saves the decay curve in a two-column text file in `<EXPNO>/pdata/ct1t2.txt`, for further processing (*e.g.* for Inverse Laplace transformation). Note that the annotation text with best-fit model parameters will not be saved, so mind that you write them down before clicking `Save&Quit`.

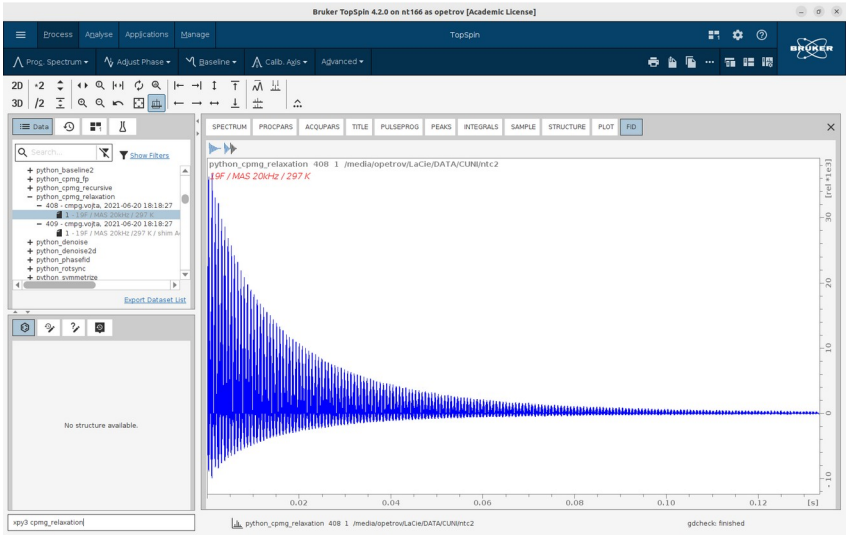

Fig. 10

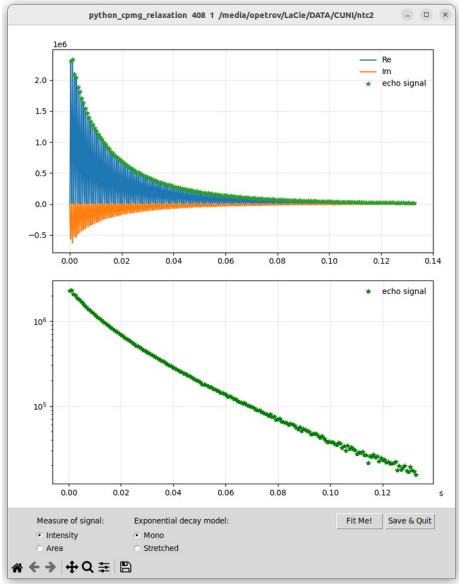

Fig. 11

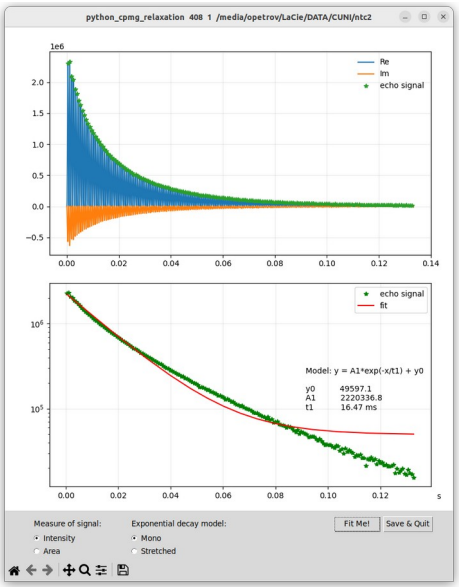

Fig. 12

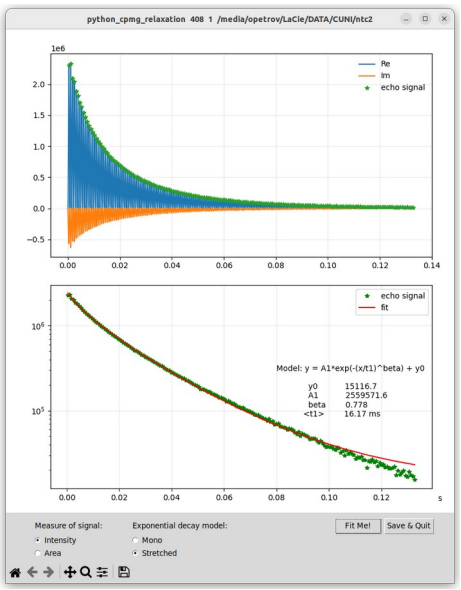

Fig. 13

4.5 cpmg_recursive

The functionality of this script is identical to `cpmg_relaxation` except for its using a cumulative sum of consecutive echoes as the measure of relaxation instead of individual echo intensity. The cumulative sum of echoes is fitted with a recursive mono-exponential relaxation model [6]. The advantage of this approach is two-fold: (i) summation decreases scatter in points on longer relaxation times; (ii) the recursive mono-exponential model is less sensitive to the presence of a fast-relaxing component, following more closely long-time relaxation trends. Fig. 14 shows the performance of `cpmg_recursive` using the same data as in Fig. 10. The GUI controls are analogous to those in Figs. 11-13.

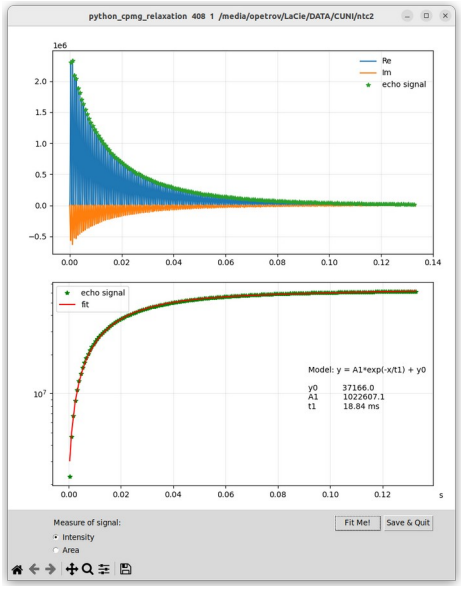

Fig. 14

4.6 denoise

This script implements Cadzow's method [11] of noise removal from 1D time-domain signals (FIDs or echoes). The method is based on approximation of a noisy signal with a low-rank matrix decomposition. For that purpose, the signal is first converted into a matrix where each row is a shifted version of the signal (aka a Toeplitz matrix). At the next step a singular value decomposition (SVD) of the matrix is performed and only the largest singular values are retained. Finally the thus approximated matrix is transformed back to a 1D signal. The signal is automatically converted to an AMX ('analog filter') type using the Topspin command `convdta` prior to the denoising procedure, to exclude difficulties with the group delay points preceding the actual data.

The script does not include a GUI for parameter adjustments. To improve the result, the user can try and truncate an FID through the Topspin's proc. parameter `TDeff`: the optimum `TDeff` lies somewhere between the full length of the FID (given by the acqu. parameter `TD`) and the value at which truncation artifacts ('sinc wiggles') become visible. Another parameter the user may change is the number of the largest singular values involved in the Toeplitz matrix approximation. This number is set automatically according to Gavish-Donoho's (G.-D.) threshold formula [7] but can be modified through the optional argument `offset` as follows:

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

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

Fig. 15 shows a ^{13}C spectrum of an organic precursor in a zeolite sample (on the top) and the same spectrum after applying `denoise` (on the bottom). Fig. 16 shows a ^{13}C spectrum of alginate, as another example of `denoise` performance, in which case it introduces artifacts at 47 and 170 ppm. As a rule, such artifacts appear 'off-phase' and thus are well distinguishable from real peaks.

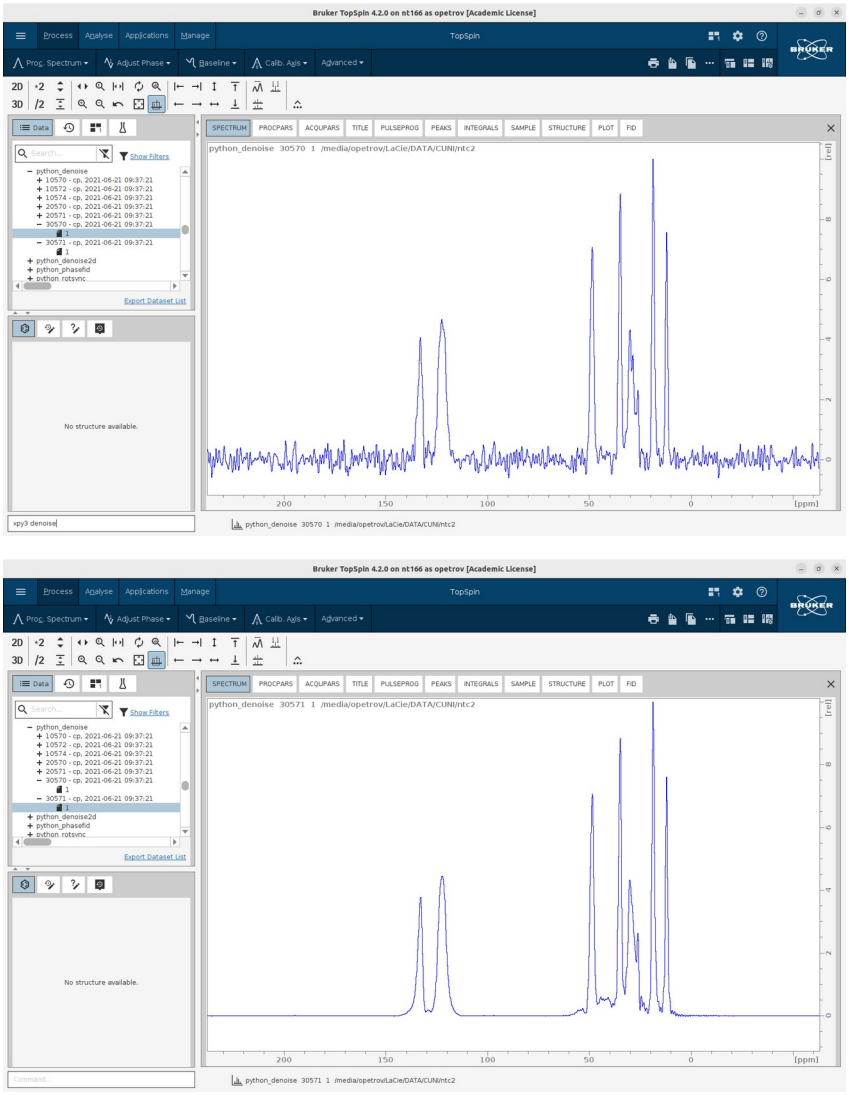

Fig. 15

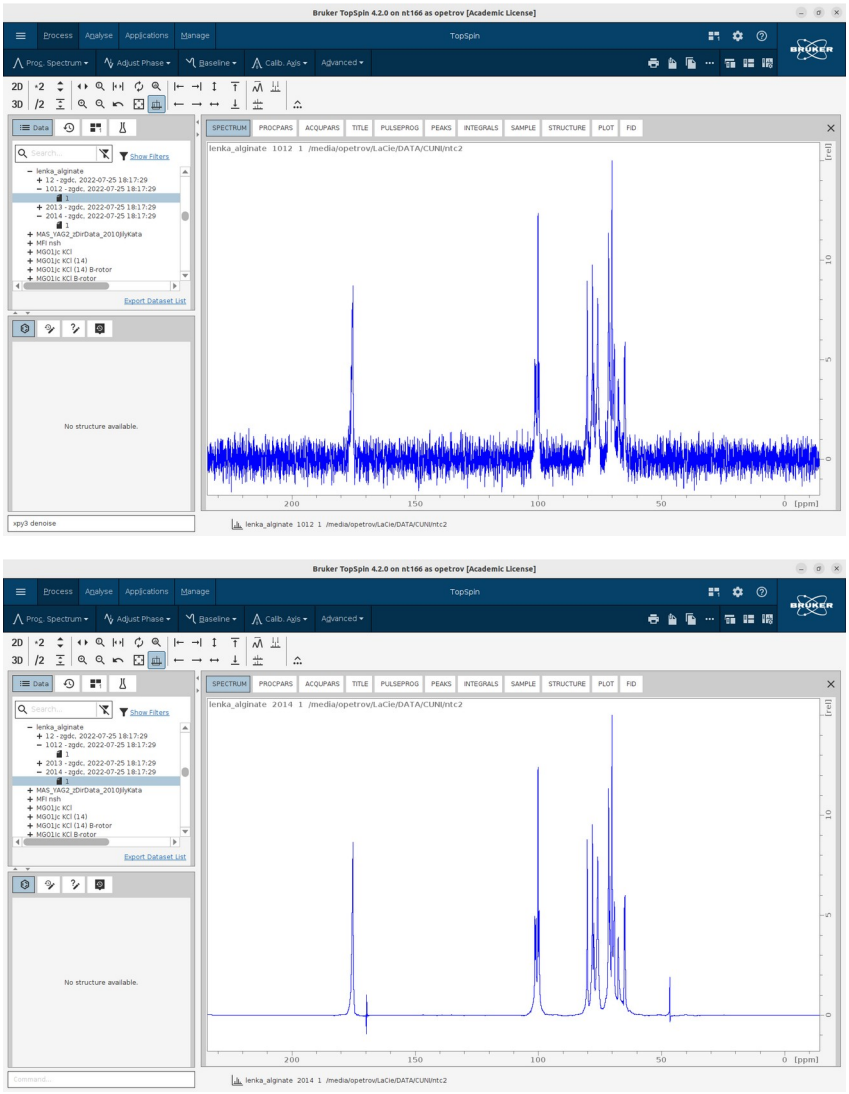

Fig. 16

4.7 denoise2

This script implements noise removal for 2D signals by a method detailing in [8]. Briefly, it is a variant of the PCA (SVD) based denoising method where the principal components (PCs) constituting a basis for signal approximation are attenuated proportionally to their noise-related content. You may think of it as a regularized SVD approximation, as opposed to the conventional low-rank SVD approximation used in `denoise`. The PC attenuation is achieved through 'averaging' same-rank PCs in repetitive SVD on randomly chosen data samples. The size of the samples is controlled by a parameter `nus` in the range of 0.25 to 1.0 and the number of samples by the parameter `samples`, each of them can be passed through external arguments (the square brackets indicate that they are optional):

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

If the arguments are not passed, the default values `nus=auto` and `samples=20` are used.

Fig. 17 shows a phase-sensitive 2D INADEQUATE spectrum of strychnine in $CDCl_3$ (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.

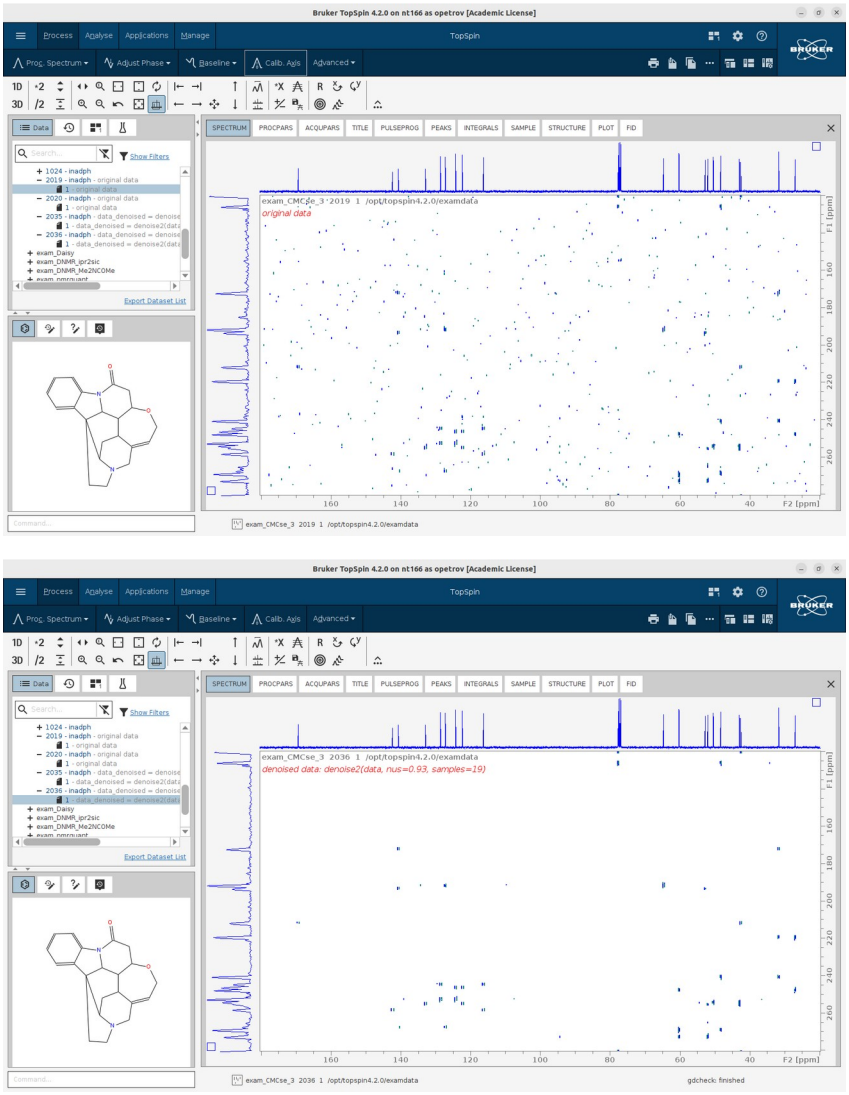

Fig. 17

Running `denoise2` on large datasets and with the number of samples greater than the default 20 may be painfully long (longer than a minute). Should you not want to wait, you could interrupt the script by closing an accompanying progress bar widget and restart it with a less number of samples. Like its 1D counterpart, `denoise2` copies the whole data set to a nearest available `EXPNO`.

4.8 icoshift

This is a frontend to utility functions of `pyicoshift` [9], a Python implementation of the *icoshift*, which stands for *interval Correlation Optimized shifting* algorithm for peak alignment. It usually applies to spectra whose peaks are prone to shifts, *e.g.*, due to fluctuations in pH, ion content, temperature, or instrument factors. The spectra can be from a pseudo-2D dataset or from 1D datasets mutually connected in `.md`, 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, enabling peak intervals selection. Omitting selection means you will treat the whole spectrum as one interval. Note that selected intervals are not editable – you can only clear them altogether with `Clear` 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, it can be chosen whether to align only this peak, keeping the rest of the spectrum intact (the option `n_intervals`), or to apply the shifts found for this peak to the whole spectra (the option `whole`). When `n_intervals` is used, 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 the sides of the spectra. 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`, meaning "*the signal with the highest correlation with all input signals*" [9]. Other targets 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 original Matlab version of `icoshift` [12]. In practice, you just go through all the targets and watch what difference it makes.

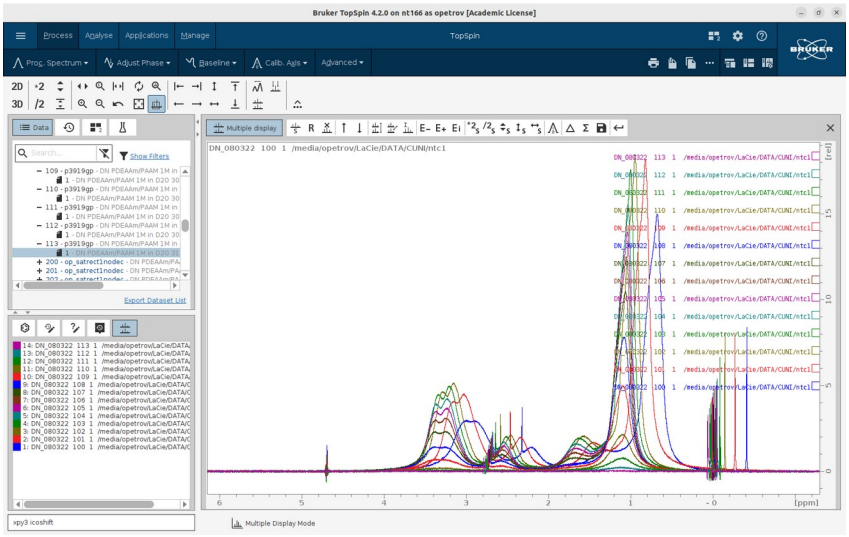

Fig. 18

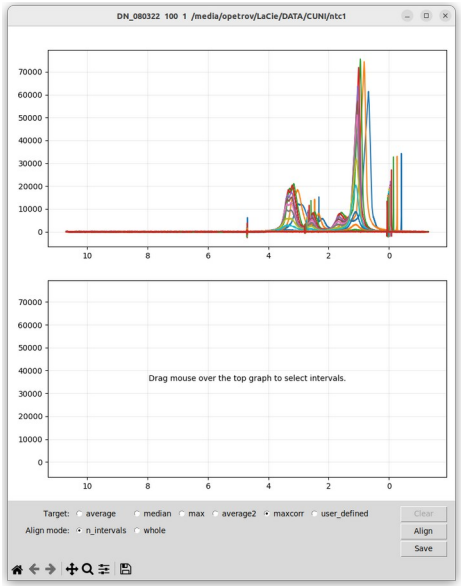

Fig. 19

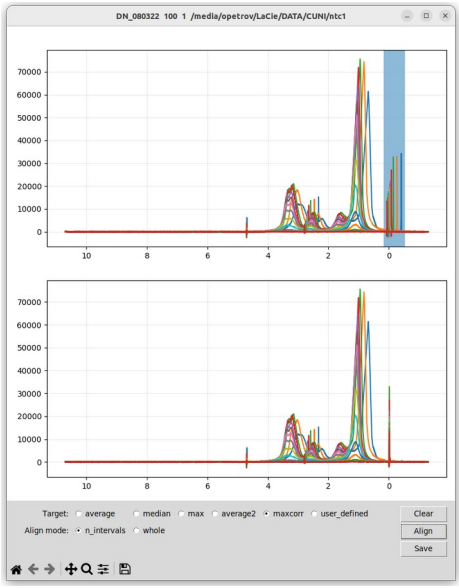

Fig. 20

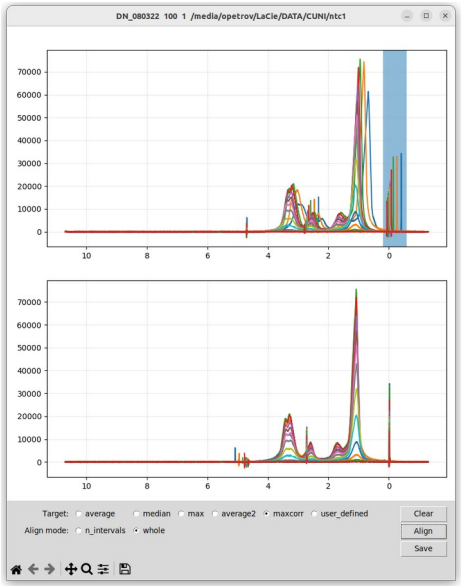

Fig. 21

4.9 phase_fid

This script implements zero-order phasing (rotation) of a signal in time domain, the first data point being used as a pivot. It is primarily intended for relaxometry where such a rotation might be needed for the real part maximization (for a signal quantitation purpose).

Figs. 23 shows rotation of a CPMG echo train from Fig. 22 such as to maximize its real part. As usual when raw data are modified, `phase_fid` saves the rotation under a new `EXPNO` (Fig. 24). There is an option to zeroize the imaginary part of the time-domain signal after rotation (*e.g.*, to symmetrize Fourier transform). The slider's value `PH_ref` can be used to correct the homonymous Topspin's parameter, for a next experiment.

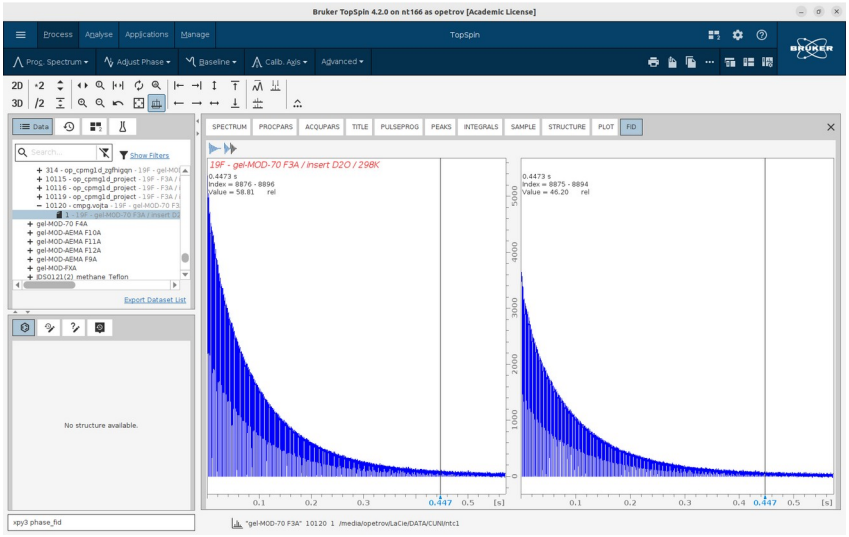

Fig. 22

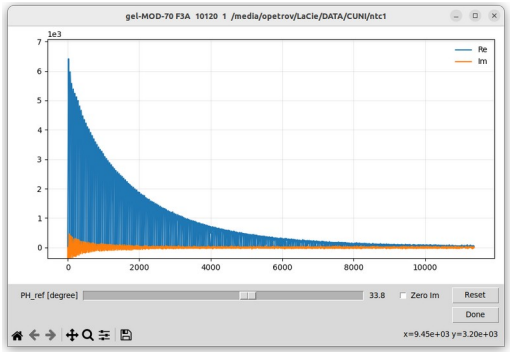

Fig. 23

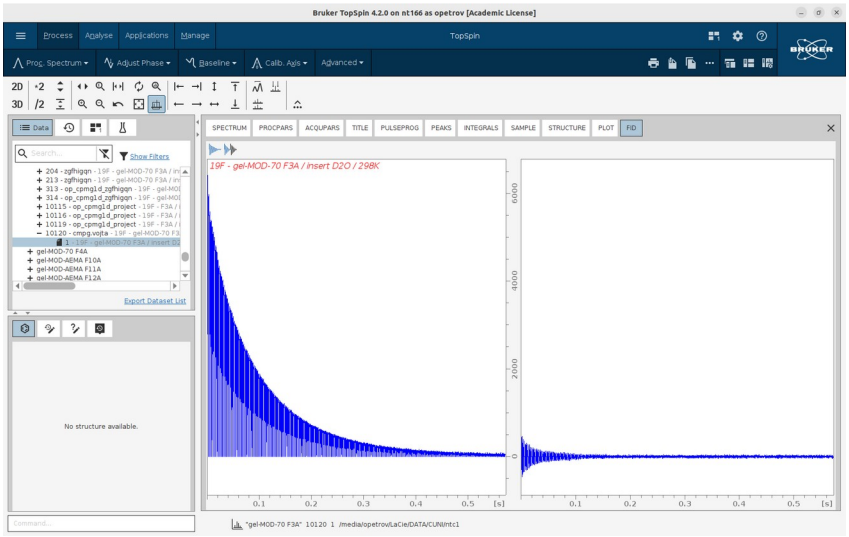

Fig. 24

4.10 rotsync

This tool is specific to magic angle spinning (MAS) NMR experiments. What it does is, it positions rotary echoes in a 1D MAS signal and down-samples the signal at those positions, which effectively makes it a rotor synchronized signal. The spectrum of rotor-synchronized signals lacks spinning sidebands (ssb), so that all spectral intensity (from both the central and satellite transitions) is folded into one “centerband”. Thus, when it comes to quantitation of chemical shift components, it can be done directly through integration over that centerband area. The pattern of the rotor-synchronized spectrum is equivalent to the case of the infinite MAS rate, which is convenient for the line shape analysis. Usually, the rotor synchronization is carried out on the fly during signal acquisition. However, the latter requires thorough adjustment of both sampling period and sampling offset as acquisition parameters – a tedious procedure which is difficult to control. One might prefer the present post-acquisition synchronization instead. Of course, it would require for the rotary echoes be well defined (detectable) over a few rotor periods.

Fig. 25 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. 26) showing a corresponding FID signal, 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 next available `EXPNO` and process it with `efp` command (Fig. 27).

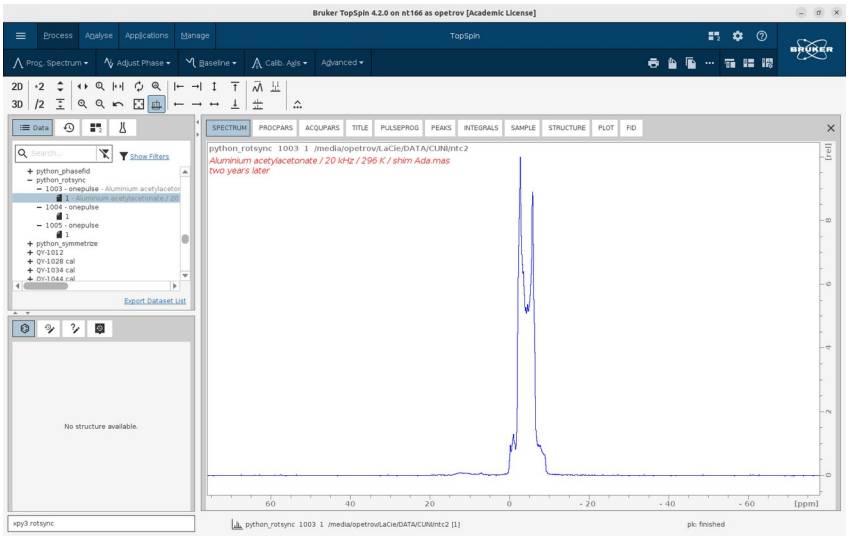

Fig. 25

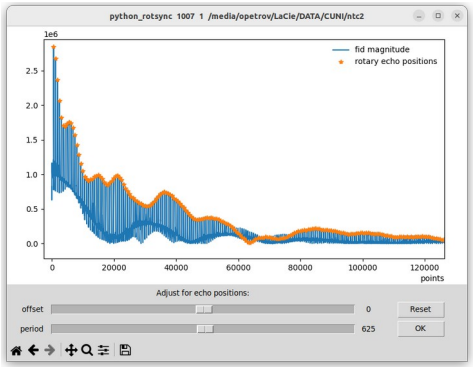

Fig. 26

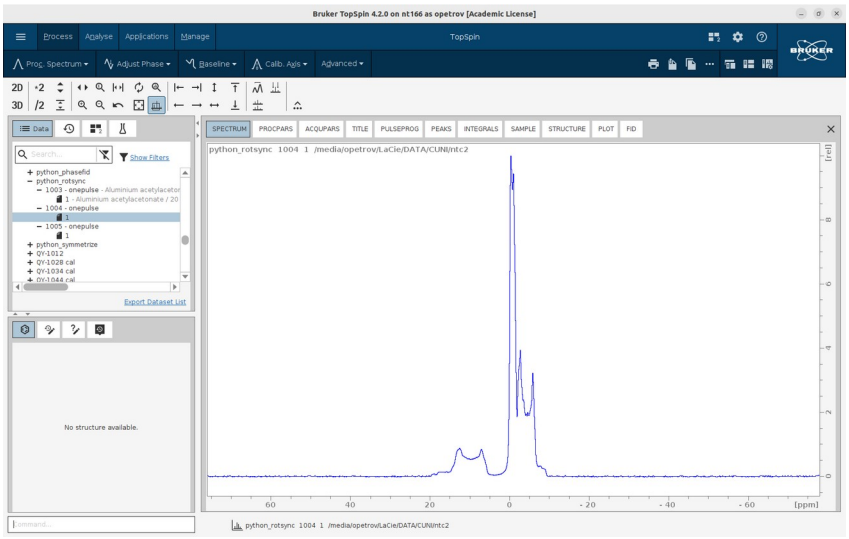

Fig. 27

4.11 rotsync_plus

Identical to `rotsync` except for a more involved procedure of echo quantitation, aiming at better SNR.

4.12 symmetrize

This, again, applies to quadrupolar MAS spectra of half-integer spins with spinning sidebands (ssb's). The ssb patterns may appear quite asymmetric, even when the prevalence of first-order quadrupolar interactions would imply symmetry. The observed ssb intensity distributions are skewed both positively and negatively, thus suggesting a few competing factors of the asymmetry, which may include magic angle misadjustment, uneven probe response as a function of frequency, a noneligible second-order quadrupolar term, etc. Fig. 28 is an example distribution of ssb intensity with positive skewness (^{27}Al NMR). If you want it symmetric, *e.g.* to see how ssb patterns compare, or to fit the first-order quadrupolar model to the ssb's, or simply to present it spectrum at its best (*i.e.*, without instrument factors engaged), then use this script.

Typing `xpy3 symmetrize` brings about a two-graph window (Fig. 29). 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 will be kept intact upon symmetrization. It is possible to re-adjust the cb limits with sliders. The cb recognition relies on a Topspin's auto-integration method (see `int auto`), namely 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 at line 105 of the script via `mode ='mean'`. If needed it can be changed to `mode = 'maximum'` in which case the ssb of a greater intensity will be placed on either side of the spectrum instead of the average intensity.) Clicking `OK` will return the symmetrized spectrum to Topspin (Fig. 30).

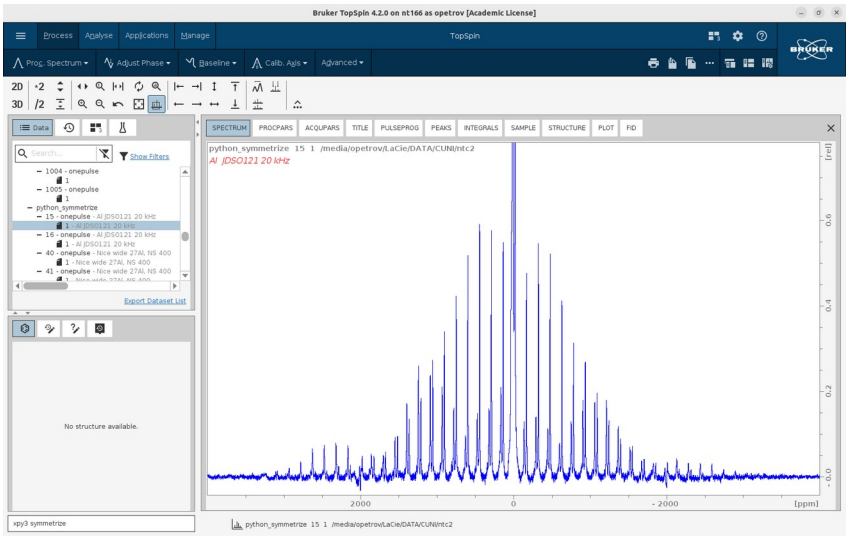

Fig. 28

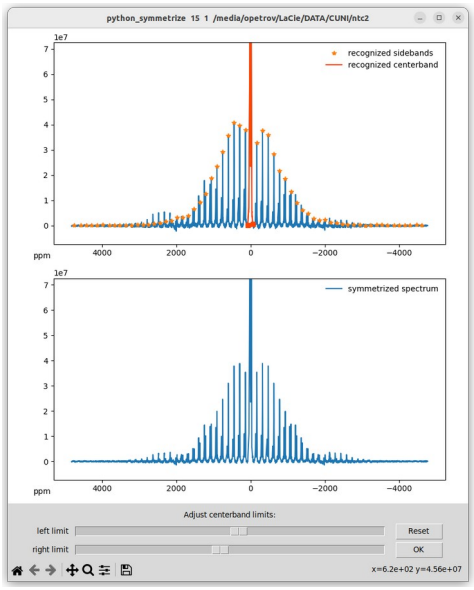

Fig. 29

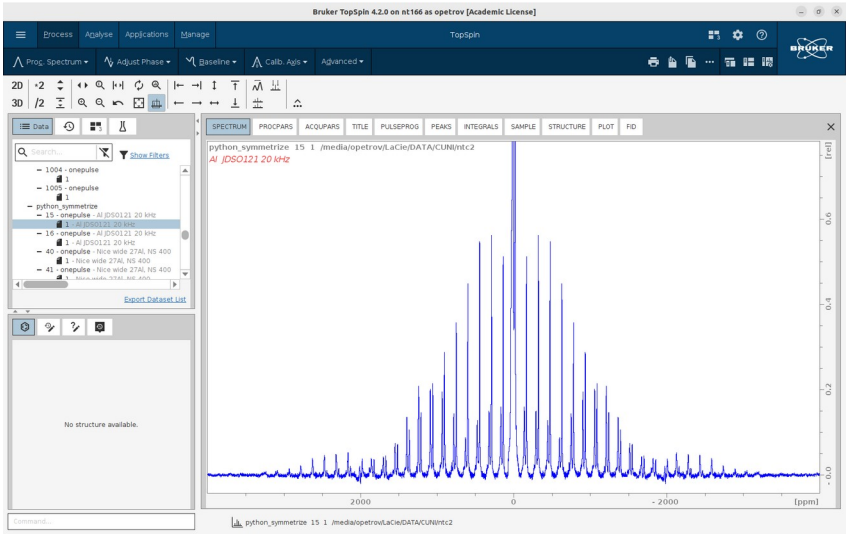

Fig. 30

5 Final remarks

The Bruker API for Python available in recent versions of Topspin [1] is a tempting offer for those who used to to write standalone Python scripts for NMR data processing. `xpy3-tools` is just a collection of such formerly standalone scripts turned into Topspin add-ons by means of the API, decorated in one style. The tasks performed by `xpy3-tools` may seem too diverse or specific, so you may say `xpy3-tools` will not find its user. But then AU programs and Jython scripts from a standard Topspin library are diverse too, yet they are being used a lot. Anyway, this version of `xpy3-tools` is just a beginning of a greater collection. The experienced NMR users 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.* (submitted)

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

[10] https://contourpy.readthedocs.io/en/v1.1.0

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

[12] https://doi.org/10.1039/9781849737531