NMRPy Documentation

Release 0.2.4

Johann Eicher, Johann Rohwer

Jun 23, 2020

CONTENTS

Contents:

CHAPTER

ONE

INTRODUCTION

NMRPy is a Python 3 module for the processing and analysis of NMR spectra. The functionality of NMRPy is structured to make the analysis of arrayed NMR spectra more intuitive.

A particular use case is the bulk processing and integration/deconvolution of arrayed NMR spectra obtained for enzyme reaction time-courses, with a view to determining enzyme-kinetic parameters for building systems-biology models [1,2].

1.1 References

1. Eicher, J. J.; Snoep, J. L. & Rohwer, J. M. (2012) Determining enzyme kinetics for systems biology with Nuclear Magnetic Resonance spectroscopy. *Metabolites* 2:818-843. DOI: 10.3390/metabo2040818
2. Badenhorst, M.; Barry, C. J.; Swanepoel, C. J.; van Staden, C. T.; Wissing, J. & Rohwer, J. M. (2019) Workflow for data analysis in experimental and computational systems biology: Using Python as 'glue'. *Processes* 7:460. DOI: 10.3390/pr7070460

CHAPTER

TWO

INSTALLATION

The following are some general guidelines for installing NMRPy, and are by no means the only way to install a Python package.

NMRPy is a pure Python package that runs on Windows, macOS and Linux. In addition to the option of installing directly from source (https://github.com/NMRPy/nmrpy), we provide binary installers for `pip` and `conda`.

2.1 Abbreviated requirements

NMRPy has a number of requirements that must be met before installation can take place. These should be taken care of automatically during installation. An abbreviated list of requirements follows:

- A Python 3.x installation (Python 3.6 or higher is recommended)
- The full SciPy Stack (see http://scipy.org/install.html).
- Jupyter Notebook (https://jupyter.org/)
- Matplotlib (https://matplotlib.org) with the `ipympl` backend
- Lmfit (https://lmfit.github.io/lmfit-py)
- Nmrglue (https://www.nmrglue.com)

Note: NMRPy will not work using Python 2.

2.2 Installation on Anaconda

The Anaconda Distribution, which is available for Windows, macOS and Linux, comes pre-installed with many packages required for scientific computing, including most of the dependencies required for NMRPy.

A number of the dependencies (lmfit, nmrglue and ipympl) are not available from the default conda channel. If you perform a lot of scientific or bioinformatics computing, it may be worth your while to add the following additional conda channels to your system, which will simplify installation (this is, however, not required, and the additional channels can also be specified once-off during the install command):

```
(base) $ conda config --add channels bioconda
(base) $ conda config --add channels conda-forge
```

2.2.1 Virtual environments

Virtual environments are a great way to keep package dependencies separate from your system files. It is highly recommended to install NMRPy into a separate environment, which first must be created (here we create an environment called `nmr`). It is recommended to use a Python version >=3.6 (here we use Python 3.7). After creation, activate the environment:

```
(base) $ conda create -n nmr python=3.7
(base) $ conda activate nmr
```

Then install NMRPy:

```
(nmr) $ conda install -c jmrohwer nmrpy
```

Or, if you have not added the additional channels system-wide:

```
(nmr) $ conda install -c bioconda -c conda-forge -c jmrohwer nmrpy
```

2.3 Direct `pip`-based install

First be sure to have Python 3 and `pip` installed. Pip is a useful Python package management system.

On Debian and Ubuntu-like systems these can be installed with the following terminal commands:

```
$ sudo apt install python3
$ sudo apt install python3-pip
```

On Windows, download Python from https://www.python.org/downloads/windows; be sure to install `pip` as well when prompted by the installer, and add the Python directories to the system PATH. You can verify that the Python paths are set up correctly by checking the `pip` version in a Windows Command Prompt:

```
> pip -V
```

On macOS you can install Python directly from https://www.python.org/downloads/mac-osx, or by installing Homebrew and then installing Python 3 with Homebrew. Both come with `pip` available.

Note: While most Linux distributions come pre-installed with a version of Python 3, the options for Windows and macOS detailed above are more advanced and for experienced users, who prefer fine-grained control. If you are starting out, we strongly recommend using Anaconda!

2.3.1 Virtual environments

As for an Anaconda-based install, it is highly recommended to install NMRPy into a separate virtual environment. There are several options for setting up your working environment. We will use virtualenvwrapper, which works out of the box on Linux and macOS. On Windows, virtualenvwrapper can be used under an MSYS environment in a native Windows Python installation. Alternatively, you can use virtualenvwrapper-win. This will take care of managing your virtual environments by maintaining a separate Python *site-directory* for you.

Install virtualenvwrapper using `pip`. On Linux and MacOS:

```
$ sudo -H pip install virtualenv
$ sudo -H pip install virtualenvwrapper
```

On Windows in a Python command prompt:

```
> pip install virtualenv
> pip install virtualenvwrapper-win
```

Make a new virtual environment for working with NMRPy (e.g. `nmr`), and specify that it use Python 3 (we used Python 3.7):

```
$ mkvirtualenv -p python3.7 nmr
```

The new virtual environment will be activated automatically, and this will be indicated in the shell prompt, e.g.:

```
(nmr) $
```

If you are not yet familiar with virtual environments we recommend you survey the basic commands (https://virtualenvwrapper.readthedocs.io/en/latest/) before continuing.

The NMRPy code and its dependencies can now be installed directly from PyPI into your virtual environment using `pip`.

```
(nmr) $ pip install nmrpy
```

2.4 Testing the installation

Various tests are provided to test aspects of the NMRPy functionality within the `unittest` framework. The tests should be run from a terminal and can be invoked with `nmrpy.test()` after importing the *nmrpy* module.

Only a specific subset of tests can be run by providing an additional argument:

```
nmrpy.test(tests='all')

:keyword tests: Specify tests to run (default 'all'). Running only a subset
                of tests can be selected using the following arguments:
'fidinit'       - Fid initialisation tests
'fidarrayinit'  - FidArray initialisation tests
'fidutils'      - Fid utilities tests
'fidarrayutils' - FidArray utilities tests
'plotutils'     - plotting utilities tests
```

When testing the plotting utilities, a number of `matplotlib` plots will appear. This tests that the peak and range selection widgets are working properly; the plot windows can be safely closed.

2.5 Working with NMRPy

Though the majority of NMRPy functionality can be used purely in a scripting context and executed by the Python interpreter, it will often need to be used interactively. We suggest two ways to do this:

2.5.1 Jupyter Notebook

The recommended way to run NMRPy is in the Jupyter Notebook environment. It has been installed by default with NMRPy and can be launched with (be sure to activate your virtual environment first):

```
(nmr) $ jupyter-notebook
```

The peak-picking and range-selection widgets in the Jupyter Notebook require the Matplotlib Jupyter Integration extension (*ipympl*). This is installed automatically but the extension needs to be activated at the beginning of every notebook thus:

```
In [1]:  %matplotlib widget
```

2.5.2 IPython

If you rather prefer a shell-like experience, IPython is an interactive Python shell with some useful functionalities like tab-completion. This has been installed by default with NMRPy and can be launched from the command line with:

```
(nmr) $ ipython
```

2.6 Documentation

Online documentation is available at https://nmrpy.readthedocs.io. The documentation is also distributed in PDF format in the `docs` subfolder of the `nmrpy` folder in site-packages where the package is installed.

The `docs` folder also contains an example Jupyter notebook (`quickstart_tutorial.ipynb`) that mirrors the *Quickstart Tutorial*.

CHAPTER

THREE

QUICKSTART TUTORIAL

This is a "quickstart" tutorial for NMRPy in which an Agilent (Varian) NMR dataset will be processed. The following topics are explored:

- *Importing*
- *Apodisation and Fourier-transformation*
- *Phase-correction*
- *Calibration*
- *Peak-picking*
- *Deconvolution*
- *Saving / Loading*
- *Full tutorial script*

This tutorial will use the test data in the nmrpy install directory:

```
site-packages/nmrpy/tests/test_data/test1.fid
```

The dataset consists of a time array of spectra of the phosphoglucose isomerase reaction:

fructose-6-phosphate -> glucose-6-phosphate

An example Jupyter notebook is provided in the `docs` subdirectory of the nmrpy install directory, which mirrors this Quickstart Tutorial.

```
site-packages/nmrpy/docs/quickstart_tutorial.ipynb
```

3.1 Importing

The basic NMR project object used in NMRPy is the *`FidArray`*, which consists of a set of *`Fid`* objects, each representing a single spectrum in an array of spectra.

The simplest way to instantiate an *`FidArray`* is by using the *`from_path()`* method, and specifying the path of the *.fid* directory:

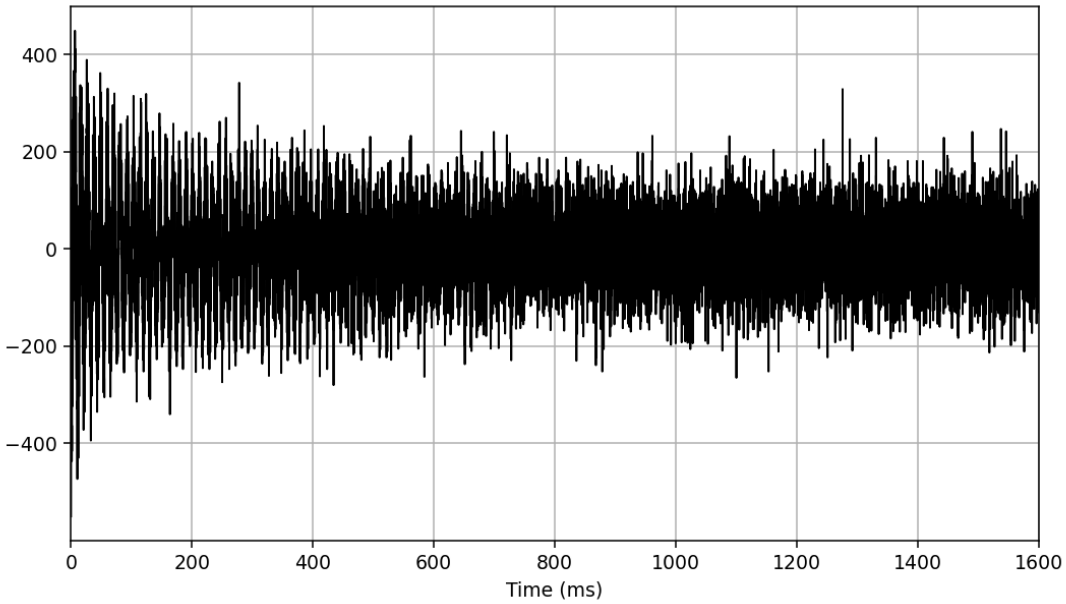
```
>>> import nmrpy
>>> import os, sysconfig
>>> fname = os.path.join(sysconfig.get_paths()['purelib'], 'nmrpy',
                         'tests', 'test_data', 'test1.fid')
>>> fid_array = nmrpy.from_path(fname)
```

You will notice that the `fid_array` object is instantiated and now owns several attributes, most of which are of the form `fidXX` where *XX* is a number starting at 00. These are the individual arrayed *Fid* objects.

3.2 Apodisation and Fourier-transformation

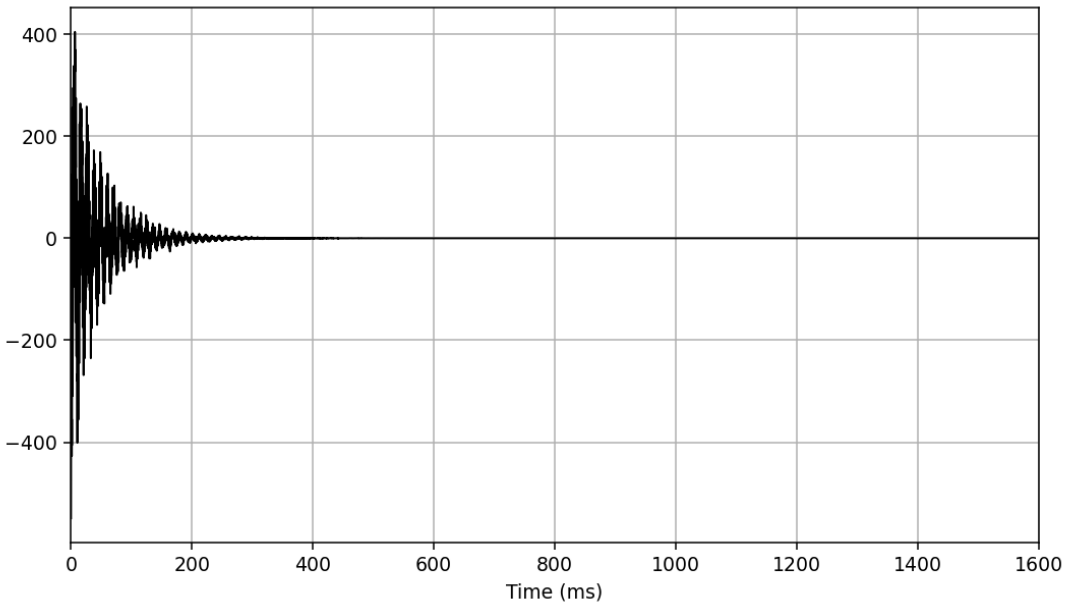
To quickly visualise the imported data, we can use the plotting functions owned by each *Fid* instance. This will not display the imaginary portion of the data:

```
>>> fid_array.fid00.plot_ppm()
```



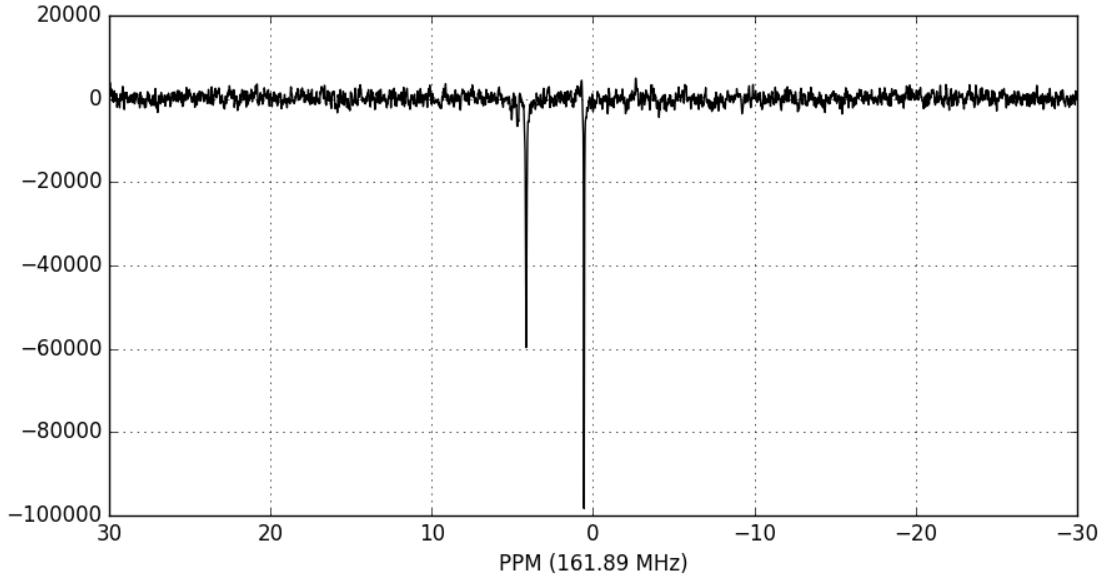
We now perform apodisation of the FIDs using the default value of 5 Hz, and visualise the result:

```
>>> fid_array.emhz_fids()
>>> fid_array.fid00.plot_ppm()
```



Finally, we Fourier-transform the data into the frequency domain:

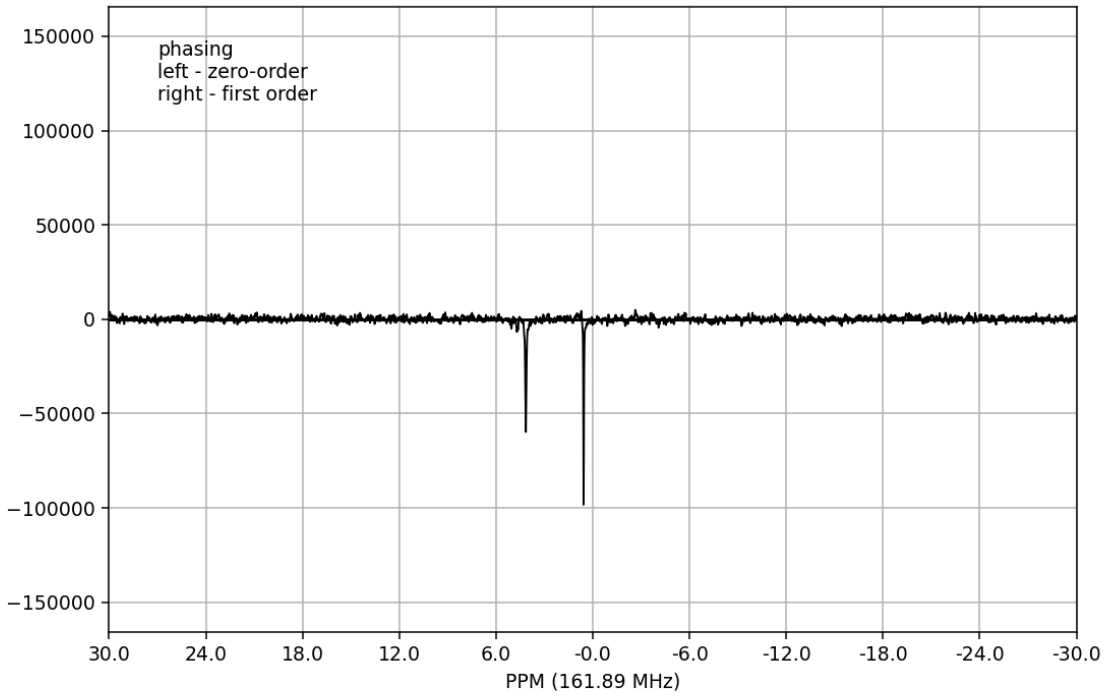
```
>>> fid_array.ft_fids()
>>> fid_array.fid00.plot_ppm()
```



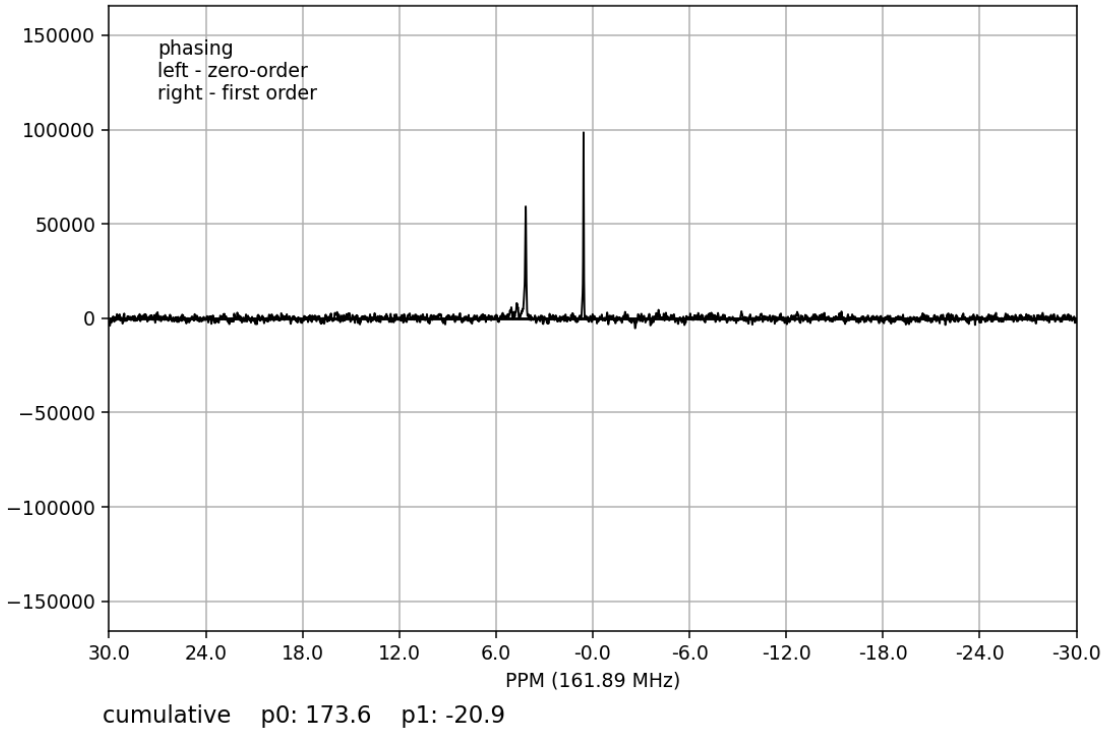
3.3 Phase-correction

It is clear from the data visualisation that at this stage the spectra require phase-correction. NMRPy provides a number of GUI widgets for manual processing of data. In this case we will use the *phaser()* method on `fid00`:

```
>>> fid_array.fid00.phaser()
```



Dragging with the left mouse button and right mouse button will apply zero- and first-order phase-correction, respectively. The cumulative phase correction for the zero-order (`p0`) and first-order (`p1`) phase angles is displayed at the bottom of the plot so that these can be applied programatically to all *`Fid`* objects in the *`FidArray`* using the *`ps_fids()`* method.

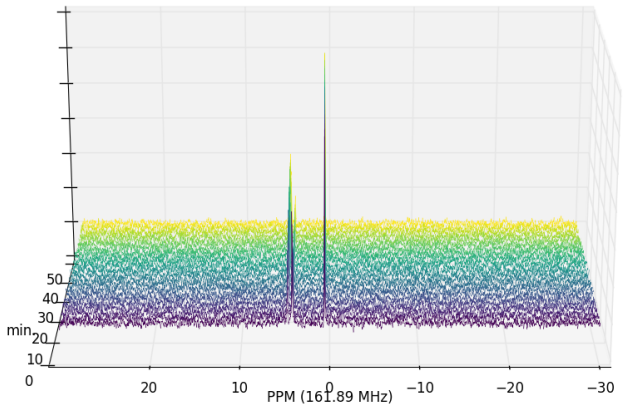


Alternatively, automatic phase-correction can be applied at either the *FidArray* or *Fid* level. We will apply it to the whole array:

```
>>> fid_array.phase_correct_fids()
```

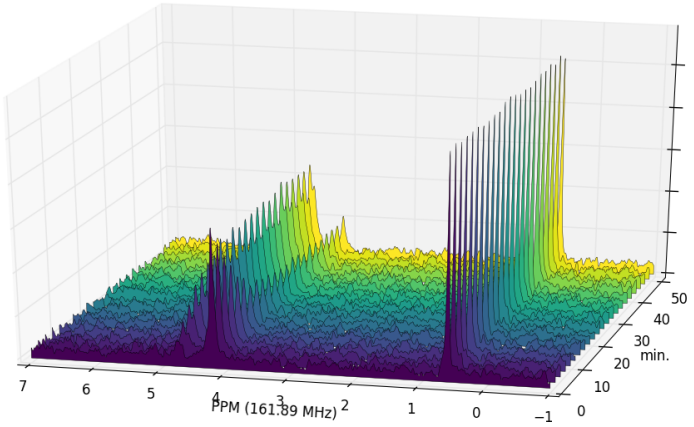
And plot an array of the phase-corrected data:

```
>>> fid_array.plot_array()
```



Zooming in on the relevant peaks, changing the view perspective, and filling the spectra produces a more interesting plot:

```
>>> fid_array.plot_array(upper_ppm=7, lower_ppm=-1, filled=True, azim=-76, elev=23)
```



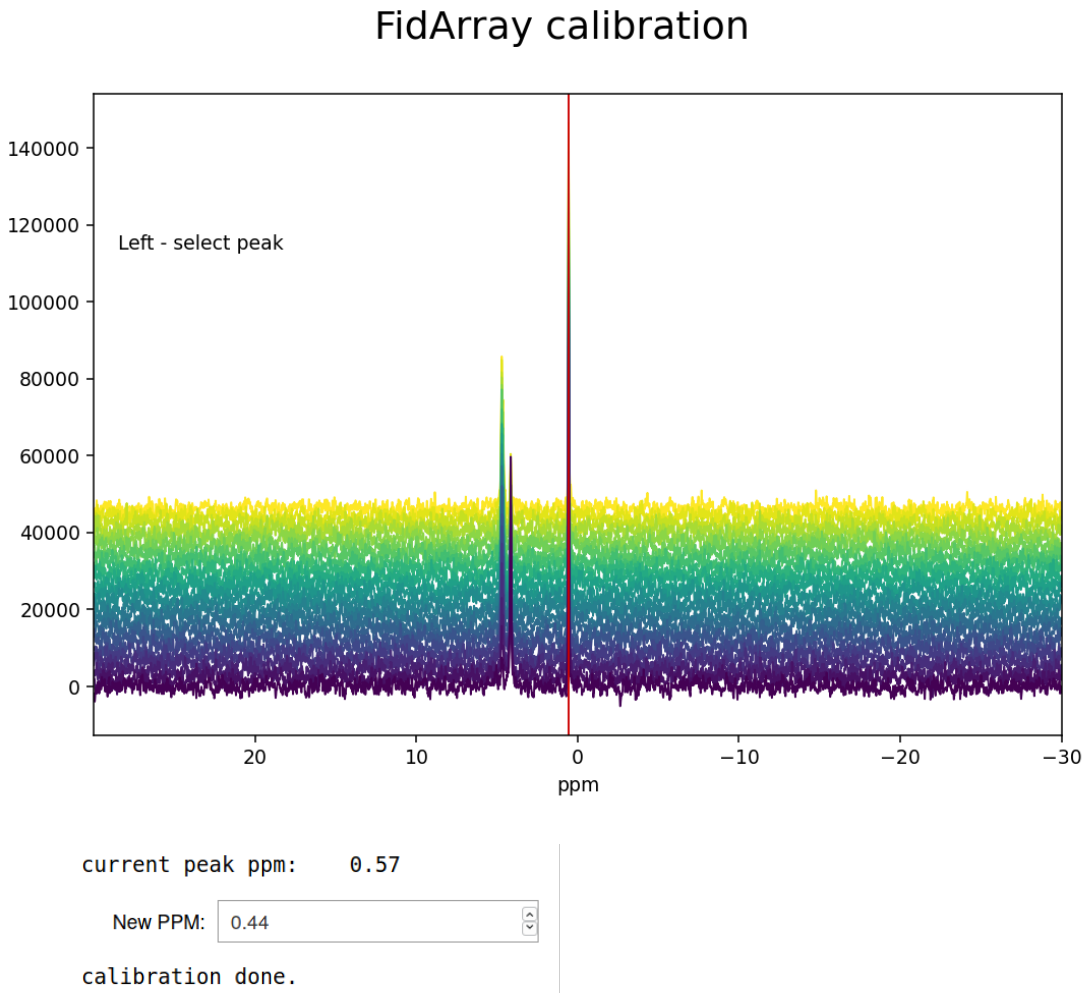
At this stage it is useful to discard the imaginary component of our data, and possibly normalise the data (by the maximum data value amongst the *Fid* objects):

```
>>> fid_array.real_fids()
>>> fid_array.norm_fids()
```

3.4 Calibration

The spectra may need calibration by assigning a chemical shift to a reference peak of a known standard and adjusting the spectral offset accordingly. To this end, a *calibrate()* convenience method exists that allows the user to easily select a peak and specify the PPM. This method can be applied at either the *FidArray* or *Fid* level. We will apply it to the whole array:

```
>>> fid_array.calibrate()
```

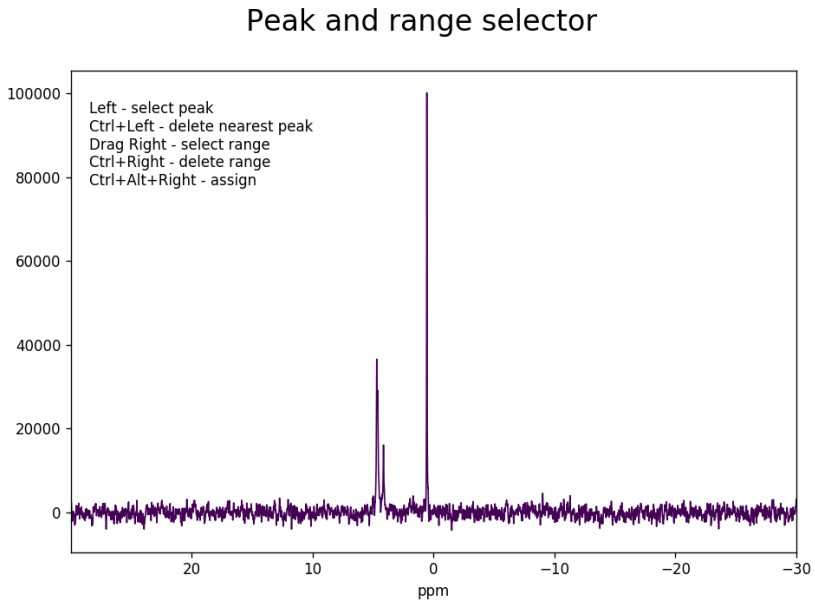


Left-clicking selects a peak and its current ppm value is displayed below the spectrum. The new ppm value can be entered in a text box, and hitting `Enter` completes the calibration process. Here we have chosen triethyl phosphate (TEP) as reference peak and assigned its chemical shift value of 0.44 ppm (the original value was 0.57 ppm, and the offset of all the spectra in the array has been adjusted by 0.13 ppm after the calibration).

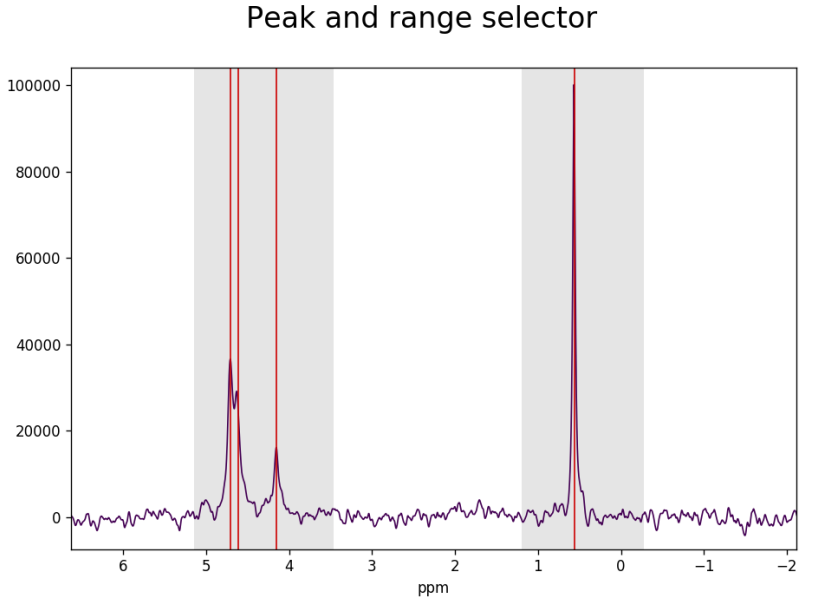
3.5 Peak-picking

To begin the process of integrating peaks by deconvolution, we will need to pick some peaks. The *peaks* attribute of a *Fid* is an array of peak positions, and *ranges* is an array of range boundaries. These two objects are used in deconvolution to integrate the data by fitting Lorentzian/Gaussian peak shapes to the spectra. *peaks* and *ranges* may be specified programatically, or picked using the interactive GUI widget:

```
>>> fid_array.peakpicker(fid_number=10)
```



Left-clicking specifies a peak selection with a vertical red line. Dragging with a right-click specifies a range to fit independently with a grey rectangle:



Inadvertent wrongly selected peaks can be deleted with Ctrl+left-click; wrongly selected ranges can be deleted with Ctrl+right-click. Once you are done selecting peaks and ranges, these need to be assigned to the *FidArray*; this is achieved with a Ctrl+Alt+right-click.

Ranges divide the data into smaller portions, which significantly speeds up the process of fitting of peakshapes to the data. Range-specification also prevents incorrect peaks from being fitted by the fitting algorithm.

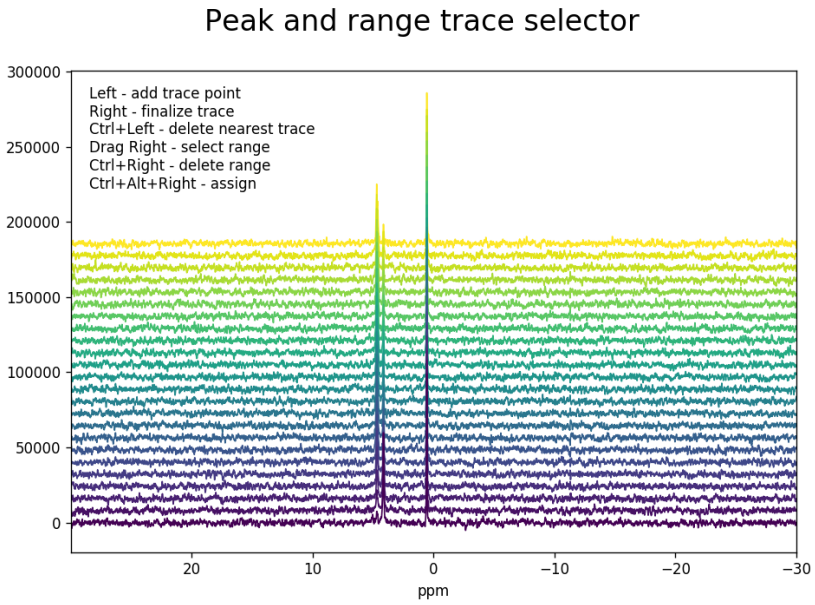
Having used the *peakpicker() FidArray* method (as opposed to the *peakpicker()* on each individual *Fid* instance), the peak and range selections have now been assigned to each *Fid* in the array:

```
>>> print(fid_array.fid00.peaks)
[ 4.73  4.63  4.15  0.55]
>>> print(fid_array.fid00.ranges)
[[ 5.92  3.24]
 [ 1.19 -0.01]]
```

3.5.1 Peak-picking trace selector

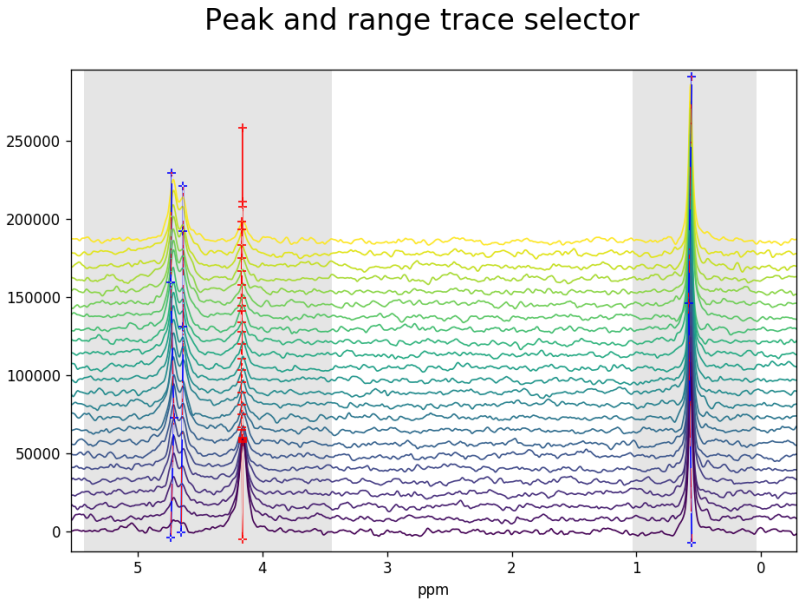
Sometimes peaks are subject to drift so that the chemical shift changes over time; this can happen, e.g., when the pH of the reaction mixture changes as the reaction proceeds. NMRPy offers a convenient trace selector, with which the drift of the peaks can be traced over time and the chemical shift selected accordingly as appropriate for the particular *Fid*.

```
>>> fid_array.peakpicker_traces(voff=0.08)
```



As for the *peakpicker()*, ranges are selected by dragging the right mouse button and can be deleted with Ctrl+right-click. A peak trace is initiated by left-clicking below the peak underneath the first Fid in the series. This selects a point and anchors the trace line, which is displayed in red as the mouse is moved. The trace will attempt to follow the highest peak. Further trace points can be added by repeated left-clicking, thus tracing the peak through the individual Fids in the series. It is not necessary to add an anchor point for every Fid, only when the trace needs to change direction. Once the trace has traversed all the Fids, select a final trace point (left-click) and then finalize the trace with a right-click. The trace will change colour from red to blue to indicate that it has been finalized.

Additional peaks can then be selected by initiating a new trace. Wrongly selected traces can be deleted by Ctrl+left-click at the bottom of the trace that should be removed. Note that the interactive buttons on the matplotlib toolbar for the figure can be used to zoom and pan into a region of interest of the spectra.



As previously, peaks and ranges need to be assigned to the *FidArray* with Ctrl+Alt+right-click. As can be seen below, the individual peaks have different chemical shifts for the different Fids, although the drift in these spectra is not significant so that *peakpicker_traces()* need not have been used and *peakpicker()* would have been sufficient. This is merely for illustrative purposes.

```
>>> print(p.fid00.peaks)
[4.73311164 4.65010807 0.55783899 4.15787759]
>>> print(p.fid10.peaks)
[4.71187817 4.6404565  0.5713512  4.16366854]
>>> print(p.fid20.peaks)
[4.73311164 4.63466555 0.57907246 4.16366854]
```

3.6 Deconvolution

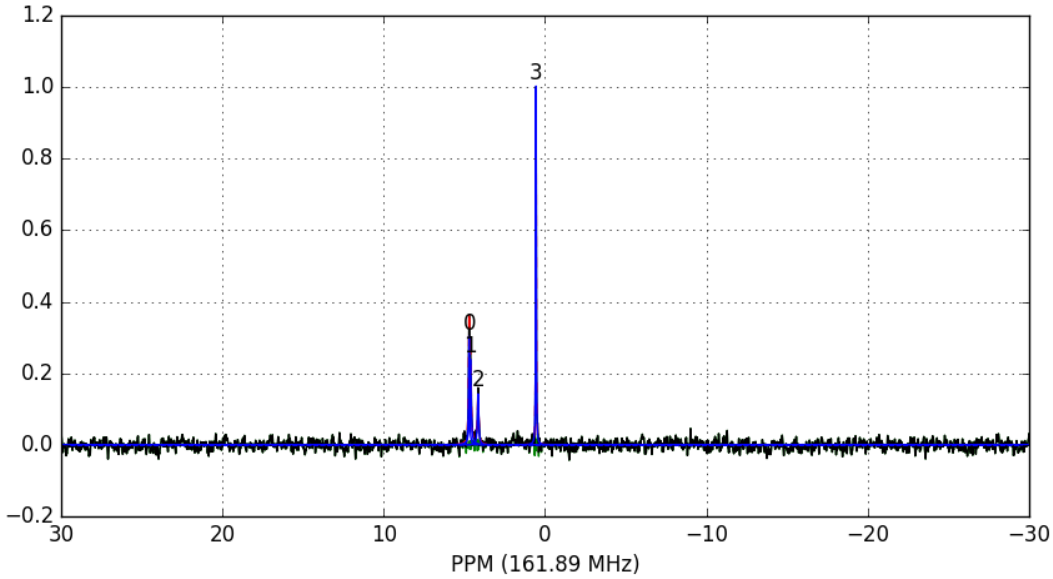
Individual *Fid* objects can be deconvoluted with *deconv()*. *FidArray* objects can be deconvoluted with *deconv_fids()*. By default this is a multiprocessed method (*mp=True*), which will fit pure Lorentzian lineshapes (*frac_gauss=0.0*) to the *peaks* and *ranges* specified in each *Fid*.

We shall fit the whole array at once:

```
>>> fid_array.deconv_fids()
```

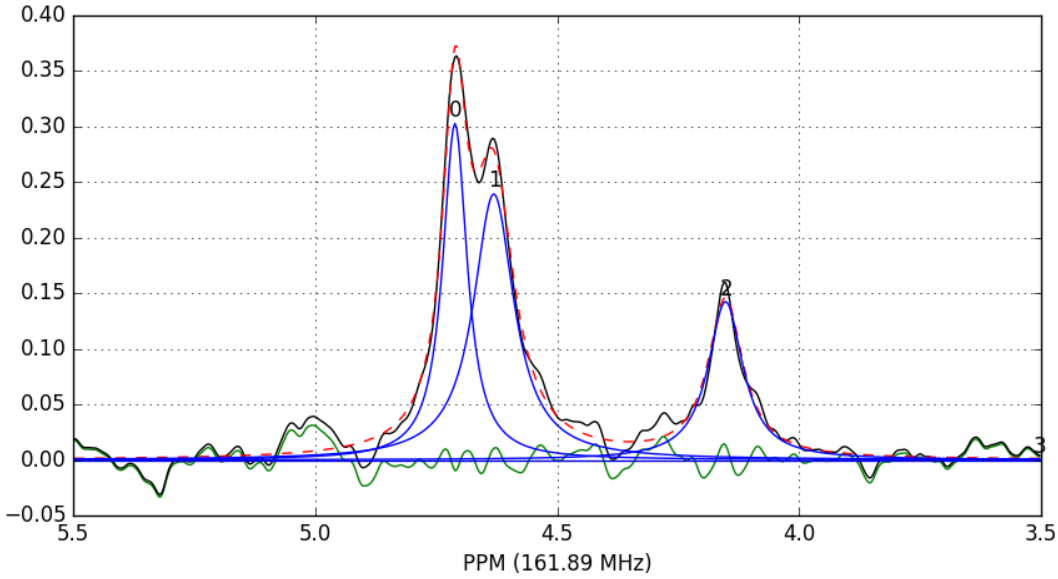
And visualise the deconvoluted spectra:

```
>>> fid_array.fid10.plot_deconv()
```



Zooming-in to a set of peaks makes clear the fitting result:

```
>>> fid_array.fid10.plot_deconv(upper_ppm=5.5, lower_ppm=3.5)
>>> fid_array.fid10.plot_deconv(upper_ppm=0.9, lower_ppm=0.2)
```



In this case, peaks 0 and 1 belong to glucose-6-phosphate, peak 2 belongs to fructose-6-phosphate, and peak 3 belongs to triethyl-phosphate.

We can view the deconvolution result for the whole array using *plot_deconv_array()*. Fitted peaks appear in red:

```
>>> fid_array.plot_deconv_array(upper_ppm=6, lower_ppm=3)
```

Peak integrals of the entire *FidArray* are stored in *deconvoluted_integrals*, or in each individual *Fid* as

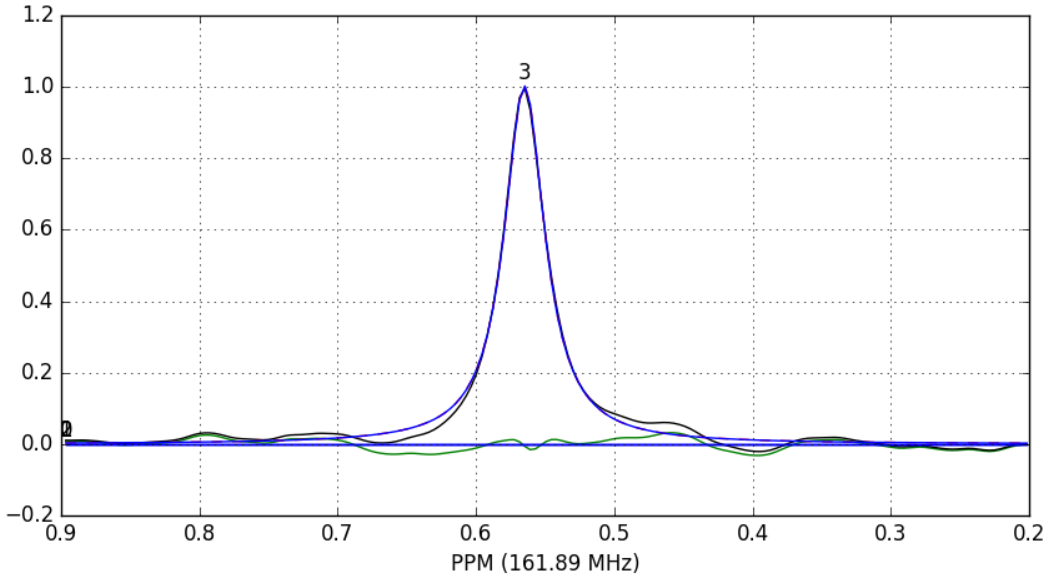
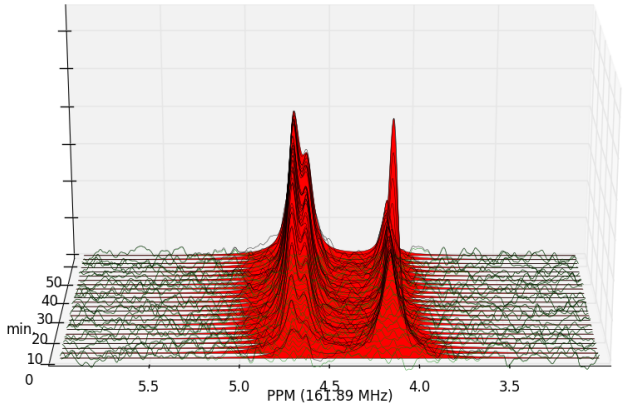


Fig. 1: The lines are colour-coded according to:

- *Blue*: individual peak shapes (and peak numbers above);
- *Black*: original data;
- *Red*: summed peak shapes;
- *Green*: residual (original data - summed peakshapes).



deconvoluted_integrals.

We could easily plot the species integrals using the following code:

```
from matplotlib import pyplot as plt

integrals = fid_array.deconvoluted_integrals.transpose()

g6p = integrals[0] + integrals[1]
f6p = integrals[2]
tep = integrals[3]

#scale species by internal standard tep (5 mM)
g6p = 5.0*g6p/tep.mean()
f6p = 5.0*f6p/tep.mean()
tep = 5.0*tep/tep.mean()

species = {'g6p': g6p,
           'f6p': f6p,
           'tep': tep}

fig = plt.figure()
ax = fig.add_subplot(111)
for k, v in species.items():
    ax.plot(fid_array.t, v, label=k)

ax.set_xlabel('min')
ax.set_ylabel('mM')
ax.legend(loc=0, frameon=False)

plt.show()
```

3.7 Saving / Loading

The current state of any *FidArray* object can be saved to file using the *save_to_file()* method:

```
>>> fid_array.save_to_file(filename='fidarray.nmrpy')
```

And reloaded using *from_path()*:

```
>>> fid_array = nmrpy.from_path(fid_path='fidarray.nmrpy')
```

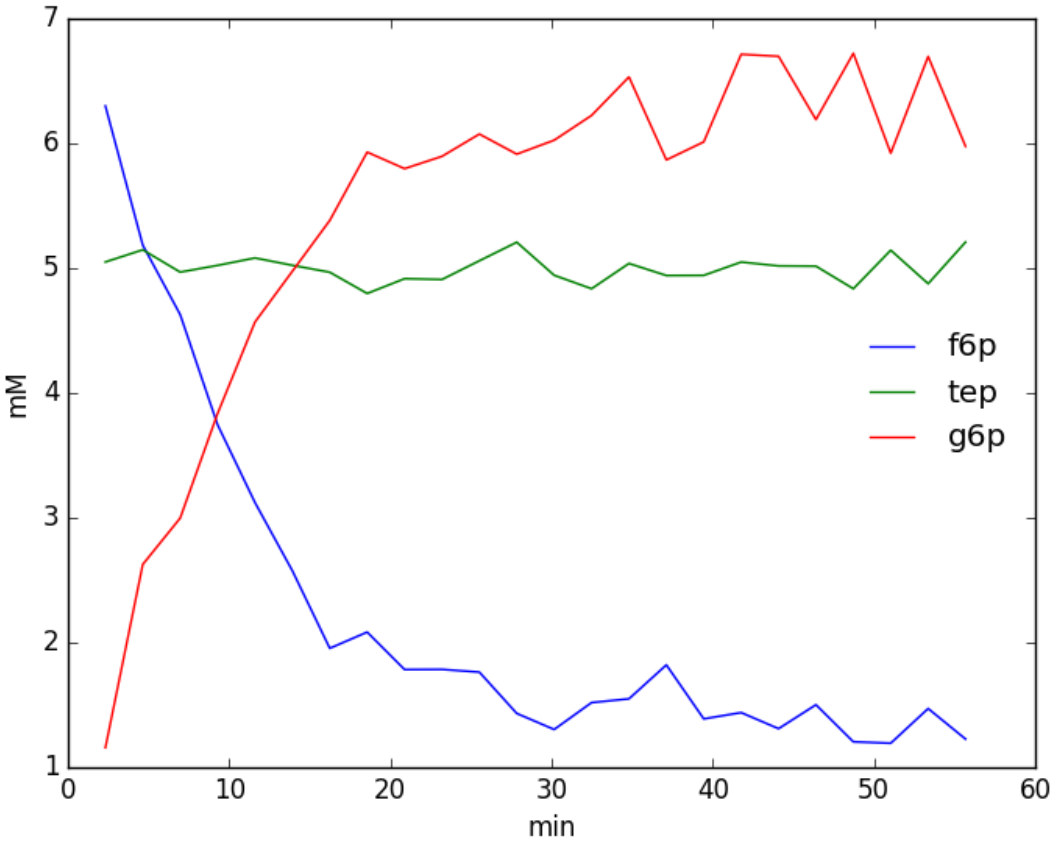
3.8 Full tutorial script

The full script for the quickstart tutorial:

```
import nmrpy
import os, sysconfig
from matplotlib import pyplot as plt

fname = os.path.join(sysconfig.get_paths()['purelib'], 'nmrpy',
                     'tests', 'test_data', 'test1.fid')
fid_array = nmrpy.from_path(fid_path=fname)
```

(continues on next page)


7
6
5
4
3
2
1
mM
0
10
20
30
40
50
60
min
f6p
tep
g6p

(continued from previous page)

```
fid_array.emhz_fids()
#fid_array.fid00.plot_ppm()
fid_array.ft_fids()
#fid_array.fid00.plot_ppm()
#fid_array.fid00.phaser()
fid_array.phase_correct_fids()
#fid_array.fid00.plot_ppm()
fid_array.real_fids()
fid_array.norm_fids()
#fid_array.plot_array()
#fid_array.plot_array(upper_ppm=7, lower_ppm=-1, filled=True, azim=-76, elev=23)
#fid_array.calibrate()

peaks = [ 4.73,  4.63,  4.15,  0.55]
ranges = [[ 5.92,  3.24], [ 1.19, -0.01]]
for fid in fid_array.get_fids():
    fid.peaks = peaks
    fid.ranges = ranges

fid_array.deconv_fids()

#fid_array.fid10.plot_deconv(upper_ppm=5.5, lower_ppm=3.5)
#fid_array.fid10.plot_deconv(upper_ppm=0.9, lower_ppm=0.2)
#fid_array.plot_deconv_array(upper_ppm=6, lower_ppm=3)

integrals = fid_array.deconvoluted_integrals.transpose()

g6p = integrals[0] + integrals[1]
f6p = integrals[2]
tep = integrals[3]

#scale species by internal standard tep at 5 mM
g6p = 5.0*g6p/tep.mean()
f6p = 5.0*f6p/tep.mean()
tep = 5.0*tep/tep.mean()

species = {'g6p': g6p,
           'f6p': f6p,
           'tep': tep}

fig = plt.figure()
ax = fig.add_subplot(111)
for k, v in species.items():
    ax.plot(fid_array.t, v, label=k)
ax.set_xlabel('min')
ax.set_ylabel('mM')
ax.legend(loc=0, frameon=False)
plt.show()

#fid_array.save_to_file(filename='fidarray.nmrpy')
#fid_array = nmrpy.from_path(fid_path='fidarray.nmrpy')
```

CHAPTER

FOUR

BASIC DATA OBJECTS

class nmrpy.data_objects.**Fid**(**args*, ***kwargs*)

The basic FID (Free Induction Decay) class contains all the data for a single spectrum (*data*), and the necessary methods to process these data.

baseline_correct(*deg=2*)

Perform baseline correction by fitting specified baseline points (stored in _bl_ppm) with polynomial of specified degree (stored in _bl_ppm) and subtract this polynomial from *data*.

Parameters **deg** – degree of fitted polynomial

baseliner()

Instantiate a baseline-correction GUI widget. Right-click-dragging defines a range. Ctrl-Right click deletes previously selected range. Indices selected are stored in _bl_ppm, which is used for baseline-correction (see baseline_correction()).

calibrate()

Instantiate a GUI widget to select a peak and calibrate spectrum. Left-clicking selects a peak. The user is then prompted to enter the PPM value of that peak for calibration.

data

The spectral data. This is the primary object upon which the processing and analysis functions work.

deconv(*method='leastsq'*, *frac_gauss=0.0*)

Deconvolute data object by fitting a series of peaks to the spectrum. These peaks are generated using the parameters in *peaks*. *ranges* splits *data* up into smaller portions. This significantly speeds up deconvolution time.

Parameters

- **frac_gauss** – (0-1) determines the Gaussian fraction of the peaks. Setting this argument to None will fit this parameter as well.
- **method** – The fitting method to use. Default is 'leastsq', the Levenberg-Marquardt algorithm, which is usually sufficient. Additional options include:

  Nelder-Mead (nelder)

  L-BFGS-B (l-bfgs-b)

  Conjugate Gradient (cg)

  Powell (powell)

  Newton-CG (newton)

deconvoluted_integrals

An array of integrals for each deconvoluted peak.

emhz (*lb=5.0*)
Apply exponential line-broadening to data array *data*.

Parameters **lb** – degree of line-broadening in Hz.

classmethod from_data (*data*)
Instantiate a new *Fid* object by providing a spectral data object as argument. Eg.

```
fid = Fid.from_data(data)
```

ft ()
Fourier Transform the data array *data*.

Calculates the Discrete Fourier Transform using the Fast Fourier Transform algorithm as implemented in NumPy (*Cooley, James W., and John W. Tukey, 1965, 'An algorithm for the machine calculation of complex Fourier series,' Math. Comput. 19: 297-301.*)

peakpick (*thresh=0.1*)
Attempt to automatically identify peaks. Picked peaks are assigned to *peaks*.

Parameters **thresh** – fractional threshold for peak-picking

peakpicker ()
Instantiate a peak-picking GUI widget. Left-clicking selects a peak. Right-click-dragging defines a range. Ctrl-left click deletes nearest peak; ctrl-right click deletes range. Peaks are stored in *peaks*; ranges are stored in *ranges*: both are used for deconvolution (see *deconv()*).

peaks
Picked peaks for deconvolution of *data*.

phase_correct (*method='leastsq'*)
Automatically phase-correct *data* by minimising total absolute area.

Parameters **method** – The fitting method to use. Default is 'leastsq', the Levenberg-Marquardt algorithm, which is usually sufficient. Additional options include:

Nelder-Mead (nelder)

L-BFGS-B (l-bfgs-b)

Conjugate Gradient (cg)

Powell (powell)

Newton-CG (newton)

phaser ()
Instantiate a phase-correction GUI widget which applies to *data*.

plot_deconv (***kwargs*)
Plot *data* with deconvoluted peaks overlaid.

Parameters

- **upper_ppm** – upper spectral bound in ppm
- **lower_ppm** – lower spectral bound in ppm
- **lw** – linewidth of plot
- **colour** – colour of the plot
- **peak_colour** – colour of the deconvoluted peaks
- **residual_colour** – colour of the residual signal after subtracting deconvoluted peaks

plot_ppm(***kwargs*)
Plot *data*.

Parameters

- **upper_ppm** – upper spectral bound in ppm
- **lower_ppm** – lower spectral bound in ppm
- **lw** – linewidth of plot
- **colour** – colour of the plot

ps(*p0=0.0*, *p1=0.0*)
Linear phase correction of *data*

Parameters

- **p0** – Zero order phase in degrees
- **p1** – First order phase in degrees

ranges
Picked ranges for deconvolution of *data*.

real()
Discard imaginary component of *data*.

zf()
Apply a single degree of zero-filling to data array *data*.

Note: extends data to double length by appending zeroes. This results in an artificially increased resolution once Fourier-transformed.

class nmrpy.data_objects.**FidArray**(**args*, ***kwargs*)
This object collects several *Fid* objects into an array, and it contains all the processing methods necessary for bulk processing of these FIDs. It should be considered the parent object for any project. The class methods *from_path()* and *from_data()* will instantiate a new *FidArray* object from a Varian/Bruker .fid path or an iterable of data respectively. Each *Fid* object in the array will appear as an attribute of *FidArray* with a unique ID of the form 'fidXX', where 'XX' is an increasing integer .

add_fid(*fid*)
Add an *Fid* object to this *FidArray*, using a unique id.

Parameters **fid** – an *Fid* instance

add_fids(*fids*)
Add a list of *Fid* objects to this *FidArray*.

Parameters **fids** – a list of *Fid* instances

baseline_correct_fids(*deg=2*)
Apply baseline-correction to all *Fid* objects owned by this *FidArray*

Parameters **deg** – degree of the baseline polynomial (see *baseline_correct()*)

baseliner_fids()
Instantiate a baseline-correction GUI widget. Right-click-dragging defines a range. Ctrl-Right click deletes previously selected range. Indices selected are stored in `_bl_ppm`, which is used for baseline-correction (see `baseline_correction()`).

calibrate(*fid_number=None*, *assign_only_to_index=False*, *voff=0.02*)
Instantiate a GUI widget to select a peak and calibrate spectra in a *FidArray*. Left-clicking selects a peak. The user is then prompted to enter the PPM value of that peak for calibration; this will be applied to all *Fid* objects owned by this *FidArray*. See also *calibrate()*.

Parameters **fid_number** – list or number, index of

Fid to use for calibration. If None, the whole data array is plotted.

Parameters **assign_only_to_index** – if True, assigns calibration only

to *Fid* objects indexed by fid_number; if False, assigns to all.

Parameters **voff** – vertical offset for spectra

data
An array of all *data* objects belonging to the *Fid* objects owned by this *FidArray*.

deconv_fids(*mp=True*, *cpus=None*, *method='leastsq'*, *frac_gauss=0.0*)
Apply deconvolution to all *Fid* objects owned by this *FidArray*, using the *peaks* and *ranges* attribute of each respective *Fid*.

Parameters

- **method** – see *phase_correct()*
- **mp** – parallelise the phasing process over multiple processors, significantly reduces computation time
- **cpus** – defines number of CPUs to utilise if 'mp' is set to True, default is n-1 cores

deconvoluted_integrals
Collected *deconvoluted_integrals*

del_fid(*fid_id*)
Delete an *Fid* object belonging to this *FidArray*, using a unique id.

Parameters **fid_id** – a string id for an *Fid*

emhz_fids(*lb=5.0*)
Apply line-broadening (apodisation) to all `nmrpy.~data_objects.Fid` objects owned by this *FidArray*

Parameters **lb** – degree of line-broadening in Hz.

classmethod from_data(*data*)
Instantiate a new *FidArray* object from a 2D data set of spectral arrays.

Parameters **data** – a 2D data array

classmethod from_path(*fid_path='.'*, *file_format=None*, *arrayset=None*)
Instantiate a new *FidArray* object from a .fid directory.

Parameters

- **fid_path** – filepath to .fid directory
- **file_format** – 'varian' or 'bruker', usually unnecessary
- **arrayset** – (int) array set for interleaved spectra, user is prompted if not specified

ft_fids(*mp=True*, *cpus=None*)
Fourier-transform all FIDs.

Parameters

- **mp** – parallelise over multiple processors, significantly reducing computation time
- **cpus** – defines number of CPUs to utilise if 'mp' is set to True

get_fid(*id*)
Return an *Fid* object owned by this object, identified by unique ID. Eg.:

```
fid12 = fid_array.get_fid('fid12')
```

Parameters **id** – a string id for an *Fid*

get_fids()
Return a list of all *Fid* objects owned by this *FidArray*.

get_integrals_from_traces()
Returns a dictionary of integral values for all *Fid* objects calculated from trace dictionary *integral_traces*.

get_masked_integrals()
After peakpicker_traces() and deconv_fids() this function returns a masked integral array.

integral_traces
Returns the dictionary of integral traces generated by `select_integral_traces()`.

norm_fids()
Normalise FIDs by maximum data value in *data*.

peakpicker(*fid_number=None*, *assign_only_to_index=True*, *voff=0.02*)
Instantiate peak-picker widget for *data*, and apply selected *peaks* and *ranges* to all *Fid* objects owned by this *FidArray*. See *peakpicker()*.

Parameters **fid_number** – list or number, index of

Fid to use for peak-picking. If None, data array is plotted.

Parameters **assign_only_to_index** – if True, assigns selections only

to *Fid* objects indexed by fid_number, if False, assigns to all

Parameters **voff** – vertical offset for spectra

peakpicker_traces(*voff=0.02*, *lw=1*)
Instantiates a widget to pick peaks and ranges employing a polygon shape (or 'trace'). This is useful for picking peaks that are subject to drift and peaks that appear (or disappear) during the course of an experiment.

Parameters

- **voff** – vertical offset fraction (0.01)
- **lw** – linewidth of plot (1)

phase_correct_fids(*method='leastsq'*, *mp=True*, *cpus=None*)
Apply automatic phase-correction to all *Fid* objects owned by this *FidArray*

Parameters

- **method** – see *phase_correct()*
- **mp** – parallelise the phasing process over multiple processors, significantly reducing computation time
- **cpus** – defines number of CPUs to utilise if 'mp' is set to True

plot_array(***kwargs*)
Plot *data*.

Parameters

- **upper_index** – upper index of array (None)

- **lower_index** – lower index of array (None)
- **upper_ppm** – upper spectral bound in ppm (None)
- **lower_ppm** – lower spectral bound in ppm (None)
- **lw** – linewidth of plot (0.5)
- **azim** – starting azimuth of plot (-90)
- **elev** – starting elevation of plot (40)
- **filled** – True=filled vertices, False=lines (False)
- **show_zticks** – show labels on z axis (False)
- **labels** – under development (None)
- **colour** – plot spectra with colour spectrum, False=black (True)
- **filename** – save plot to .pdf file (None)

plot_deconv_array (***kwargs*)
Plot all *data* with deconvoluted peaks overlaid.

Parameters

- **upper_index** – upper index of Fids to plot
- **lower_index** – lower index of Fids to plot
- **upper_ppm** – upper spectral bound in ppm
- **lower_ppm** – lower spectral bound in ppm
- **data_colour** – colour of the plotted data ('k')
- **summed_peak_colour** – colour of the plotted summed peaks ('r')
- **residual_colour** – colour of the residual signal after subtracting deconvoluted peaks ('g')
- **data_filled** – fill state of the plotted data (False)
- **summed_peak_filled** – fill state of the plotted summed peaks (True)
- **residual_filled** – fill state of the plotted residuals (False)
- **figsize** – [x, y] size of plot ([15, 7.5])
- **lw** – linewidth of plot (0.3)
- **azim** – azimuth of 3D axes (-90)
- **elev** – elevation of 3D axes (20)

ps_fids (*p0=0.0*, *p1=0.0*)
Apply manual phase-correction to all *Fid* objects owned by this *FidArray*

Parameters

- **p0** – Zero order phase in degrees
- **p1** – First order phase in degrees

real_fids ()
Discard imaginary component of FID data sets.

save_to_file (*filename=None*)
Save *FidArray* object to file, including all objects owned.

Parameters **filename** – filename to save *FidArray* to

select_integral_traces(*voff=0.02*, *lw=1*)
Instantiate a trace-selection widget to identify deconvoluted peaks. This can be useful when data are subject to drift. Selected traces on the data array are translated into a set of nearest deconvoluted peaks, and saved in a dictionary: *integral_traces*.

Parameters

- **voff** – vertical offset fraction (0.01)
- **lw** – linewidth of plot (1)

t
An array of the acquisition time for each FID.

zf_fids()
Zero-fill all *Fid* objects owned by this *FidArray*

CHAPTER

FIVE

PLOTTING OBJECTS

class nmrpy.plotting.**Calibrator**(*fid*, *lw=1*, *label=None*, *title=None*)
Interactive data-selection widget for calibrating PPM of a spectrum.

class nmrpy.plotting.**DataPeakRangeSelector**(*fid_array*, *peaks=None*, *ranges=None*, *y_indices=None*, *aoti=True*, *voff=0.001*, *lw=1*, *label=None*)
Interactive data-selection widget with lines and ranges. Lines and spans are saved as self.peaks, self.ranges.

class nmrpy.plotting.**DataPeakSelector**(*fid*, *peaks=None*, *ranges=None*, *voff=0.001*, *lw=1*, *label=None*, *title=None*)
Interactive data-selection widget with lines and ranges for a single Fid. Lines and spans are saved as self.peaks, self.ranges.

class nmrpy.plotting.**DataSelector**(*data*, *params*, *extra_data=None*, *extra_data_colour='k'*, *peaks=None*, *ranges=None*, *title=None*, *voff=0.001*, *label=None*)

Interactive selector widget. can inherit from various mixins for functionality: Line selection: `LineSelectorMixin` Span selection: `SpanSelectorMixin` Poly selection: `PolySelectorMixin`

This class is not intended to be used without inheriting at least one mixin.

class nmrpy.plotting.**DataTraceRangeSelector**(*fid_array*, *peaks=None*, *ranges=None*, *voff=0.001*, *lw=1*, *label=None*)
Interactive data-selection widget with traces and ranges. Traces are saved as self.data_traces (WRT data) and self.index_traces (WRT index). Spans are saves as self.spans.

class nmrpy.plotting.**DataTraceSelector**(*fid_array*, *extra_data=None*, *extra_data_colour='b'*, *voff=0.001*, *lw=1*, *label=None*)
Interactive data-selection widget with traces and ranges. Traces are saved as self.data_traces (WRT data) and self.index_traces (WRT index).

class nmrpy.plotting.**FidArrayRangeSelector**(*fid_array*, *ranges=None*, *y_indices=None*, *voff=0.001*, *lw=1*, *title=None*, *label=None*)
Interactive data-selection widget with ranges. Spans are saved as self.ranges.

class nmrpy.plotting.**FidRangeSelector**(*fid*, *title=None*, *ranges=None*, *y_indices=None*, *voff=0.001*, *lw=1*, *label=None*)
Interactive data-selection widget with ranges. Spans are saved as self.ranges.

class nmrpy.plotting.**IntegralDataSelector**(*data*, *params*, *extra_data=None*, *extra_data_colour='k'*, *peaks=None*, *ranges=None*, *title=None*, *voff=0.001*, *label=None*)

class nmrpy.plotting.**LineSpanDataSelector**(*data*, *params*, *extra_data=None*, *extra_data_colour='k'*, *peaks=None*, *ranges=None*, *title=None*, *voff=0.001*, *label=None*)

class nmrpy.plotting.**PeakDataSelector**(*data*, *params*, *extra_data=None*, *extra_data_colour='k'*, *peaks=None*, *ranges=None*, *title=None*, *voff=0.001*, *label=None*)

class nmrpy.plotting.**PeakTraceDataSelector**(*data*, *params*, *extra_data=None*, *extra_data_colour='k'*, *peaks=None*, *ranges=None*, *title=None*, *voff=0.001*, *label=None*)

class nmrpy.plotting.**Phaser**(*fid*)
Interactive phase-correction widget

class nmrpy.plotting.**Plot**
Basic 'plot' class containing functions for various types of plots.

class nmrpy.plotting.**RangeCalibrator**(*fid_array*, *y_indices=None*, *aoti=True*, *voff=0.001*, *lw=1*, *label=None*)
Interactive data-selection widget for calibrating PPM of an array of spectra.

class nmrpy.plotting.**SpanDataSelector**(*data*, *params*, *extra_data=None*, *extra_data_colour='k'*, *peaks=None*, *ranges=None*, *title=None*, *voff=0.001*, *label=None*)

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

n

INDEX

P

R

S

T

Z