A User's Guide

The DAMARIS Script Library

Version 1.0, June 2015

Compiled by Oleg V. Petrov

AG Vogel @ Technische Universität Darmstadt

The DAMARIS Script Library: A User's Guide

v1.0 June 2015

By Oleg V. Petrov

The group of Prof. Michael Vogel @ Technical University of Darmstadt

Please report comments and errors to: `oleg@nmr.physik.tu-darmstadt.de`

Contents

1. Introduction

This manual describes pulse programs and basic parameter setups for the NMR experiments compiled in The DAMARIS Script Library. The library is intended primary to comply with the tasks performed in the group of Prof. Michael Vogel's at TUD, though it might be of interest to other groups carrying out like experiments. It currently includes 16 scripts (by June 2015), beginning with the basic pulse-acquire sequence and ending with pseudo-2D experiments to measure kinetics under slow exchange (Table 1).

The very concept of DAMARIS and how to program with this NMR software can be found at `element.fkp.physik.tu-darmstadt.de/damaris_cms/`. Briefly saying, to set up an NMR experiment with DAMARIS means to provide two scripts. One is called an experiment script (the module `*_exp.py`). It accommodates in turn two functions: one has a pre-defined name `experiment()` and serves to drive a pulse program; the other, with an arbitrarily name, e.g. `fid_experiment()`, defines the pulse program which is called from within `experiment()`. Certainly, the `*_exp.py` module may include more functions, but these two are usually all what you need to run a particular experiment. The second script is called a result script (the module `*_res.py`). It provides processing routines for the incoming signal and displays the signal and whatever measurements performed on it. It includes a function with pre-defined name `result()` and may hold some auxiliary functions, e.g. for data fitting and signal transforms. The scripts are written in the Python programming language (hence the extension `.py`), using DAMARIS front-end classes (`Experiment`, `ADCResult`, `Accumulation`, `MeasurementResult`, etc.).

The experiment and result modules are loaded in separate DAMARIS' tabs. Communication between them is implemented via a parameter description routine. Namely, a dictionary consisting parameter's key-and-value pairs is created within the experiment script by the command `set_description` and available in the result script by the `get_description_dictionary` command. Also, there are number of 'getters' implemented as DAMARIS classes' methods to gain access to specific signal's attributes (such as the actual sampling rate) and signal's bounds.

The DAMARIS framework has much in common with Bruker's Minispec where the whole NMR experiment, from setting acquisition parameters to measuring on the incoming signal, is programmed in one script. Who has experience with Minispec will therefore find himself amid familiar surroundings when working with DAMARIS. It may seem foreign, however, to those who are accustomed to Bruker's XWINNMR/TopSpin or Oxford Instruments' RINMR software, where setting parameters, programming pulse sequences and writing AU-programs for automatic performance are done separately and take different places. To provide a certain degree of consistency, I put emphasis in the present DAMARIS scripts on those elements that resemble the way that commercial NMR software is organized. For example, although DAMARIS does not use such a thing as a parameter table, the scripts share a common parameter namespace throughout, all the parameters being stored in one variable `pars` and specified in one place in the beginning of `experiment()`. For another example, all pulse programs are designed to be self-sufficient in the sense that they are independent from their driver (the `experiment()` function) as long as an appropriate `pars` dictionary is provided and that they are not intended to be modified by the user in the course of experiment.

In the next chapter, I describe a general plan of the scripts and those elements that they have in common. Then each script will be discussed particularly, and results of test measurements will be provided.

Table 1. A list of scripts

Folder's name	*Scripts' names*	*Comments*
CPMG	`op_cpmg_exp.py` `op_cpmg_res.py`	Carr-Purcell-Meiboom-Gill sequence, to measure T_2
FID	`op_fid_exp.py` `op_fid_res.py`	The basic pulse-acquire experiment
FID_with_Background_Suppression	`op_zgbs_exp.py` `op_zgbs_res.py`	The pulse-acquisition with a background signal reduction
Hahn Echo	`op_hahn_exp.py` `op_hahn_res.py`	90_x-180_x spin-echo
Miscellaneous	`op_gs_exp.py` `op_gs_res.py`	The pulse-acquisition in a loop (no accumulation)
Satuaration_Recovery	`op_satrec_exp.py` `op_satrec_res.py`	Saturation-recovery experiment, to measure T_1
Saturation_Recovery_with_Solid_Echo_Detection	`op_satrec2_exp.py` `op_satrec2_res.py`	Saturation-recovery with SE detection (for short FID's)
Solid_Echo	`op_solidecho_exp.py` `op_solidecho_res.py`	90_x-90_y spin-echo
Spin_Alignment	`op_spinal_exp.py` `op_spinal_res.py`	Spin-echo after quadrupolar order during t_m (for spins-1)
Spin_Alignment_Spin32	`op_spinal32_exp.py` `op_spinal32_res.py`	Spin-echo after quadrupolar order during t_m (for spins-3/2)
Steady_Gradient_Spin_Echo	`op_sgse_exp.py` `op_sgse_res.py`	STE-based diffusiometry in a steady gradient (SGSE)
Steady_Gradient_Spin_Echo_with_CPMG_Detection	`op_sgse2_exp.py` `op_sgse2_res.py`	SGSE with CPMG readout (to enhance SNR)
Stimulated_Echo	`op_ste_exp.py` `op_ste_res.py`	The basic STE experiment
T1Q	`op_t1q_exp.py` `op_t1q_res.py`	To measure a quadrupolar order relaxation (T_{1Q})
Zeeman_Order	`op_zeeman_exp.py` `op_zeeman_res.py`	Spin-echo after Zeeman order during t_m (spin-1)
ZZ_Exchange_with_T2_Selection	`op_t2zz_exp.py` `op_t2zz_res.py`	Z-magnetization build-up after T_2 relaxation via chemical exchange

2. The script layout

2.1. Experiment scripts

In the very beginning of an experiment script, I specify hardware-related parameters such as TTL-lines that control a RF transmitter, the transmitter enabling delay, and ADC sensitivity (Fig. 1, lines 3-6). Since these parameters relate to the NMR hardware configuration rather than to the NMR experiment proper, I have chosen to place them outside the `experiment()` function. Most likely, this is the only part of the script that might have to be changed when porting between spectrometers.

Next comes the function `experiment()` with acquisition parameters initialization in first few lines (Fig. 1, lines 12-24). In the example script, there are 13 such parameters: 11 for running a pulse program and 2 to specify whether and where to save data. All acquisition parameters are stored in the dictionary `pars` under names (mainly borrowed from commercial NMR systems) written in upper-case to distinguish them from other scripts' variables. To add a new parameter, say `XYZ`, you simply type `pars['XYZ']=value`. You can choose one parameter to be variable to allow for so-called arrayed experiment (as in Varian-Chemagnetic's Spinsight). In the example script, the parameter `D2` is chosen to vary from 30 us to 2 s through the array of 24 log-spaced values (lines 27-32).

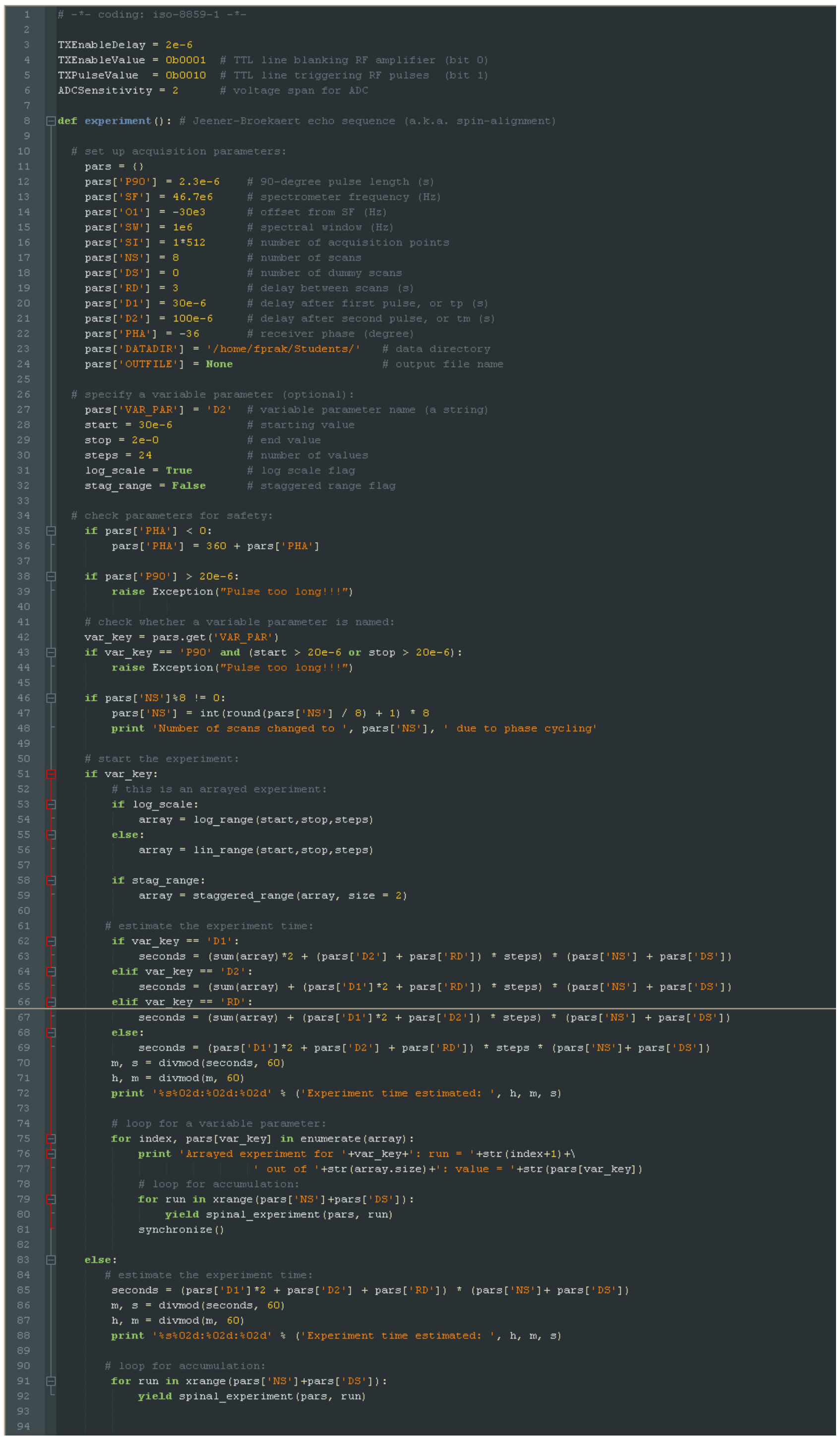
Next, the `pars` values are checked for safety (lines 35-48). In particular, one pays attention to the r.f. pulse length (don't go burn anything) and makes sure that the number of scans is a multiple of the number of steps in the phase cycle used.

Then it comes to calling a pulse-program function. The way it is called depends on whether a variable parameter has been named. If yes (meaning it is an arrayed experiment), the pulse-program function is called in two nested loops. The inner loop is for signal accumulation and it runs the number of scans specified by the values `NS` plus `DS`, all parameters being fixed except for pulse phases (see below). The outer loop runs for a variable parameter. The latter takes on values preliminary calculated and arrayed according to the variable parameter settings (Fig. 1, lines 51-59). If no variable parameter is named (meaning it is a one-time experiment with all-fixed parameters), only the loop for signal accumulation is run. In either case, the total experiment time is calculated and output in the log tab (Fig. 1, lines 62-72, 85-88).

The pulse-program function (named `spinal_experiment()` in the example script) takes two arguments. One is `pars` and the other is a current `run` of the accumulation loop. In the beginning of the function, a DAMARIS' `Experiment` object is created (Fig.1, line 97). Methods of this class serve as hardware configuration commands in the pulse sequence (Fig. 1, lines 134-154). Prior to applying the commands, one works a little longer on acquisition parameters. Thus, to deal with dummy scans (those which are not for accumulation), the `DS` value is subtracted from `run`, so that the signal is not accumulated (on the result script's side) until the `run` become non-negative again. Here one also specifies phase lists for r.f. pulses and a receiver, which will complete the parameters setup (lines 106-109). The phase lists are the property of the pulse program and, as such, are not intended to be modified during experiment. For this reason, they are separated from the rest of `pars` set in the `experiment()`. Note that the name `PH2` is reserved for the receiver phase list.

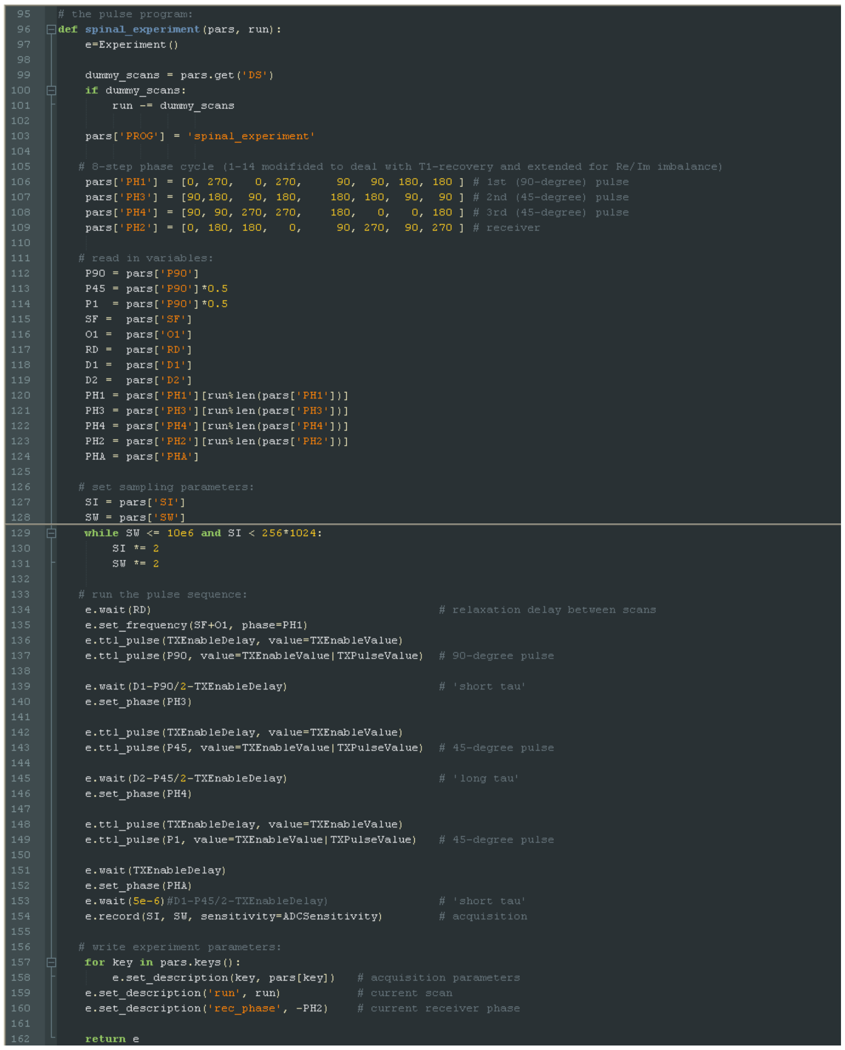
The `pars` are read in local variables before passing to the pulse sequence commands (lines 112-124), for two reasons. First, aesthetic, is to avoid bulky expressions with constructions like `pars['XYZ']` in the commands. Second is to calculate dependent parameters and adjust them for optimum performance. Thus, in lines 120-123, the phase lists are indexed according to the current `run`. And in lines 127-131, the ADC sampling rate and the number of samples are maximized, in synchrony, for digital filtering purposes (see below).

After the pulse sequence commands, all `pars`, the current `run`, and the receiver phase `rec_phase` are written in the parameter description dictionary and passed to a result script (lines 157-160). In the end, the pulse-program function returns the configured `Experiment` object to DAMARIS for execution (line 162).



```python
# -*- coding: iso-8859-1 -*-

TXEnableDelay = 2e-6
TXEnableValue = 0b0001  # TTL line blanking RF amplifier (bit 0)
TXPulseValue  = 0b0010  # TTL line triggering RF pulses  (bit 1)
ADCSensitivity = 2      # voltage span for ADC

def experiment(): # Jeener-Broekaert echo sequence (a.k.a. spin-alignment)

  # set up acquisition parameters:
    pars = {}
    pars['P90'] = 2.3e-6    # 90-degree pulse length (s)
    pars['SF'] = 46.7e6     # spectrometer frequency (Hz)
    pars['O1'] = -30e3      # offset from SF (Hz)
    pars['SW'] = 1e6        # spectral window (Hz)
    pars['SI'] = 1*512      # number of acquisition points
    pars['NS'] = 8          # number of scans
    pars['DS'] = 0          # number of dummy scans
    pars['RD'] = 3          # delay between scans (s)
    pars['D1'] = 30e-6      # delay after first pulse, or tp (s)
    pars['D2'] = 100e-6     # delay after second pulse, or tm (s)
    pars['PHA'] = -36       # receiver phase (degree)
    pars['DATADIR'] = '/home/fprak/Students/'   # data directory
    pars['OUTFILE'] = None                      # output file name

  # specify a variable parameter (optional):
    pars['VAR_PAR'] = 'D2'  # variable parameter name (a string)
    start = 30e-6           # starting value
    stop = 2e-0             # end value
    steps = 24              # number of values
    log_scale = True        # log scale flag
    stag_range = False      # staggered range flag

  # check parameters for safety:
    if pars['PHA'] < 0:
        pars['PHA'] = 360 + pars['PHA']

    if pars['P90'] > 20e-6:
        raise Exception("Pulse too long!!!")

    # check whether a variable parameter is named:
    var_key = pars.get('VAR_PAR')
    if var_key == 'P90' and (start > 20e-6 or stop > 20e-6):
        raise Exception("Pulse too long!!!")

    if pars['NS']%8 != 0:
        pars['NS'] = int(round(pars['NS'] / 8) + 1) * 8
        print 'Number of scans changed to ', pars['NS'], ' due to phase cycling'

    # start the experiment:
    if var_key:
        # this is an arrayed experiment:
        if log_scale:
            array = log_range(start,stop,steps)
        else:
            array = lin_range(start,stop,steps)

        if stag_range:
            array = staggered_range(array, size = 2)

       # estimate the experiment time:
        if var_key == 'D1':
            seconds = (sum(array)*2 + (pars['D2'] + pars['RD']) * steps) * (pars['NS'] + pars['DS'])
        elif var_key == 'D2':
            seconds = (sum(array) + (pars['D1']*2 + pars['RD']) * steps) * (pars['NS'] + pars['DS'])
        elif var_key == 'RD':
            seconds = (sum(array) + (pars['D1']*2 + pars['D2']) * steps) * (pars['NS'] + pars['DS'])
        else:
            seconds = (pars['D1']*2 + pars['D2'] + pars['RD']) * steps * (pars['NS']+ pars['DS'])
        m, s = divmod(seconds, 60)
        h, m = divmod(m, 60)
        print '%s%02d:%02d:%02d' % ('Experiment time estimated: ', h, m, s)

        # loop for a variable parameter:
        for index, pars[var_key] in enumerate(array):
            print 'Arrayed experiment for '+var_key+': run = '+str(index+1)+\
                                 ' out of '+str(array.size)+': value = '+str(pars[var_key])
            # loop for accumulation:
            for run in xrange(pars['NS']+pars['DS']):
                yield spinal_experiment(pars, run)
            synchronize()

    else:
       # estimate the experiment time:
        seconds = (pars['D1']*2 + pars['D2'] + pars['RD']) * (pars['NS']+ pars['DS'])
        m, s = divmod(seconds, 60)
        h, m = divmod(m, 60)
        print '%s%02d:%02d:%02d' % ('Experiment time estimated: ', h, m, s)

       # loop for accumulation:
        for run in xrange(pars['NS']+pars['DS']):
            yield spinal_experiment(pars, run)
```

Fig. 1. The experiment script op_spinal_exp.py (the spin-alignment experiment)



```python
# the pulse program:
def spinal_experiment(pars, run):
    e=Experiment()

    dummy_scans = pars.get('DS')
    if dummy_scans:
        run -= dummy_scans

    pars['PROG'] = 'spinal_experiment'

    # 8-step phase cycle (1-14 modifided to deal with T1-recovery and extended for Re/Im imbalance)
    pars['PH1'] = [0, 270,   0, 270,     90,  90, 180, 180 ] # 1st (90-degree) pulse
    pars['PH3'] = [90,180,  90, 180,    180, 180,  90,  90 ] # 2nd (45-degree) pulse
    pars['PH4'] = [90, 90, 270, 270,    180,   0,   0, 180 ] # 3rd (45-degree) pulse
    pars['PH2'] = [0, 180, 180,   0,     90, 270,  90, 270 ] # receiver

    # read in variables:
    P90 = pars['P90']
    P45 = pars['P90']*0.5
    P1  = pars['P90']*0.5
    SF =  pars['SF']
    O1 =  pars['O1']
    RD =  pars['RD']
    D1 =  pars['D1']
    D2 =  pars['D2']
    PH1 = pars['PH1'][run%len(pars['PH1'])]
    PH3 = pars['PH3'][run%len(pars['PH3'])]
    PH4 = pars['PH4'][run%len(pars['PH4'])]
    PH2 = pars['PH2'][run%len(pars['PH2'])]
    PHA = pars['PHA']

    # set sampling parameters:
    SI = pars['SI']
    SW = pars['SW']
    while SW <= 10e6 and SI < 256*1024:
        SI *= 2
        SW *= 2

    # run the pulse sequence:
    e.wait(RD)                                          # relaxation delay between scans
    e.set_frequency(SF+O1, phase=PH1)
    e.ttl_pulse(TXEnableDelay, value=TXEnableValue)
    e.ttl_pulse(P90, value=TXEnableValue|TXPulseValue)  # 90-degree pulse

    e.wait(D1-P90/2-TXEnableDelay)                      # 'short tau'
    e.set_phase(PH3)

    e.ttl_pulse(TXEnableDelay, value=TXEnableValue)
    e.ttl_pulse(P45, value=TXEnableValue|TXPulseValue)  # 45-degree pulse

    e.wait(D2-P45/2-TXEnableDelay)                      # 'long tau'
    e.set_phase(PH4)

    e.ttl_pulse(TXEnableDelay, value=TXEnableValue)
    e.ttl_pulse(P1, value=TXEnableValue|TXPulseValue)   # 45-degree pulse

    e.wait(TXEnableDelay)
    e.set_phase(PHA)
    e.wait(5e-6)#D1-P45/2-TXEnableDelay)                # 'short tau'
    e.record(SI, SW, sensitivity=ADCSensitivity)        # acquisition

    # write experiment parameters:
    for key in pars.keys():
        e.set_description(key, pars[key])   # acquisition parameters
    e.set_description('run', run)           # current scan
    e.set_description('rec_phase', -PH2)    # current receiver phase

    return e
```

Fig. 1 (cntd). The experiment script op_spinal_exp.py (the spin-alignment experiment)

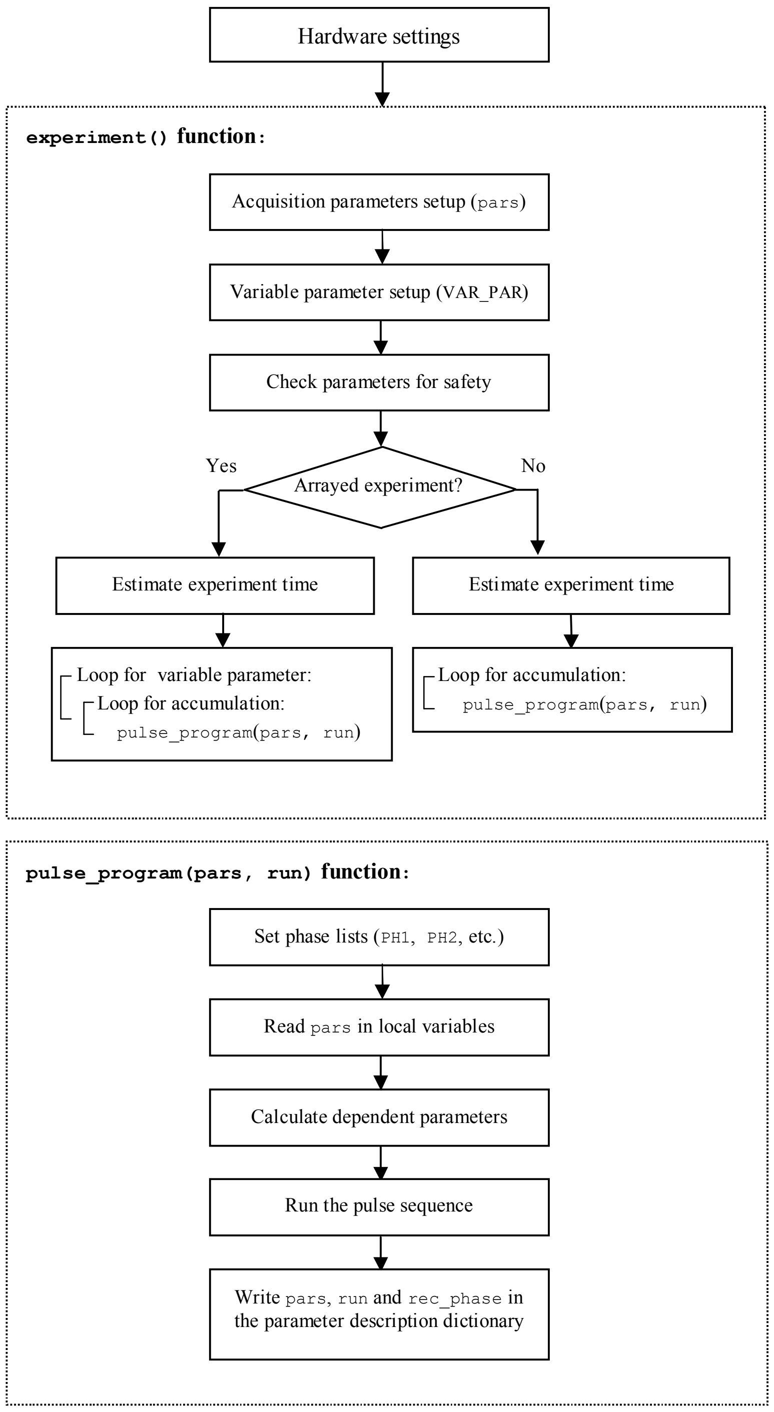

Hardware settings
experiment() function:
Acquisition parameters setup (pars)
Variable parameter setup (VAR_PAR)
Check parameters for safety
Yes
Arrayed experiment?
No
Estimate experiment time
Estimate experiment time
Loop for variable parameter:
Loop for accumulation:
pulse_program(pars, run)
Loop for accumulation:
pulse_program(pars, run)
pulse_program(pars, run) function:
Set phase lists (PH1, PH2, etc.)
Read pars in local variables
Calculate dependent parameters
Run the pulse sequence
Write pars, run and rec_phase in the parameter description dictionary

Fig. 2. Flowchart of an experiment scripts

2.2. Result scripts

The central (and often the only) component of a result script is a pre-defined function `result()`. Its principle job is to extract the incoming signals buffered in `results` into a current-scan container `timesignal` and accumulate them in `accu` after first phasing according to the scan's settings. The settings are available as `timesignal`'s attributes which are read in by `get_description_dictionary()` and other getters (Fig. 2, lines 25 and 30).

Even before phasing, `timesignal` is subject to digital filtering to improve signal-to-noise ratio. The digital filtering is introduced, by default, to all experiments but CPMG. It is realized in three steps. First, the NMR signal is oversampled (on the experiment script's side) to take as much noise as possible out of the spectral window requested by the user (the parameter `SW`) by minimizing Nyquist fold-backs. Then, a low-pass finite impulse response (FIR) filter is applied to remove the noise components above `SW`. Finally, the filtered signal is re-sampled down according to the originally requested `SW`. In commercial NMR systems, digital filtering is implemented on the hardware level by using a signal processor which performs FIR filtering "on the fly" during the signal acquisition. In DAMARIS, it is a post-acquisition procedure. FIR filtering means that each value of the output signal is constructed as a weighed sum of *N* most recent samples of the input signal, where *N* is the filter order. It means, in turn, that very first *N*-1 samples become corrupted compared to the rest of the signal and therefore are to be discarded. To keep the required number of samples (parameter `SI`), I set those *N*-1 samples to zero and shift the whole signal to the left so that the zeros appear in the end of the signal (to be joining zeroes appended in zero-filling procedure if applied). For the default $N = 29$ and the actual dwell time 0.05-0.1 us, the digital filtering consumes first 1.4-2.8 μs of the signal. This consumption (called GroupDelay in RINMR and DEAD2 in XWINNMR/TopSpin) must be accounted for when positioning a signal acquisition onset.

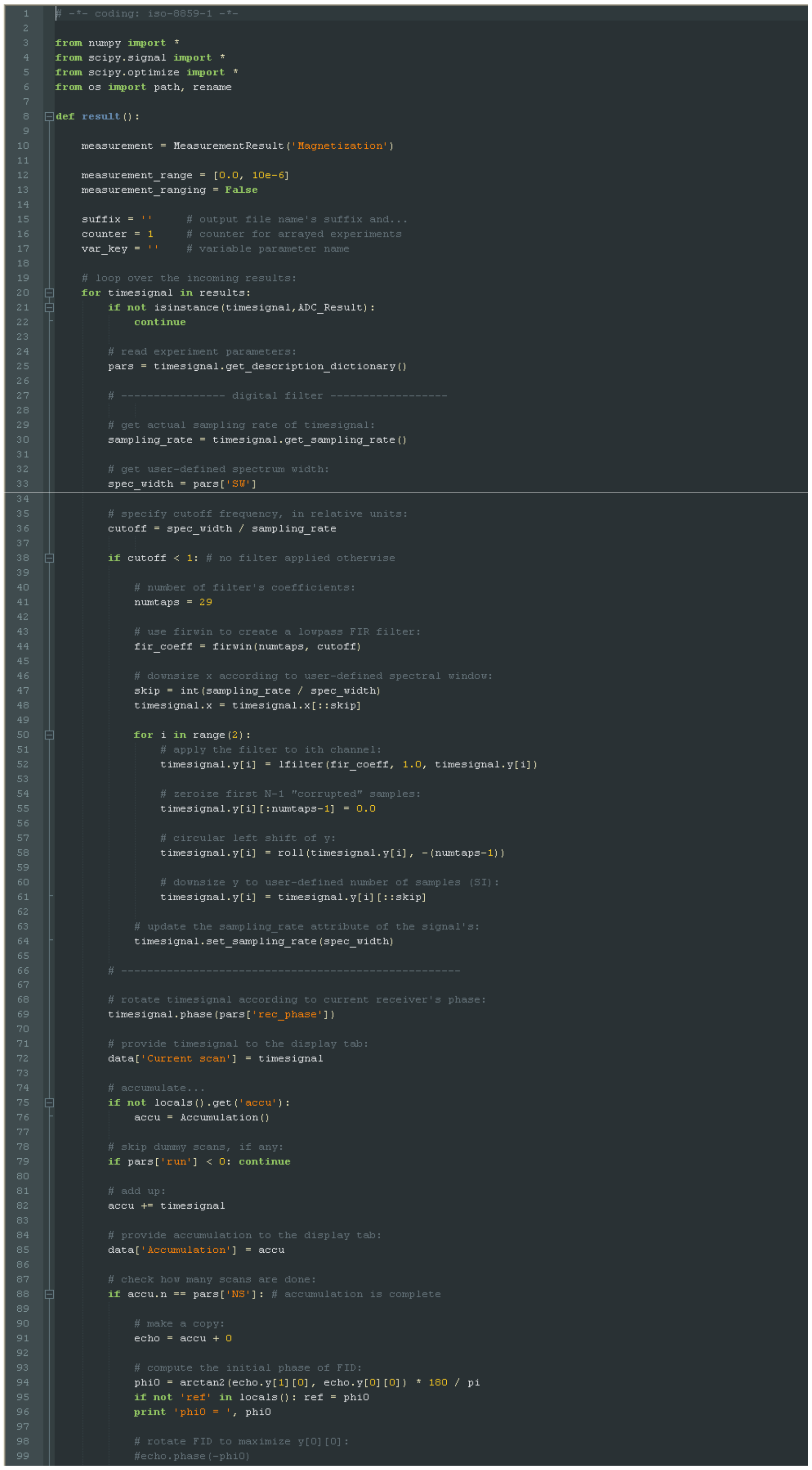
The phasing, or digital rotation, of `timesignal` is controlled by a special parameter `rec_phase` coming amid `pars` from the experiment script. This parameter is a current `PH2` value but taken with an opposite sign (see Fig. 1, line 160). The rotation is performed with a DAMARIS' method `phase()` (Fig. 2, line 69). The phase-rotated `timesignal` is output in the Display tab (line 72) under the name Current Scan.

After being filtered and phased, `timesignal` is accumulated in `accu`. Unlike `measurement`, the `accu` is a local variable which is reset every time when a variable parameter (if any) is updated (lines 75-76 and 164). The `accu` is refreshed in the Display tab after each new scan.

Once all scans requested by the user (parameter `NS+DS`) are done, the script proceeds with signal processing. In most cases, it includes FFT (Fig. 2, line 103) followed by a 'first-order phasing' of the resultant `spectrum` (line 106). To rotate `spectrum`, the initial phase `phi0` of `accu` is calculated (line 94). (The `phi0` value is printed in the Log tab and thus can be used to maximize a Re-signal by its adding to `PHA`.) For the FFT purpose, `accu` is also weighted with exponential window function (line 102) and, following a standard practice, filled with zeroes to double its dimension. If it is an arrayed experiment, the next bit of the script will perform a required measurement on either `accu` or `spectrum` and plot it against a variable parameter. In the example script, the intensity of the Re-component of `accu` is measured against the mixing time `D2` . The intensity is defined as either the sum of samples within the time interval given by `measurement_range` (lines 121-123) or the sum of first 32 samples (line 126).

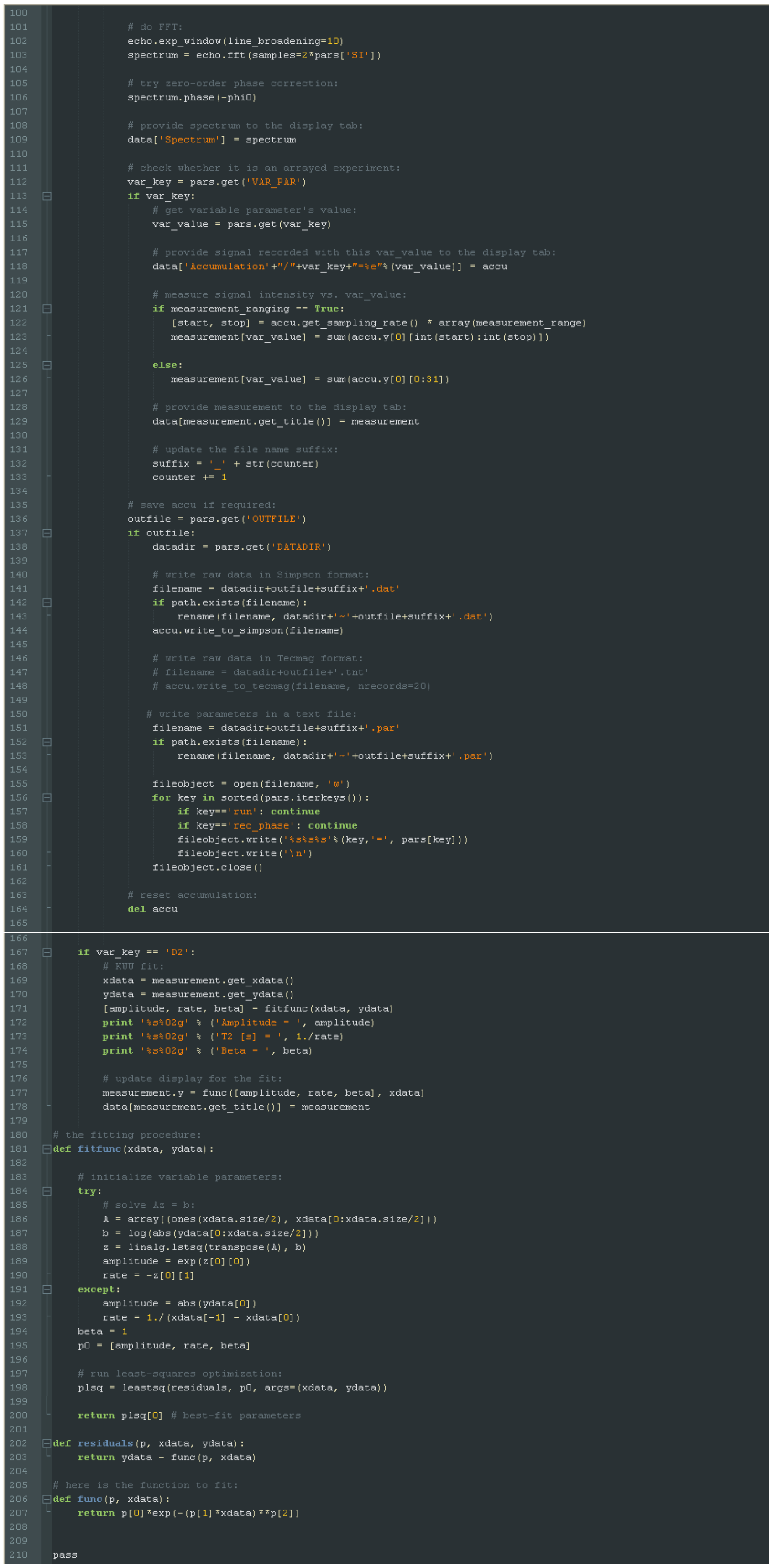
In the end of `result()`, data from `accu` and the acquisition parameters used are written in files, according to the `OUTFILE` value. The acquisition parameters are saved in a text file `*.par`, while for the data there is a choice between a SIMPSON (text) file `*.dat` and a Tecmag (binary) file `*.tnt`. Both formats are readable by NMRnotebook, the NMR processing program installed on our DAMARIS spectrometer PC's. More about data handling is explained in the next section. After the data saving, `accu` is deleted (line 164) but the data remain in the Display tab. The above steps are repeated for each variable parameter's value, the data being saved separately.

Once all variable parameter's values are taken, or the experiment is interrupted by pressing the Stop button, the script continues outside `result()` to perform post-measurement tasks, such as data fitting (lines 167-178). Presently, only mono- or stretched-exponential fit with automatically assigned initial values are implemented. The auxiliary functions for fitting and other possible tasks are placed in the end of the script (lines 181, 202, 206). And necessary Python libraries such as `numpy`, `scipy.signal`, `scipy.optimize`, `os` are inserted at the top of the script (Fig. 2, lines 3-6).



```python
# -*- coding: iso-8859-1 -*-

from numpy import *
from scipy.signal import *
from scipy.optimize import *
from os import path, rename

def result():

    measurement = MeasurementResult('Magnetization')

    measurement_range = [0.0, 10e-6]
    measurement_ranging = False

    suffix = ''      # output file name's suffix and...
    counter = 1      # counter for arrayed experiments
    var_key = ''     # variable parameter name

    # loop over the incoming results:
    for timesignal in results:
        if not isinstance(timesignal,ADC_Result):
            continue

        # read experiment parameters:
        pars = timesignal.get_description_dictionary()

        # ---------------- digital filter ------------------

        # get actual sampling rate of timesignal:
        sampling_rate = timesignal.get_sampling_rate()

        # get user-defined spectrum width:
        spec_width = pars['SW']

        # specify cutoff frequency, in relative units:
        cutoff = spec_width / sampling_rate

        if cutoff < 1: # no filter applied otherwise

            # number of filter's coefficients:
            numtaps = 29

            # use firwin to create a lowpass FIR filter:
            fir_coeff = firwin(numtaps, cutoff)

            # downsize x according to user-defined spectral window:
            skip = int(sampling_rate / spec_width)
            timesignal.x = timesignal.x[::skip]

            for i in range(2):
                # apply the filter to ith channel:
                timesignal.y[i] = lfilter(fir_coeff, 1.0, timesignal.y[i])

                # zeroize first N-1 "corrupted" samples:
                timesignal.y[i][:numtaps-1] = 0.0

                # circular left shift of y:
                timesignal.y[i] = roll(timesignal.y[i], -(numtaps-1))

                # downsize y to user-defined number of samples (SI):
                timesignal.y[i] = timesignal.y[i][::skip]

            # update the sampling_rate attribute of the signal's:
            timesignal.set_sampling_rate(spec_width)

        # ---------------------------------------------------

        # rotate timesignal according to current receiver's phase:
        timesignal.phase(pars['rec_phase'])

        # provide timesignal to the display tab:
        data['Current scan'] = timesignal

        # accumulate...
        if not locals().get('accu'):
            accu = Accumulation()

        # skip dummy scans, if any:
        if pars['run'] < 0: continue

        # add up:
        accu += timesignal

        # provide accumulation to the display tab:
        data['Accumulation'] = accu

        # check how many scans are done:
        if accu.n == pars['NS']: # accumulation is complete

            # make a copy:
            echo = accu + 0

            # compute the initial phase of FID:
            phi0 = arctan2(echo.y[1][0], echo.y[0][0]) * 180 / pi
            if not 'ref' in locals(): ref = phi0
            print 'phi0 = ', phi0

            # rotate FID to maximize y[0][0]:
            #echo.phase(-phi0)
```

Fig. 3. The result script op_spinal_res.py (the spin-alignment experiment)



```python

            # do FFT:
            echo.exp_window(line_broadening=10)
            spectrum = echo.fft(samples=2*pars['SI'])

            # try zero-order phase correction:
            spectrum.phase(-phi0)

            # provide spectrum to the display tab:
            data['Spectrum'] = spectrum

            # check whether it is an arrayed experiment:
            var_key = pars.get('VAR_PAR')
            if var_key:
                # get variable parameter's value:
                var_value = pars.get(var_key)

                # provide signal recorded with this var_value to the display tab:
                data['Accumulation'+"/"+var_key+"=%e"%(var_value)] = accu

                # measure signal intensity vs. var_value:
                if measurement_ranging == True:
                   [start, stop] = accu.get_sampling_rate() * array(measurement_range)
                   measurement[var_value] = sum(accu.y[0][int(start):int(stop)])

                else:
                   measurement[var_value] = sum(accu.y[0][0:31])

                # provide measurement to the display tab:
                data[measurement.get_title()] = measurement

                # update the file name suffix:
                suffix = '_' + str(counter)
                counter += 1

            # save accu if required:
            outfile = pars.get('OUTFILE')
            if outfile:
                datadir = pars.get('DATADIR')

                # write raw data in Simpson format:
                filename = datadir+outfile+suffix+'.dat'
                if path.exists(filename):
                    rename(filename, datadir+'~'+outfile+suffix+'.dat')
                accu.write_to_simpson(filename)

                # write raw data in Tecmag format:
                # filename = datadir+outfile+'.tnt'
                # accu.write_to_tecmag(filename, nrecords=20)

               # write parameters in a text file:
                filename = datadir+outfile+suffix+'.par'
                if path.exists(filename):
                    rename(filename, datadir+'~'+outfile+suffix+'.par')

                fileobject = open(filename, 'w')
                for key in sorted(pars.iterkeys()):
                    if key=='run': continue
                    if key=='rec_phase': continue
                    fileobject.write('%s%s%s'%(key,'=', pars[key]))
                    fileobject.write('\n')
                fileobject.close()

            # reset accumulation:
            del accu


    if var_key == 'D2':
        # KWW fit:
        xdata = measurement.get_xdata()
        ydata = measurement.get_ydata()
        [amplitude, rate, beta] = fitfunc(xdata, ydata)
        print '%s%02g' % ('Amplitude = ', amplitude)
        print '%s%02g' % ('T2 [s] = ', 1./rate)
        print '%s%02g' % ('Beta = ', beta)

        # update display for the fit:
        measurement.y = func([amplitude, rate, beta], xdata)
        data[measurement.get_title()] = measurement

# the fitting procedure:
def fitfunc(xdata, ydata):

    # initialize variable parameters:
    try:
        # solve Az = b:
        A = array((ones(xdata.size/2), xdata[0:xdata.size/2]))
        b = log(abs(ydata[0:xdata.size/2]))
        z = linalg.lstsq(transpose(A), b)
        amplitude = exp(z[0][0])
        rate = -z[0][1]
    except:
        amplitude = abs(ydata[0])
        rate = 1./(xdata[-1] - xdata[0])
    beta = 1
    p0 = [amplitude, rate, beta]

    # run least-squares optimization:
    plsq = leastsq(residuals, p0, args=(xdata, ydata))

    return plsq[0] # best-fit parameters

def residuals(p, xdata, ydata):
    return ydata - func(p, xdata)

# here is the function to fit:
def func(p, xdata):
    return p[0]*exp(-(p[1]*xdata)**p[2])


pass
```

Fig. 3 (cntd). The result script op_spinal_res.py (the spin-alignment experiment)

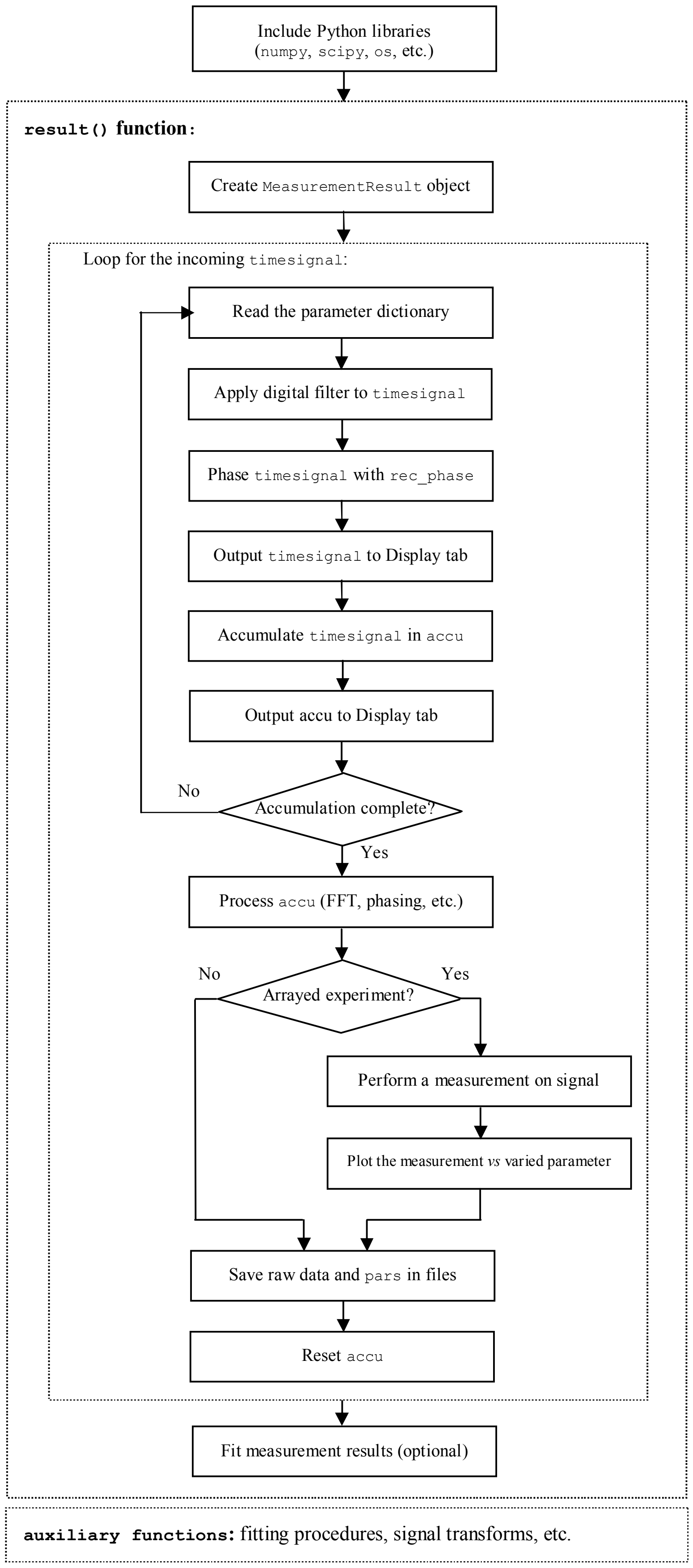

Include Python libraries
(numpy, scipy, os, etc.)
result() function:
Create MeasurementResult object
Loop for the incoming timesignal:
Read the parameter dictionary
Apply digital filter to timesignal
Phase timesignal with rec_phase
Output timesignal to Display tab
Accumulate timesignal in accu
Output accu to Display tab
No
Accumulation complete?
Yes
Process accu (FFT, phasing, etc.)
No
Yes
Arrayed experiment?
Perform a measurement on signal
Plot the measurement vs varied parameter
Save raw data and pars in files
Reset accu
Fit measurement results (optional)
auxiliary functions: fitting procedures, signal transforms, etc.

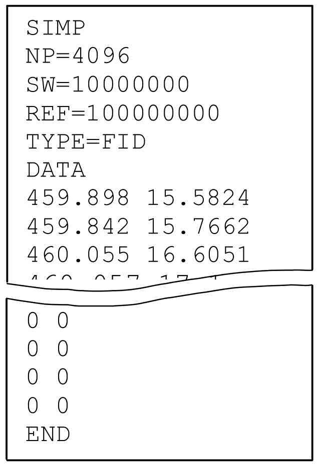
Fig. 4. Flowchart of a result script

3. Data handling

The DAMARIS does not provide an interactive GUI for data processing. Hence, there is a demand for porting data into other processing tools. By default, the DAMARIS stores data in a binary HDF format in the data pool file specified in the Configuration tab, wherefrom the data can be imported to an appropriate program for data analysis. The HDF data are accessible in the Python environment by means of `pyTables` module or with the Java program `HDFView`. A new HDF file is created each time the dictionary `data[]` is updated. Another option built in DAMARIS is saving data in a text format using the Save As Text button in the Display tab.

DAMARIS frond-end classes have few methods for in-line data writing within a result script, which are: `write_to_csv`, `write_to_hdf`, `write_to_simpson`, and `write_to_tecmag`.

The `write_to_simpson` has been chosen to be a default method in the present library, for using it with the NMR processing program NMRnotebook installed on our DAMARIS spectrometers' PC's. The SIMPSON file is a two-column text file – one column for Re and one for Im data – with a little header that reports the number of acquisition points, sampling rate and reference frequency (Fig. 5). The `write_to_simpson` method takes one necessary argument – the path to the file as specified by the parameters `DATADIR` and `OUTFILE`, and two optional keyword arguments – a data delimiter (by default " ") and a reference frequency (by default 100e6). The data file gets the extension `.dat`. Another text file with the extension `.par` is created to store experiment's parameters.



```
SIMP
NP=4096
SW=10000000
REF=100000000
TYPE=FID
DATA
459.898 15.5824
459.842 15.7662
460.055 16.6051

0 0
0 0
0 0
0 0
END
```

Fig. 5. A SIMPSON data file.

The `write_to_tecmag` method is intended primary for 2D experiments. You can find snippets with this method commented out next to `write_to_simpson` (e.g. Fig. 3, lines 146-148). It takes two required keyword arguments – the path to the file and the number of records (2nd data dimension), and few optional arguments – a reference frequency (`frequency`, by default 100e6), last inter-scan delay (`last_delay`, by default 1), receiver phase (`receiver_phase`, by default 0), and the nucleus observed (`nucleus`, by default '1H'). The binary Tecmag files (`.tnt`) are readable by NMRnotebook too and can be imported by many other NMR data processing software let alone the native NTNMR on our Tecmag spectrometer.

4. Composite scripts

A special issue which has not been yet discussed is how to run several experiments sequentially within one script. The way in which DAMARIS jobs are executed allows for concatenation of experiments simply by copying and pasting them in one greater `experiment()` function, one after another. No modifications will be needed in the scripts' code unless you want for such a composite function to report the total acquisition time. Once you press the Start button, you will see in the Log page the time reported for the first experiment only and thus have to wait for the next such report until the first experiment is over. To change this behavior will require a substantial modification of the original scripts and will make individual experiment settings depend on one another, which I dislike. So I decided to bear with this shortcoming for the possibility to combine experiments in a simple copy-and-paste manner. What you still might want to change in the composite `experiment()` is to enclose the individual experiments' code with `if`-statements and use corresponding `True` and `False` controls in the beginning of `experiment()` for switching them on and off.

On the result script's side, I use a variable `the_experiment` to identify the current experiment and to control the way a result is treated. The variable takes the name of the pulse program that is stored in the parameter `pars['PROG']` (assigned locally within program functions). Usually, there is no need to discriminate between experiments until it comes to measuring the signal vs. variable parameter. Then `the_experiment` is checked to determine how to measure the signal intensity and which fitting function to call. For this purpose, separate `MeasurementResult` objects are created, one for each experiment, and put in a common dictionary `measurements`. The experiment names serves as the dictionary's keys so that you can index it with `the_experiment`. Whenever a new experiment is added, you just append an entry with a new experiment's name to `measurements` and specify either or both measurement and fitting procedure for this new experiment in a proper place below. That will be all.

In the folder `AU-Programs`, you can find two examples of such composite scripts: one for diffusiometry, which comprises saturation-recovery with solid-echo detection, hahn echo and stimulated echo experiments, and one for experiments we typically run when measuring on deuterons, which includes saturation-recovery with solid-echo detection, solid echo, spin-alignment, and Zeeman order experiments.

5. The scripts in alphabetical order

5.1. CPMG

Carr-Purcell-Meiboom-Gill (CPMG) echo train acquisition

File names: `op_cpmg_exp.py, op_cpmg_res.py`

Applications: single-shot measurement of T_2 with minimized diffusion effect; measuring the line shapes of very broad lines; signal detection in strong gradients.

Pulse sequence: $90^\circ_x - [\tau - 180^\circ_y - \tau - Acq]_n$

Phase cycle:

step:	*1*	*2*	*3*	*4*
φ_{90}	0	180	90	270
φ_{180}	90	90	180	180
φ_{rec}	0	180	90	270

(minimizes the effect of imperfect 180°-pulses; cancels DC offset; averages channel imbalance)

Acquisition parameters:

Parameter	*Description*	*Typical value*
P90	90° pulse length	few µs
SF	PTS' frequency	Larmor frequency
O1	Offset from SF	up to ±1 MHz
NS	Number of scans	≥ 4
DS	Dummy scans	0
RD	Recycle delay	3-5 T_1
NECH	Number of 180° pulses	many
TAU	Half period of 180° pulses	≥ 40 µs
PHA	Receiver phase	to maximize Re

Comments: 128 samples are acquired from each echo at the 20 MHz rate. The acquisition interval accommodates 6 extra samples, for technical reasons. By default, the echo intensity is measured on Re-channel, as the sum of the samples where the first echo exceeds 10% of its maximum ("noise level"). Alternatively, it can be the sum of all samples or just a middle point of the echo. The echo decay (T_2-decay) is fitted with a mono-exponential function.

5.2. FID

The basic pulse-acquire experiment

File names: `op_fid_exp.py, op_fid_res.py`

Applications: Free-induction signal acquisition; spectroscopy

Pulse sequence: $90^{\circ} - Acq$

Phase cycle:

step:	*1*	*2*	*3*	*4*
φ_{90}	0	180	90	270
φ_{rec}	0	180	90	270

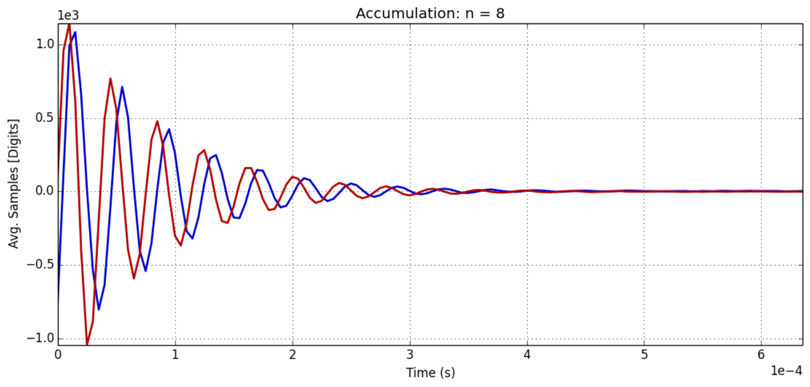
(standard CYCLOPS: cancels DC offset & averages channel imbalance)

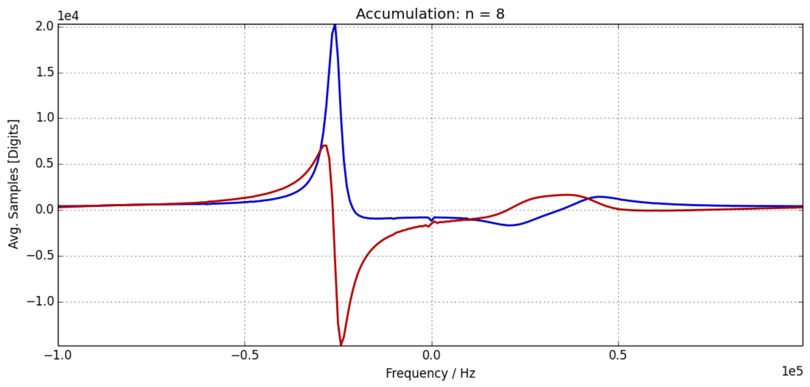
Acquisition parameters:

Parameter	*Description*	*Typical value*
P90	90° pulse length	few μs
SF	PTS' frequency	Larmor frequency
O1	Offset from SF	up to ±1 MHz
SW	Spectral window, or spectrum width	1 kHz - 1 MHz
SI	Number of acquisition points	256-16k
NS	Number of scans	≥ 4
DS	Dummy scans	0-2
RD	Recycle delay	3-5 T_1
DEAD1	Pre-acquisition delay for NMR coil ringing	5-10 μs
PHA	Receiver phase	arbitrary

Comments: The time signal intensity is measured vs variable parameter as a sum of first few samples in the Re-channel. Prior to these measurements, the time signal is phase-corrected by the value calculated from the first FID in the array.

Example: FID and spectrum of ^{19}F in a mono-crystal of LaF_3+5% SrF_2, at room temperature. Parameters of the experiment: P90 = 1.7 us, SW= 200 kHz, SI= 126, RD = 3 s, DEAD1 = 5 us, NS= 8, SF = 338.7 MHz (spectrometer Birgit). Certainly, the 'zero-order phase correction' implemented in `op_fid_res.py` will not do for such a broad spectrum as of LaF_3, so that an interactive phasing with third-party software is required.


Accumulation: n = 8
1e3
1.0
0.5
0.0
−0.5
−1.0
Avg. Samples [Digits]
0
1
2
3
4
5
6
Time (s)
1e−4


Accumulation: n = 8
1e4
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
Avg. Samples [Digits]
−1.0
−0.5
0.0
0.5
Frequency / Hz
1e5

5.3. FID with Background Suppression

The FID experiment with a composite 90°-pulse for background suppression

File names: `op_zgbs_exp.py, op_zgbs_res.py`

Applications: To suppress signals from the probe's parts near the NMR coil and the cables that contain the nuclei being observed.

Pulse sequence: $[90^\circ - 180^\circ - 180^\circ] - Acq$

Phase cycle (from the Bruker pulse program "zgbs"):

step:	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*
φ_{90}	0	0	0	0	90	90	90	90	180	180	180	180	270	270	270	270
φ_{180}	0	90	180	270	0	90	180	270	0	90	180	270	0	90	180	270
φ_{180}	0	0	0	0	180	180	180	180	270	270	270	270	90	90	90	90
φ_{rec}	0	180	0	180	90	270	90	270	0	180	0	180	90	270	90	270

Acquisition parameters: same as in the FID experiment

Comments: The pulse sequence is due to Cory and Ritchey [*JMR*, **80**, 128 (1988)]

5.4. Hahn Echo

Two-pulse Hahn echo acquisition

File names: `op_hahn_exp.py, op_hahn_res.py`

Applications: T_2-relaxometry; diffusiometry; signal filtering

Pulse sequence: $90^\circ_x - \tau - 180^\circ_x - \tau - Acq$

Phase cycle:

step:	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
φ_{90}	0	180	0	180	90	270	90	270
φ_{180}	0	0	180	180	270	270	90	90
φ_{rec}	0	180	0	180	90	270	90	270

(CYCLOPS & cancellation of *z*-component of signal present after 180°-pulse)

Acquisition parameters:

Parameter	*Description*	*Typical value*
P90	90° pulse length	few μs
SF	PTS' frequency	Larmor frequency
O1	Offset from SF	up to ±1 MHz
SW	Spectral window, or spectrum width	1 kHz - 1 MHz
SI	Number of acquisition points	256-16k
NS	Number of scans	≥ 8
DS	Dummy scans	0
RD	Recycle delay	3-5 T_1
DEAD1	Pre-acquisition delay for NMR coil ringing	4-10 μs
PHA	Receiver phase	arbitrary

Comments:

5.5. Miscellaneous

"Go-setup" experiment

File names: `op_gs_exp.py, op_gs_res.py`

Applications: Interactive adjustment of acquisition parameters and shimming

Pulse sequence: $90^{\circ} - Acq$

Phase cycle: None

Acquisition parameters:

Parameter	*Description*	*Typical value*
P90	90° pulse length	few µs
SW	Sampling rate	up to 20 MHz
SI	Number of acquisition points	256-16k
RD	Recycle delay	1 s
DEAD1	Pre-acquisition delay for NMR-coil ringing	5-10 µs

Comments: Nonstop pulse-acquisition without accumulation. Three measurements are reported after each shot: FID amplitude, spectrum integral and spectrum middle frequency (center of gravity).

5.6. Saturation-Recovery

Saturation-recovery experiment

File names: `op_satrec_exp.py, op_satrec_res.py`

Applications: T_1 relaxometry; T_1-weighted FID acquisition

Pulse sequence: $[90^\circ - \mathrm{var}\,\Delta]_n - \tau - 90^\circ - Acq$

Phase cycle:	*step:*	*1*	*2*	*3*	*4*
	φ_{90}	0	0	0	0
(CYCLOPS)	φ_{90}	0	180	90	270
	φ_{rec}	0	180	90	270

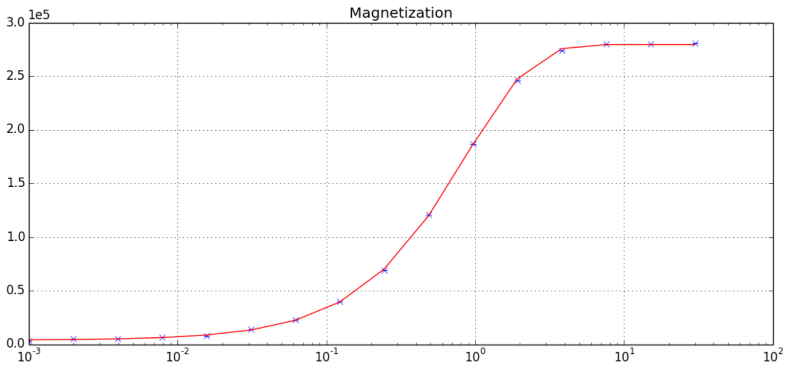
Acquisition parameters:

Parameter	*Description*	*Typical value*
P90	90° pulse length	few μs
SF	PTS' frequency	Larmor frequency
O1	Offset from SF	up to ±1 MHz
SW	Spectral window, or spectrum width	1 - 10 MHz
SI	Number of acquisition points	256-16k
NS	Number of scans	≥ 4
DS	Dummy scans	0
TAU	Delay for recovery	up to $5\times T_1$
DEAD1	Pre-acquisition delay for NMR coil ringing	5-10 μs
PHA	Receiver phase	arbitrary
NECH	Number of saturation pulses	20-40
D1	First interval in saturation pulse train	100 ms
D2	Last interval in saturation pulse train	100 μs

Comments: The saturation sequence comprises up to 40 pulses with logarithmically decreasing intervals from 100 ms down to 100 μs. The FID amplitude is measured against a variable parameter (usually `TAU`) as a sum of few first samples in Re-channel. For the best performance, maximize Re by adjusting `PHA` and set `SW` high enough to make sure that the samples taken are all positive. If it is `TAU` that varies, the recovery curve is fitted with mono-exponential function.

Example: T_1-relaxation of ^{19}F in a mono-crystal of LaF_3+5% SrF_2, at room temperature. Parameters of the experiment: P90 = 1.7 us, SW= 10 kHz, SI= 1024, DEAD1 = 15 us, NS= 8, SF =

338.7 MHz (spectrometer Birgit), TAU is varied from 1 ms to 30 s. The red line is a mono-exponential fit with T_1=0.89 s.



5.7. Saturation-Recovery with Solid-Echo Detection

Saturation-recovery with two-pulse solid echo acquisition

File names: `op_satrec2_exp.py, op_satrec2_res.py`

Applications: T_1 relaxometry on solid samples; acquisition of T_1-weighted echo signals

Pulse sequence: $[90^\circ_x - \mathrm{var}\,\Delta]_n - \tau - 90^\circ_x - t_e - 90^\circ_y - t_e - Acq$

Phase cycle:

step:	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
φ_{90}	0	0	0	0	0	0	0	0
φ_{90}	0	180	0	180	90	270	90	270
φ_{90}	90	90	270	270	0	0	180	180
φ_{rec}	0	180	0	180	90	270	90	270

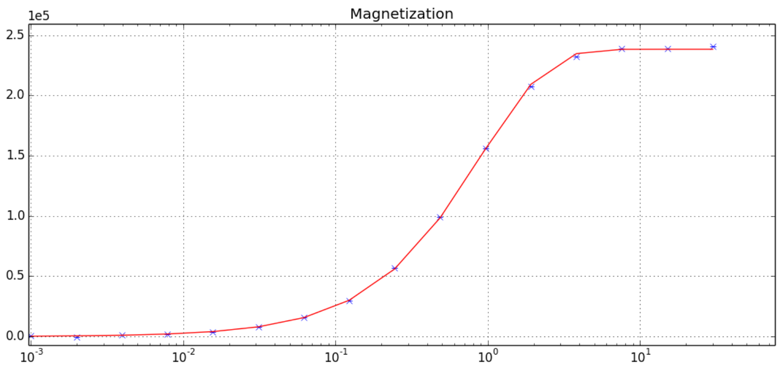
(from Tecmag program: CYCLOPS & cancellation of *z*-component present before the 3rd pulse)

Acquisition parameters:

Parameter	*Description*	*Typical value*
P90	90° pulse length	few μs
SF	PTS' frequency	Larmor frequency
O1	Offset from SF	up to ±1 MHz
SW	Spectral window, or spectrum width	1 - 10 MHz
SI	Number of acquisition points	256-16k
NS	Number of scans	≥ 8
DS	Dummy scans	0
TAU	Delay for recovery	up to 5×T_1
D3	Echo pulse interval	10-20 μs
D4	Echo pre-acquisition delay	0 or ± few μs
PHA	Receiver phase	arbitrary
NECH	Number of saturation pulses	20-40
D1	First interval in saturation pulse train	100 ms
D2	Last interval in saturation pulse train	100 μs

Comments: The saturation sequence comprises up to 40 pulses with logarithmically decreasing intervals from 100 ms down to 100 μs. The solid echo is acquired with the echo delay (parameter `D3)`, usually set as short as the receiver dead time, starting from the top of the echo. It might be necessary to vary the echo pre-acquisition delay `D4` to localize the echo maximum. The echo amplitude is measured the same way as FID in 4.6.

Example: T_1-relaxation of ^{19}F in a mono-crystal of LaF_3+5% SrF_2, at room temperature. Parameters of the experiment: P90 = 1.7 us, SW= 10 kHz, SI= 1024, D3 = 20 us, NS= 8, SF = 338.7 MHz (spectrometer Birgit). TAU is varied from 1 ms to 30 s. The red line is a mono-exponential fit with T_1=0.91 s.



5.8. Solid Echo

Two-pulse solid (quadrupole) echo acquisition

File names: `op_solidecho_exp.py, op_solidecho_res.py`

Applications: acquisition of solid-state spectra free of receiver's dead time artifacts

Pulse sequence: $90^\circ_x - t_e - 90^\circ_y - t_e - Acq$

Phase cycle:

step:	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
φ_{90}	0	180	0	180	90	270	90	270
φ_{90}	90	90	270	270	0	0	180	180
φ_{rec}	0	180	0	180	90	270	90	270

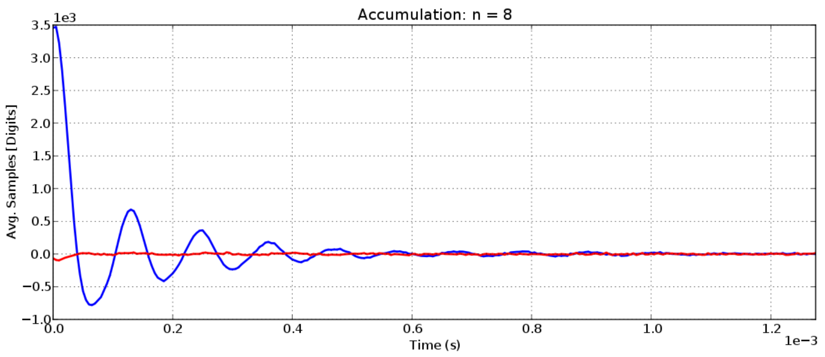
(from Tecmag's program: CYCLOPS & cancellation of *z*-component present after 1st pulse)

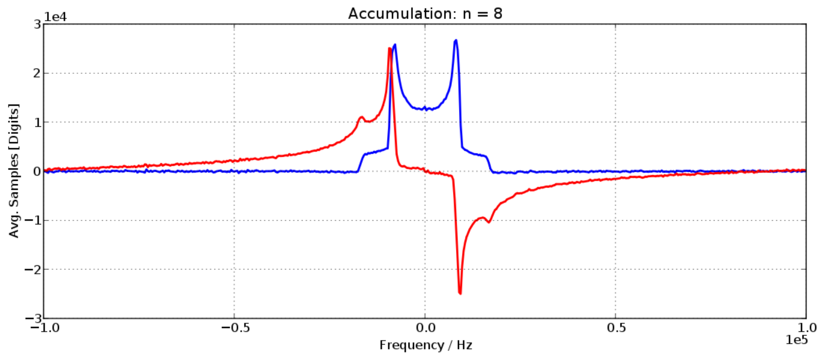
Acquisition parameters:

Parameter	*Description*	*Typical value*
P90	90° pulse length	few μs
SF	PTS' frequency	Larmor frequency
O1	Offset from SF	up to ±1 MHz
SW	Spectral window, or spectrum width	1 kHz - 1 MHz
SI	Number of acquisition points	256-16k
NS	Number of scans	≥ 8
DS	Dummy scans	0
RD	Delay between scans	3-5×T_1
TAU	Echo delay	10-20 μs
D4	Echo pre-acquisition delay	± few μs
PHA	Receiver phase	arbitrary

Comments: The solid echo is acquired with the echo delay (parameter `TAU`) that is usually set as short as the receiver dead time. The acquisition starts from the top of the echo, which is positioned by varying `D4`.

Example: ^{2}H spectrum of hexamethylbenzene-d_{18}, at room temperature (spectrometer Eis). Parameters of the experiment: SF = 46.704 MHz; P90 = 2.5 us; TAU = 20 us; SW = 200 kHz; SI = 256; RD=0.5 s; NS = 8


Accumulation: n = 8
1e3
Avg. Samples [Digits]
Time (s)
1e−3


Accumulation: n = 8
1e4
Avg. Samples [Digits]
Frequency / Hz
1e5

5.9. Spin Alignment

Stimulated spin-echo after storing in a quadrupolar order (Jeener-Broekaert echoes)

File names: `op_spinal_exp.py, op_spinal_res.py`

Applications: studying slow molecular motion in solids (probing sin-sin correlation function)

Pulse sequence: $90^\circ_x - t_p - 45^\circ_y - t_m - 45^\circ_y - t_p - Acq$

Phase cycle:

step:	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
φ_{90}	0	270	0	270	90	90	180	180
φ_{45}	90	180	90	180	180	180	90	90
φ_{45}	90	90	270	270	180	0	0	180
φ_{rec}	0	180	180	0	90	270	90	270

(suppresses single- (SQ) and double-quantum (DQ) coherences present after the second pulse; cancels T_1-recovered magnetization; CYCLOPS)

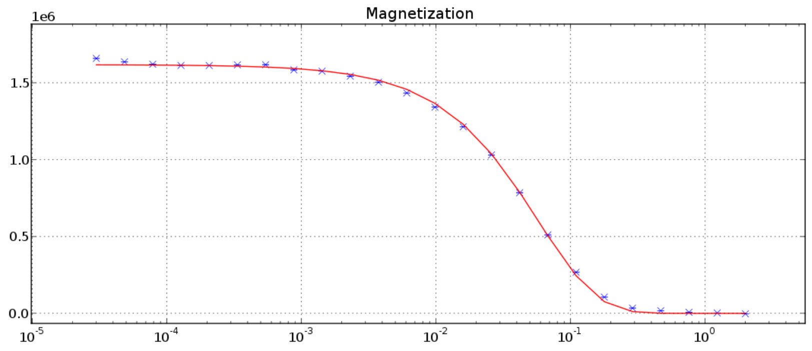
Acquisition parameters:

Parameter	*Description*	*Typical value*
P90	90° pulse length	few μs
SF	PTS' frequency	Larmor frequency
O1	Offset from SF	up to ±1 MHz
SW	Spectral window or spectrum width	1 kHz - 1 MHz
SI	Number of acquisition points	256-16k
NS	Number of scans	≥ 8
DS	Dummy scans	0
RD	Delay between scans	3-5×T_1
D1	Delay after the 1st pulse (t_p)	10-100 μs
D2	Delay after the 2nd pulse (t_m)	from μs to T_1
PHA	Receiver phase	arbitrary

Comments: Not applicable to spins-3/2 and up. Typical use is to measure intensity of the echo arising at `D1` after the last pulse while varying `D2`. The intensity is measured from few first samples in Re-channel. To maximize Re, `PHA` is adjusted with a reference phase `phi0` obtained from the experiment with the shortest `D2`. The first pulse interval `D1` (or t_p) is usually kept as short as the receiver dead time, though a maximum echo is achieved at `D1` ~ $(2\delta)^{-1}$, where δ is a quadrupolar/dipolar spectrum width. It is not so important to stay precisely at the top of the echo as the experiment is not intended for line shape analysis (nevertheless the processing part does include

FFT). In the end of the measurement "echo vs `D2`", the result script performs a KWW (stretched exponential) fit to the data.

Example: ^{2}H STE decay in hexamethylbenzene-d_{18}, at room temperature (spectrometer Eis). Parameters of the experiment: SF = 46.704 MHz; P90 = 2.3 us; D1 = 30 us; RD=0.5 s; NS = 32. The red line is a mono-exponential fit.



5.10. Spin Alignment: Spin-3/2

Stimulated spin-echo after storing in a quadrupolar order (Jeener-Broekaert echoes). The only difference from 4.9 is in the phase cycle.

File names: `op_spinal32_exp.py, op_spinal32_res.py`

Applications: studying slow molecular motion in solids (probes sin-sin correlation function)

Pulse sequence: $90^{\circ}_{x} - t_p - 45^{\circ}_{y} - t_m - 45^{\circ}_{y} - t_p - Acq$

Phase cycle: [based on Qi et al., JMR 169 (2004) 225-239]

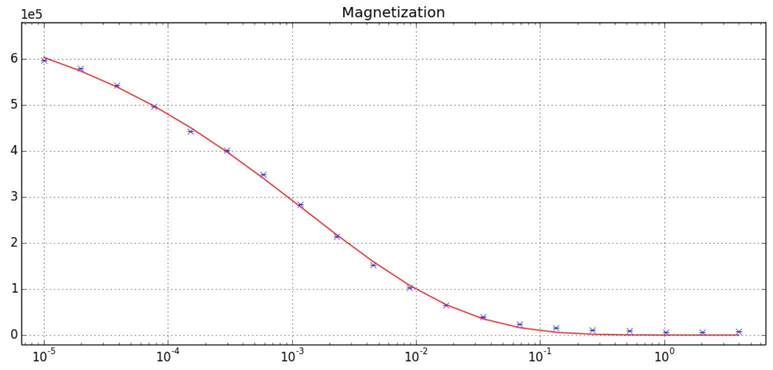
step:	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*
φ_{90}	0	180	0	180	90	270	90	270	90	270	90	270	180	0	180	0
φ_{45}	90	90	270	270	0	0	180	180	180	180	0	0	90	90	270	270
φ_{45}	0	0	0	0	180	180	180	180	90	90	90	90	270	270	270	270
φ_{rec}	180	0	0	180	180	0	0	180	270	90	90	270	270	90	90	270

(suppresses SQ, DQ and triple-quantum (TQ) coherences present after the 2nd pulse; cancels T_1-recovered magnetization; CYCLOPS)

Acquisition parameters: same as for 4.9

Comments: Suitable for spins-3/2. The phase cycle is made twice as long compared to spin-alignment sequence 4.9 for spins-1 to cancel the triple-quantum coherence arising after the second pulse.

Example: ^{7}Li STE decay in a $Li_2Ti_3O_7$ poly-crystallite sample, at room temperature (spectrometer Birgit). Parameters of the experiment: SF = 139.9 MHz; P90 = 1.8 us; D1 = 60 us; RD = 25 s; NS = 32. The red line is a stretched-exponential (KWW) fit.



5.11. Steady Gradient Spin Echo

Stimulated echo-based measurements of diffusion in a static magnetic field gradient

File names: `op_sgse_exp.py, op_sgse_res.py`

Applications: diffusiometry

Pulse sequence: $90^\circ_x - t_p - 90^\circ_x - t_m - 90^\circ_x - t_p - Acq$

Phase cycle: [due to Hürlimann and Venkataramanan, JMR 157, 31 (2002)]

step:	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*
φ_{90}	0	180	0	180	0	180	0	180	90	270	90	270	90	270	90	270
φ_{90}	0	0	180	180	0	0	180	180	0	0	180	180	0	0	180	180
φ_{90}	0	0	0	0	180	180	180	180	0	0	0	0	180	180	180	180
φ_{rec}	0	180	180	0	180	0	0	180	270	90	90	270	90	270	270	90

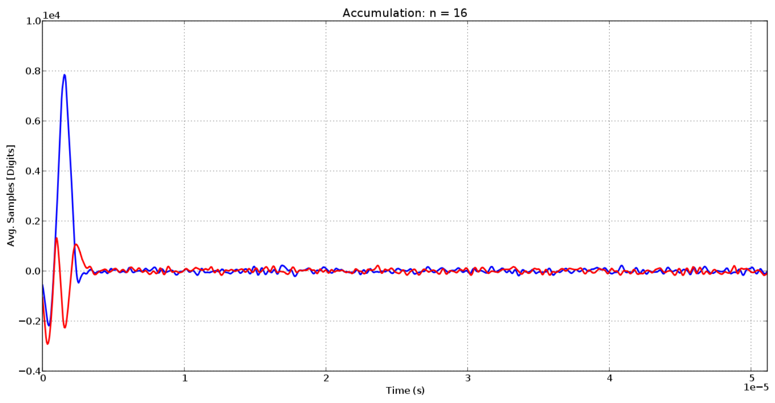
(suppresses SQ coherences present after the second pulse; cancels T_1-recovered magnetization; CYCLOPS)

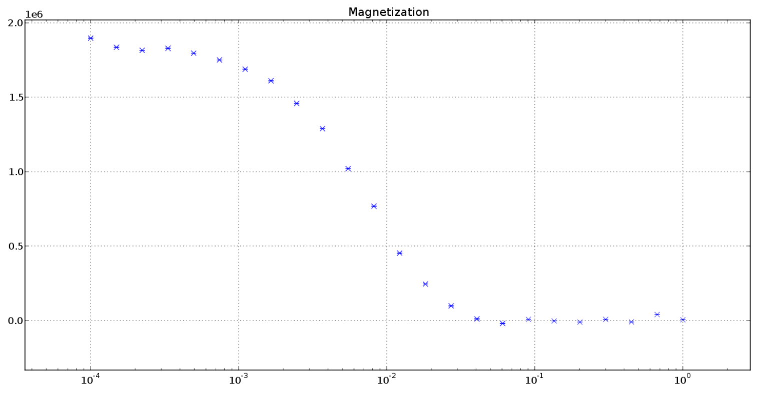
Acquisition parameters:

Parameter	*Description*	*Typical value*
P90	90° pulse length	few μs
SF	PTS' frequency	Larmor frequency
O1	Offset from SF	up to ±1 MHz
SW	Spectral window or spectrum width	1 kHz - 1 MHz
SI	Number of acquisition points	256-16k
NS	Number of scans	≥ 16
DS	Dummy scans	0
RD	Delay between scans	3-5×T_1
D1	Delay after the 1st pulse (t_p)	10-30 μs
D2	Delay after the 2nd pulse (t_m)	from few μs to T_1
PHA	Receiver phase	arbitrary

Comments: The scripts for this experiment are identical to Stimulated Echo (see 4.13). The only difference is in measurement of the echo intensity. Namely, due to an intrinsically poor SNR and short FID, all echo samples above the noise level are taken and added up. When `D2` is varied (a usual scenario), the echo decay is fitted with mono-exponential function.

Example: (Upper) ^{1}H STE in tap water, in a static field gradient of … T/m (spectrometer Magnex). Parameters of the experiment: P90 = 0.7 us, SF = 91.183 MHz, SW = 20 MHz, SI = 1024, NS = 16, RD = 4 s, D1 = 20 us, D2 = 30 us, room temperature. (Lower) STE intensity as a function of D2.





5.12. Steady Gradient Spin Echo with CPMG Detection

Stimulated-echo sequence followed by a CPMG readout of the signal

File names: `op_sgse2_exp.py, op_sgse2_res.py`

Applications: diffusiometry

Pulse sequence: $90^\circ_x - t_p - 90^\circ_x - t_m - 90^\circ_x - t_p - [t_e - 180^\circ_y - t_e - Acq]_n$

Phase cycle: [due to Hürlimann and Venkataramanan, JMR 157, 31 (2002)]

step:	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*
φ_{90}	0	180	0	180	0	180	0	180	90	270	90	270	90	270	90	270
φ_{90}	0	0	180	180	0	0	180	180	0	0	180	180	0	0	180	180
φ_{90}	0	0	0	0	180	180	180	180	0	0	0	0	180	180	180	180
φ_{180}	90	90	90	90	90	90	90	90	0	0	0	0	0	0	0	0
φ_{rec}	180	0	0	180	0	180	180	0	90	270	270	90	270	90	90	270

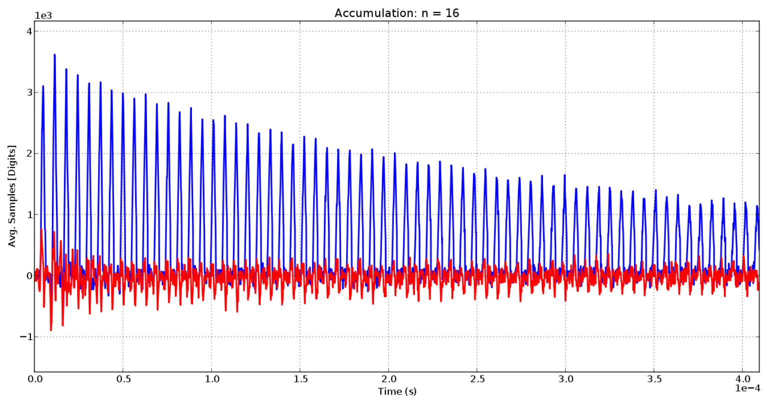
(suppresses SQ coherences present after the 2nd pulse; cancels T_1-recovered magnetization; CYCLOPS)

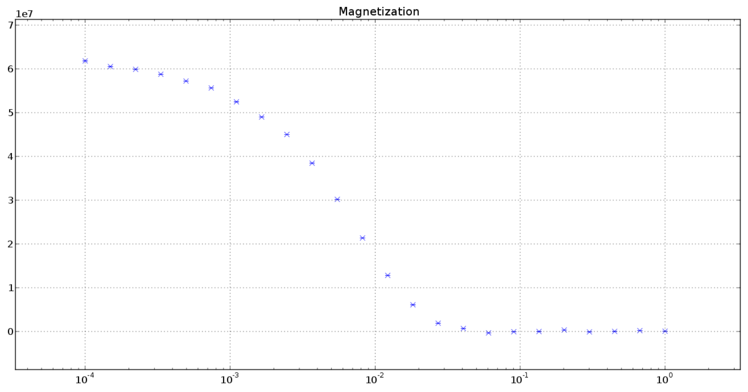
Acquisition parameters:

Parameter	*Description*	*Typical value*
P90	90° pulse length	few μs
SF	PTS' frequency	Larmor frequency
O1	Offset from SF	up to ±1 MHz
NS	Number of scans	≥ 16
DS	Dummy scans	0
RD	Delay between scans	3-5×T_1
D1	Delay after the 1st pulse (t_p)	10-20 μs
D2	Delay after the 2nd pulse (t_m)	from few μs to T_1
NECH	Number of 180°-pulses in the CPMG train	up to few hundreds
TAU	Half pulse period in the CPMG train	> 40 μs
PHA	Receiver phase	arbitrary

Comments: Instead of a single-echo acquisition, STE is re-focused many times by 180°-pulses (a CPMG train) and the resulting echoes are added up to improve SNR. When `D2` is varied (an usual scenario), the cumulative echo intensity decay is fitted with mono-exponential function. The CPMG echoes is liable to change in phase under `D2` variation. It may attenuate the measured Re-signal intensity, thereby causing a faster decay at short `D2`'s.

Example: (Upper) CPMG readout of ^{1}H STE in tap water, in a static field gradient of … T/m (spectrometer Magnex). Parameters of the experiment: P90 = 0.7 us, SF = 91.183 MHz, SW = 20 MHz, SI = 1024, NS = 16, RD = 4 s, D1 = 20 us, D2 = 30 us, NECH = 128, TAU = 40 us, room temperature. (Lower) Total sum of CPMG echoes as a function of D2.





5.13. Stimulated Echo

Three-pulse stimulated echo sequence

File names: `op_ste_exp.py, op_ste_res.py`

Applications: diffusiometry; signal filtering

Pulse sequence: $90^\circ_x - t_p - 90^\circ_x - t_m - 90^\circ_x - t_p - Acq$

Phase cycle: [due to Hürlimann and Venkataramanan, JMR 157, 31 (2002)]

step:	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*
φ_{90}	0	180	0	180	0	180	0	180	90	270	90	270	90	270	90	270
φ_{90}	0	0	180	180	0	0	180	180	0	0	180	180	0	0	180	180
φ_{90}	0	0	0	0	180	180	180	180	0	0	0	0	180	180	180	180
φ_{rec}	0	180	180	0	180	0	0	180	270	90	90	270	90	270	270	90

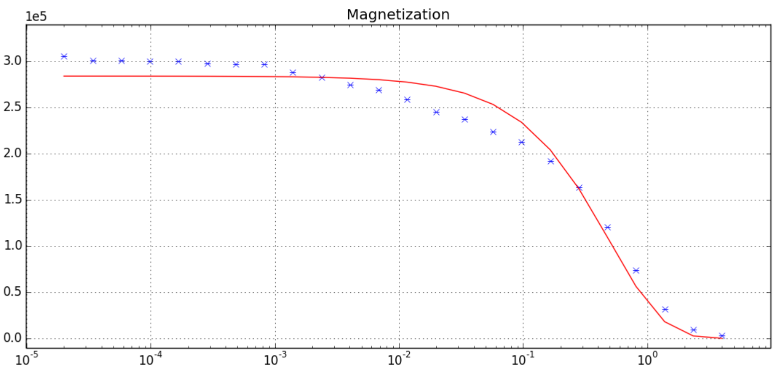
(suppresses SQ coherences present after the 2nd pulse; cancels T_1-recovered magnetization after 3rd pulse; CYCLOPS)

Acquisition parameters:

Parameter	*Description*	*Typical value*
P90	90° pulse length	few μs
SF	PTS' frequency	Larmor frequency
O1	Offset from SF	up to ±1 MHz
SW	Spectral window (or spectrum width)	1 kHz - 1 MHz
SI	Number of acquisition points	256-16k
NS	Number of scans	≥ 16
DS	Dummy scans	0
RD	Delay between scans	3-5×T_1
D1	Delay after the 1st pulse (t_p)	10-20 μs
D2	Delay after the 2nd pulse (t_m)	from few μs to T_1
D4	Echo pre-acquisition delay	± few μs
PHA	Receiver phase	arbitrary

Comments: The sequence is used for diffusion measurements in a static field gradient (see 4.11 and 4.12) and in other experiments that include a z-storage interval (mixing time `D2`). The phase cycle does not suppress DQ coherences arising after the 2nd pulse in the case of non-averaged dipole or quadrupole interactions. It may give rise to the echo intensity at short mixing times.

Example: ^{1}H STE decay in an eraser, at room temperature. Parameters of the experiment: P90 = 3.5 us, SF = 360 MHz (spectrometer Birgit), SW = 10 MHz, SI = 1024, D1 = 20 us, RD = 4 s, D2 is varied from 20 us to 4 s. The red line is a mono-exponential fit.



5.14. T1Q

Jeener-Broekaert sequence with solid-echo detection, to measure quadrupolar order relaxation

File names: `op_t1q_exp.py, op_t1q_res.py`

Applications: T_{1Q} measurements

Pulse sequence: $90^\circ_x - t_p - 45^\circ_y - t_m - 45^\circ_x - \Delta - 90^\circ_x - \Delta - Acq$

Phase cycle:

step:	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
φ_{90}	0	0	90	90	180	180	270	270
φ_{45}	90	90	180	180	270	270	0	0
φ_{45}	0	180	0	180	180	0	180	0
φ_{90}	0	0	180	180	270	90	90	270
φ_{rec}	0	180	0	180	180	0	180	0

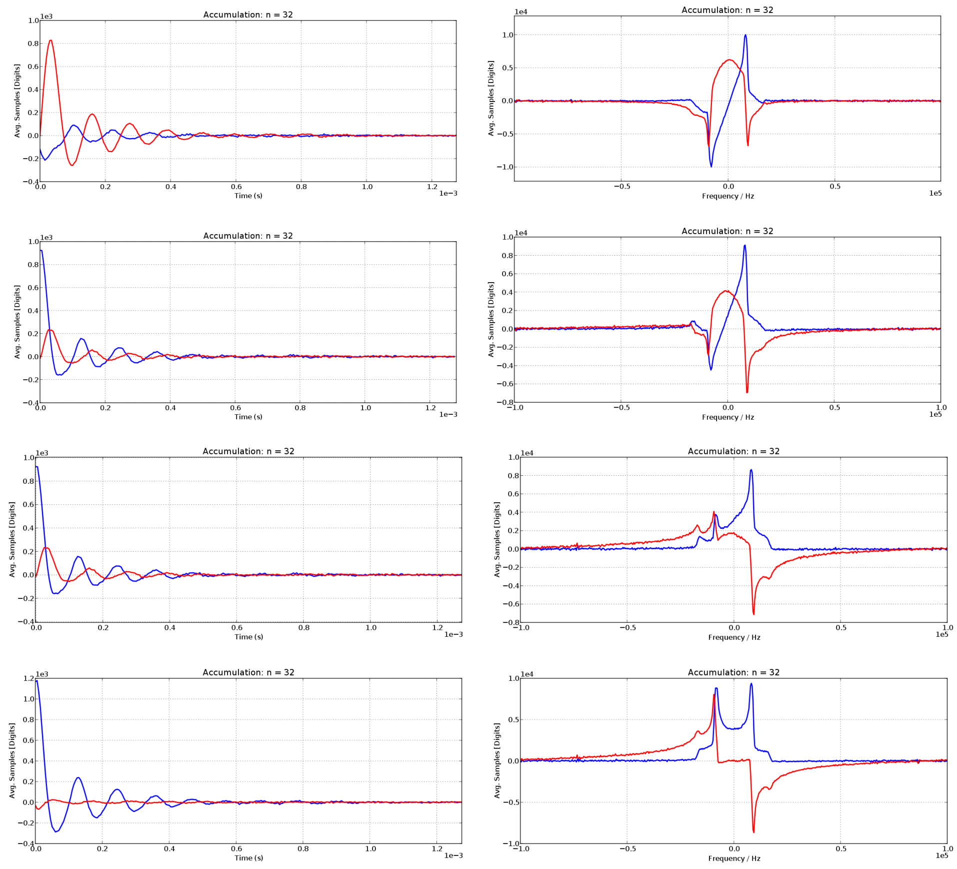
(suppresses SQ & DQ coherences present after the 2nd pulse; destroys STE after 3rd pulse; cancels T_1-recovered magnetization after 3rd pulse; CYCLOPS)

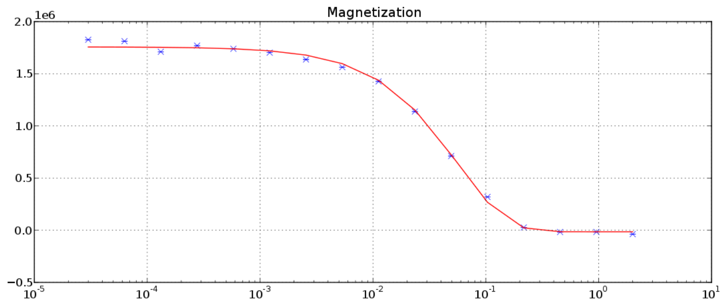
Acquisition parameters:

Parameter	*Description*	*Typical value*
P90	90° pulse length	few μs
SF	PTS' frequency	Larmor frequency
O1	Offset from SF	up to ±1 MHz
SW	Spectral window (or spectrum width)	1 kHz - 1 MHz
SI	Number of acquisition points	256-16k
NS	Number of scans	≥ 8
DS	Dummy scans	0
RD	Delay between scans	3-5×T_1
D1	Delay after the 1st pulse (t_p)	10-20 μs
D2	Delay after the 2nd pulse (t_m)	from few μs to T_1
PHA	Receiver phase	arbitrary

Comments: The experiment is based on JMR 43, 213 (1981). The 4th refocusing 90°-pulse is added to enable measuring fast-decaying signals. It also destroys a stimulated echo signal, which is required when `D1` is set shorter than the free-induction decay time. The quadrupolar order relaxation is monitored via vanishing of the Im-component of the signal. The amplitude of the Im-component is measured as a sum of few first points of its discrete cosine transform. The T_{1Q} value is used in data fitting in spin-alignment experiments (4.8, 4.9) as a fixed value of the respective exponential factor of the echo decay. It is a valuable metric of molecular dynamics of its own: the reciprocal T_{1Q} is proportional to the spectral density $J_1(\omega_0)$ of quadrupole fluctuations.

Example: (Upper) ^{2}H time signals and spectra in hexamethylbenzene-d$_{18}$, at four different mixing times D2 (15 us, 15 ms, 70 ms, and 1s), at room temperature (spectrometer Eis). Quadrupolar order relaxation is tracked via decreasing the Im-channel time signal (red one). Parameters of the experiment: P90 = 2.3 us; SF = 46.704 MHz; SW = 200 kHz, SI = 256; D1 = 30 us, RD=0.5 s; NS = 32. (Lower) The discrete cosine transform amplitude of the Im-signal against D2. The red line is a mono-exponential fit with T_{1Q} = 56 ms.





5.15. Zeeman Order

Stimulated spin-echo after storing in Zeeman order

File names: `op_zeeman_exp.py, op_zeeman_res.py`

Applications: studying slow molecular motion in solids (cos-cos correlation)

Pulse sequence: $90^\circ_x - t_p - 90^\circ_x - t_m - 90^\circ_x - t_p - Acq$

Phase cycle:

step:	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
φ_{90}	0	270	0	270	180	90	180	90
φ_{90}	0	90	0	90	0	90	0	90
φ_{90}	0	0	180	180	270	270	90	90
φ_{rec}	0	180	180	0	90	270	270	90

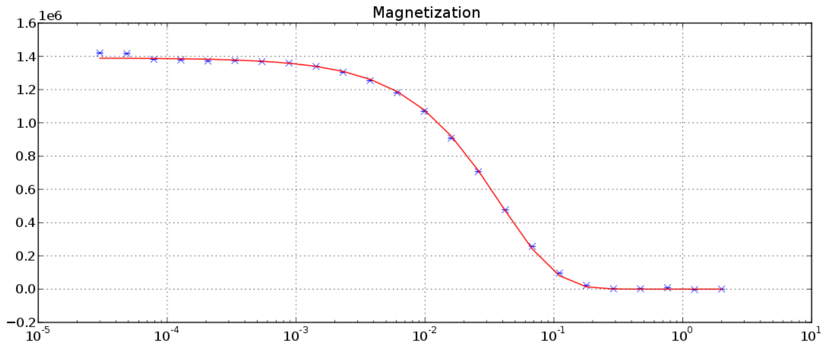
(suppresses single- (SQ) and double-quantum (DQ) coherences present after the second pulse; cancels T_1-recovered magnetization; CYCLOPS)

Acquisition parameters:

Parameter	*Description*	*Typical value*
P90	90° pulse length	few µs
SF	PTS' frequency	Larmor frequency
O1	Offset from SF	up to ±1 MHz
SW	Spectral window or spectrum width	1 kHz - 1 MHz
SI	Number of acquisition points	256-16k
NS	Number of scans	≥ 8
DS	Dummy scans	0
RD	Delay between scans	3-5×T_1
D1	Delay after the 1st pulse (t_p)	10-20 µs
D2	Delay after the 2nd pulse (t_m)	from µs to T_1
PHA	Receiver phase	arbitrary

Comments: Not applicable to spins-3/2 and up. Typical use is to vary D2 and measure echo intensity at `D1` after the last pulse, as a sum of some first samples in Re-channel. To maximize Re, `PHA` is adjusted with a reference phase `phi0` obtained from the first of arrayed experiments (that with the shortest D2). The first pulse interval D1 (or t_p) is usually kept as short as the receiver dead time. It is not so important to stay precisely at the top of the echo as the experiment is not intended for lineshape analysis (nevertheless the processing part does include FFT). In the end of the measurement "echo vs D2", the result script fits a KWW (stretched exponential) function to the data. Unlike the Spin-Alignment sequence 4.9, the Zeeman Order sequence refocuses both motionally-narrowed and rigid-like components present in the spectrum. If you want a STE free of motionally narrowed components, use 4.9 instead.

Example: ^{2}H STE decay in hexamethylbenzene-d_{18}, at room temperature (spectrometer Eis). Parameters of the experiment: SF = 46.704 MHz; P90 = 2.3 us; D1 = 30 us; RD=0.5 s; NS = 32, D2 is varied from 30 us to 2 s. The red line is a mono-exponential fit with decay time constant 39 ms.



5.16. ZZ-Exchange with T_2-Selection

Preliminarily filtered short-T_2 component is built up during *z*-storage upon chemical exchange.

File names: `op_t2zz_exp.py, op_t2zz_res.py`

Applications: to measure kinetics under slow exchange

Pulse sequence: $90^\circ_x - t_e - \beta_y - t_e - 90^\circ_x - t_m - 90^\circ_x - Acq$

Phase cycle:

step:	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*
φ_{90}	0	180	90	270	180	0	270	90	180	0	270	90	0	180	90	270
φ_{β}	90	90	180	180	270	270	0	0	270	270	0	0	90	90	180	180
φ_{90}	0	0	90	90	180	180	270	270	180	180	270	270	0	0	90	90
φ_{90}	0	0	90	90	180	180	270	270	0	0	90	90	180	180	270	270
φ_{rec}	0	180	90	270	180	0	270	90	0	180	90	270	180	0	270	90

(eliminates T_1-recovered signal and contaminating echo signals; CYCLOPS)

Acquisition parameters:

Parameter	*Description*	*Typical value*
P90	90° pulse length	few μs
SF	PTS' frequency	Larmor frequency
O1	Offset from SF	up to ±1 MHz
SW	Spectral window or spectrum width	1 kHz - 1 MHz
SI	Number of acquisition points	256-16k
NS	Number of scans	≥ 16
DS	Dummy scans	0
RD	Delay between scans	3-5×T_1
D1	Echo delay in the T_2 filter (t_e)	3-5×T_2
D2	Z-storage time (t_m)	from tens of μs to T_1
DEAD1	Pre-acquisition delay for NMR coil ringing	5-10 μs
PHA	Receiver phase	arbitrary

Comments: The first two pulses constitute a T_2 filter. The delay D1 is set such as to suppress the spectral peak with a shorter T_2 (usually appears broader in the spectrum). The second pulse of the filter is, by default, a 90° pulse, which is optimum for solid-like (Gaussian) peaks. If, however, the longer-T_2 peak that is supposed to pass through the T_2-filter has a Lorentzian line shape, use a 180° pulse instead. This will minimize the zz-exchange between longer- and shorter-T_2 components during D1. Because of T_1-relaxation, the total intensity of the signal decreases with increasing D2. The measurable exchange rates are thus limited between $1/T_{2\text{short}}$ and $1/T_1$.

6. Concluding remarks

This collection of scripts is but to give one a general idea of scripting NMR experiments with DAMARIS. It seems quite suitable in the present form for those to whom NMR is a new technique, while experienced users who tend to write their own scripts can use it as template / building blocks / complement to their own efforts. A special attention was paid to make the scripts look similar in as many aspects as possible – from manipulating experiment's parameters to data handling, so that the user can switch from one experiment to the other swiftly. Such an approach has proved itself useful for both extending scripts with new functionality and writing composite scripts for several experiments. Given time, new scripts will be added. If you have a demand for a special experiment or find bugs in writing, let me know via `oleg@nmr.physik.tu-darmstadt.de`.