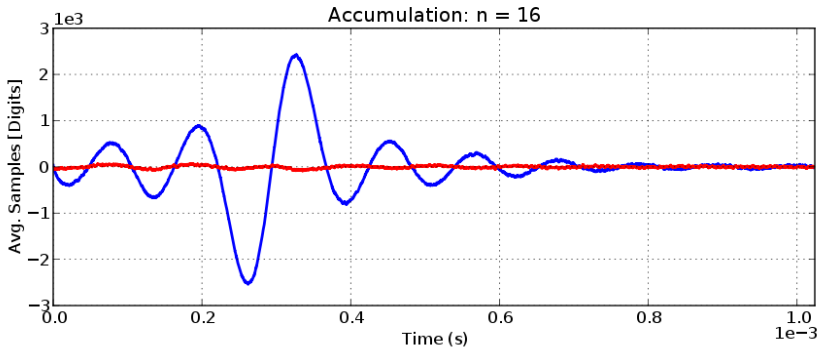
A User's Guide

The DAMARIS Script Library

Version 1.1, September 2015



Oleg V. Petrov • AG Vogel • Technische Universität Darmstadt

The DAMARIS Script Library: A User's Guide

v1.1 September 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 3

2. The script layout
   2.1. Experiment scripts 6
   2.2. Result scripts 11

3. Data output 16

4. Combining experiments 17

5. The scripts in alphabetical order
   5.1. CPMG 18
   5.2. EXSY 19
   5.3. EXSY 2H 21
   5.4. FID 23
   5.5. FID with Background Suppression 25
   5.6. Hahn Echo 26
   5.7. Miscellaneous 27
   5.8. Saturation Recovery 28
   5.9. Saturation Recovery with Solid Echo Detection 30
   5.10. Solid Echo 32
   5.11. Spin Alignment 34
   5.12. Spin Alignment Four Pulses 36
   5.13. Spin Alignment Spin-3/2 38
   5.14. Steady Gradient Spin Echo 39
   5.15. Steady Gradient Spin Echo with CPMG Detection 41
   5.16. Steady Gradient Spin Echo with XY-16 Detection 43
   5.17. Stimulated Echo 45
   5.18. T_{1Q} 47
   5.19. T_2-filtered ZZ-Exchange 49
   5.20. Zeeman Order 51
   5.21. Zeeman Order Four Pulses 53

6. Concluding remarks 55

1. Introduction

This document describes pulse programs and acquisition parameters setup for NMR experiments compiled in the DAMARIS Script Library. The experiments were chosen to comply primary 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. The library currently includes 22 scripts (by September 2015), beginning with the basic pulse-acquire sequence and ending with 2D experiments to measure kinetics under slow exchange (Table 1).

The concept of DAMARIS and how to program within this NMR software can be found in a manual by Markus Rosenstihl 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 two functions: one has a pre-defined name `experiment()` and serves as a driver of a pulse program; the other, with an arbitrarily name, e.g. `fid_experiment()`, defines the pulse program proper and is called from within `experiment()`. There may be more functions in `*_exp.py`, but these two are usually all what you need to run a particular experiment. The other script is called a result script (the module `*_res.py`). It provides processing routines for an incoming signal and outputs the signal and whatever measurements having been performed on it to the screen or a file. It contains a pre-defined function `result()` and may also hold auxiliary functions for signal transforms and data fitting. The experiment and result scripts are written in the Python programming language (hence the extension `.py`), using DAMARIS front-end classes `Experiment`, `ADCResult`, `Accumulation`, `MeasurementResult`, and others.

The experiment and result modules reside under separate DAMARIS' tabs. Communication between the modules is implemented via a parameter description routine. Namely, a dictionary consisting parameter key-and-value pairs is created within the experiment script by the command `set_description` and is retrieved in the result script by the `get_description_dictionary` command. Also, there are few '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 seems to have 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 used to working with Bruker's XWINNMR/TopSpin or Varian's 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

on those elements of the present DAMARIS scripts that resemble the way the commercial NMR systems are organized. For example, although DAMARIS does not have such a thing as a parameter table, a common parameter namespace is used throughout the library, all the parameters being stored in a special dictionary `pars` which is configured 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 supposed to be modified by the user to run the 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
EXSY	`op_noesy_exp.py` `op_noesy_res.py`	2D Exchange Spectroscopy experiment (for spins-1/2)
EXSY_2H	`op_exsy2h_exp.py` `op_exsy2h_res.py`	2D Exchange Spectroscopy experiment (for spins-1)
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_Four_Pulses	`op_spinal4pulses_exp.py` `op_spinal4pulses_res.py`	As above but with refocusing 90° pulse (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)
Steady_Gradient_Spin_Echo_with_XY-16_Detection	`op_sgse16_exp.py` `op_sgse16_res.py`	SGSE with XY-16 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})
T2-Filtered_ZZ-Exchange	`op_t2zz_exp.py` `op_t2zz_res.py`	T_2-filtered magnetization builds up via *ZZ*-exchange
Zeeman_Order	`op_zeeman_exp.py` `op_zeeman_res.py`	Spin-echo after Zeeman order during t_m (for spins-1)
Zeeman_Order_Four_Pulses	`op_zeeman4pulses_exp.py` `op_zeeman4pulses_res.py`	As above but with refocusing 90° pulse (for spins-1)

2. The script layout

2.1. Experiment scripts

In the very beginning of an experiment script, hardware-related parameters are specified such as TTL-lines which control a RF transmitter, the transmitter's enabling delay, and ADC sensitivity (Fig. 1, lines 3-6). As these parameters relate to the NMR hardware configuration rather than to an NMR experiment itself, 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 initialized in first few lines (Fig. 1, lines 12-24). In the example script on Fig. 1, 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 (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. The outer loop runs for a variable parameter. The latter takes on values which are calculated and arrayed according to the variable parameter settings (Fig. 1, lines 51-59). If no variable parameter is named (a one-time experiment with all-fixed parameters), only the loop for signal accumulation runs. 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, the `DS` value is subtracted from `run`, and the signal is not accumulated on the result script's side until the `run` becomes non-negative, so skipping dummy scans. Here are also specified phase lists for r.f. pulses and a receiver, which will complete the parameter setup (lines 106-109). Phase lists are the property of a pulse program and, as such, are not intended to be modified in the course of experiment. For this reason, they are separated from the rest of `pars` and 'hidden' in the pulse program. Note that the name `PH2` is reserved for the receiver phase.

The `pars` are read in local variables before passing to the pulse sequence commands (lines 112-124), for two reasons. First, aesthetic, to avoid bulky expressions with constructions like `pars['XYZ']` in the commands. Second, 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). For 2D experiments (EXSY and 2H EXSY), this is where sampling parameters for an indirect dimension F1 are set.

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

```
# -*- 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 modified 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)

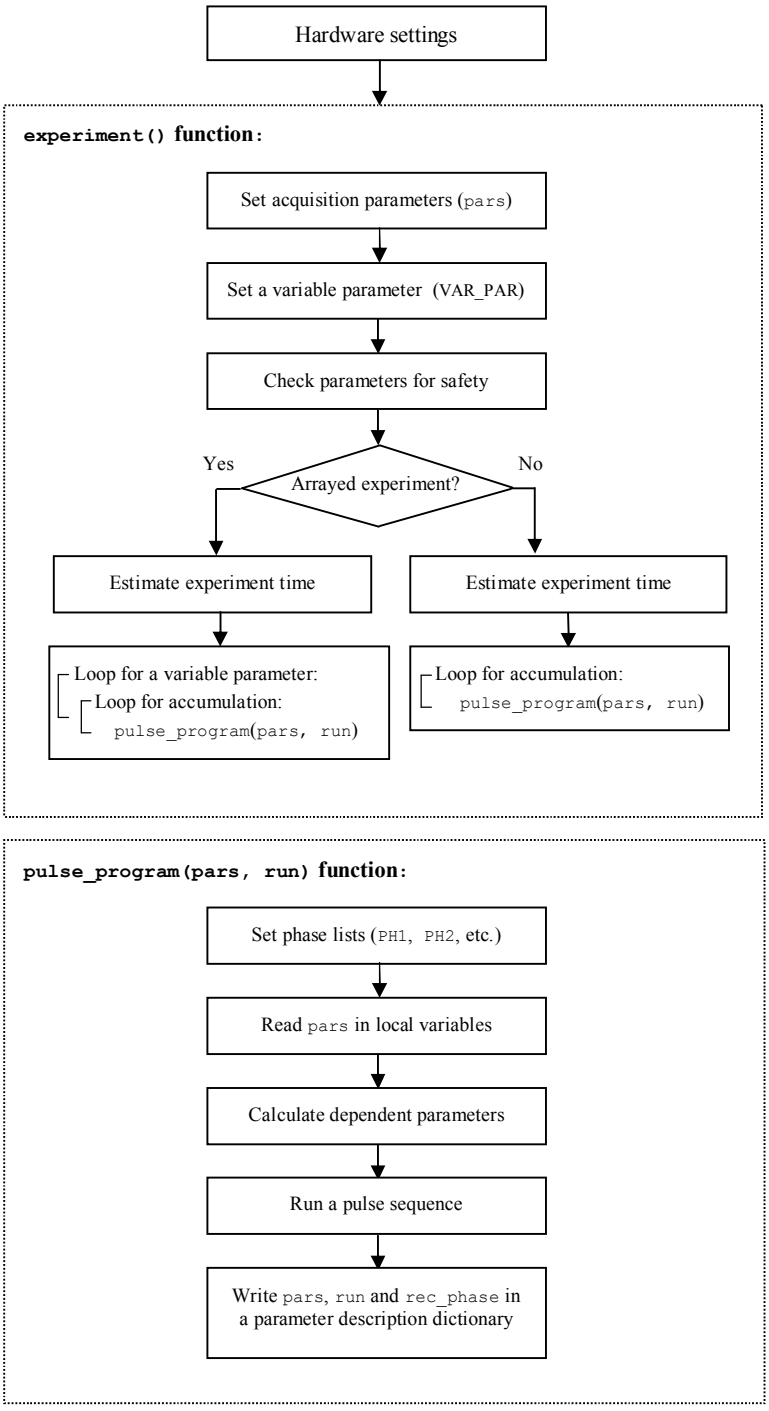


Fig. 2. Flowchart of an experiment scripts

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 current scan's settings. The settings are read in by the `get_description_dictionary()` method amid other `timesignal`'s attributes (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 those including CPMG and XY-16 pulse trains. 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, the digital filtering is implemented on a hardware level by means of a special signal processor which performs the FIR filtering during the signal acquisition. In DAMARIS, it is a post-acquisition procedure. The 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 are no longer consistent with the rest of the signal and, as such, must be discarded. To keep the required number of samples (parameter `SI`), those first N-1 samples are made into tailing zeroes (they will 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. One should account for this time interval (called GroupDelay in RINMR and DEAD2 in XWINNMR/TopSpin) when positioning the acquisition onset.

The phasing, or digital rotation, of `timesignal` is controlled by a special parameter `rec_phase`, which is a current value from the `PH2` phase list taken with an opposite sign (see Fig. 1, line 160). The rotation is performed by 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`.

Having been 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 under the name `Accumulation`.

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). For that purpose, an initial phase `phi0` of `accu` is

calculated (line 94). The `phi0` value is also output on the Log page and can be appended to `PHA` to maximize a Re-signal. Prior to FFT, `accu` is multiplied by an exponential window function (apodisation, line 102) and, following a standard practice, filled with zeroes to double the dimension. If it is an arrayed experiment, the next bit of the script will perform the 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, following the path specified by `DATADIR` and `OUTFILE`. More about data filing is 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 the `result()`'s main loop to perform post-measurement tasks such as data fitting (lines 167-178). The auxiliary functions for fitting and other would-be tasks are placed in the end of the script (lines 181, 202, 206). All necessary Python libraries such as `numpy`, `scipy.signal`, `scipy.optimize`, `os` are inserted at the top of the script (Fig. 2, lines 3-6).

```
# -*- 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)

```
            # 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)

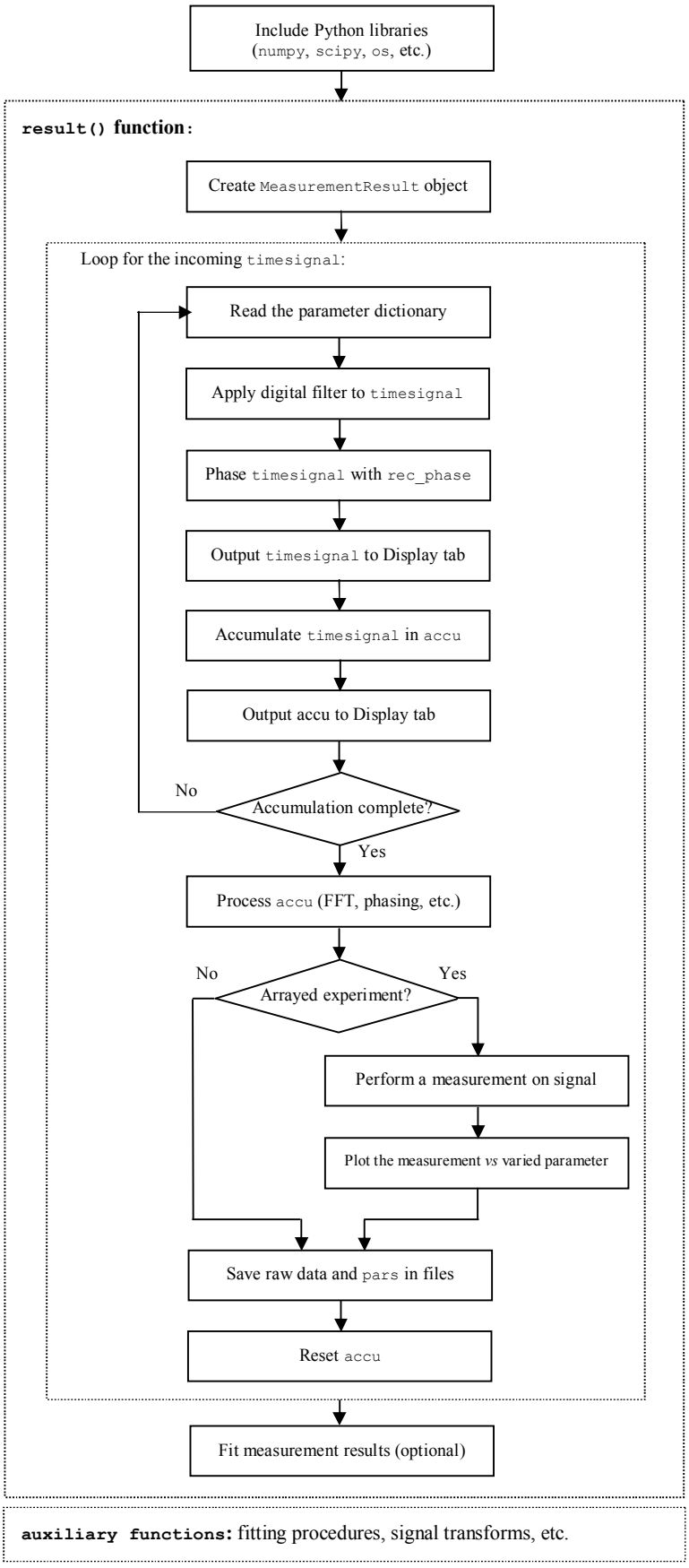


Fig. 4. Flowchart of a result script

3. Data output

The DAMARIS does not provide an interactive GUI for data processing. Hence, there is a demand for porting data into a third-party program. By default, the DAMARIS stores data in a binary HDF format in the data pool file specified in the Configuration tab, wherefrom the data are imported to an appropriate program for data analysis. The HDF data are accessible in the Python environment by means of the `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 several methods for in-line data writing from within result scripts, which include: `write_to_csv`, `write_to_hdf`, `write_to_simpson`, and `write_to_tecmag`.

By default, the method `write_to_simpson` is in use, considering a further processing with NMRnotebook, the NMR program which is installed on our DAMARIS spectrometers' PCs. A SIMPSON file is a two-column text file – one column for Re data and the other 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 required keyword argument – the path to the file specified by the parameters `DATADIR` and `OUTFILE`, and two optional keyword arguments – a data delimiter (by default " ") and a reference frequency (by default 100e6). SIMPSON data files get 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 mostly for 2D experiments; it is used in the `op_noesy_res.py` and `op_exsy2h_res.py` scripts. The method takes two required keyword arguments – the path to the file and the number of records (i.e. the number of samples in an indirect dimension F1), and few optional arguments – a reference frequency (`frequency`, by default 100), 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 resulting binary file have the extension `.tnt`. They are readable by NMRnotebook and can be imported in many other NMR processing software let alone the native NTNMR on our Tecmag spectrometer. The `write_to_tecmag` method allocates the space for all requested records in advance and closes the file after a new record written, so that one can start using the file before the 2D experiment ends.

4. Combining experiments

It is often desirable to run several experiments sequentially from within one script. The way in which DAMARIS jobs are executed allows for a straightforward concatenation of individual experiments' scripts, simply by copying-and-pasting them in one greater `experiment()` function. No modification is needed in the individual scripts' code unless you want for such a composite function to report the total experiment time. Once the Start button pressed, a time will be reported for the first experiment, and next such a report will come only after this first experiment is over. To change this behavior requires a substantial modification of the original scripts and makes 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 the simple copy-and-paste manner. What you still might change in the composite `experiment()` function 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 how treat the incoming result. The variable takes the name of the pulse program that is stored in the parameter `pars['PROG']` (assigned locally within pulse-program functions). Usually, there is no need to discriminate between experiments until it comes to measuring the signal vs. variable parameter. Then one checks `the_experiment` 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, stored in a common dictionary `measurements`. The experiment names serve as the dictionary's keys so that you can index it with `the_experiment`. Whenever a new experiment is added, you merely append to `measurements` an entry with a new experiment's name and specify either or both measurement and fitting procedures for this new experiment. That will be all.

In the folder `AU-Programs`, there are two examples of such composite scripts: one for diffusiometry, comprising saturation-recovery-with-solid-echo-detection, hahn-echo and stimulated-echo experiments, and one for experiments we typically carry out 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	Base 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's reference 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 a single (middle) point. The echo decay (T_2-decay) is fitted with a mono-exponential function, the best-fit parameters being reported on the Log page.

5.2. EXSY

Two-dimensional exchange spectroscopy.

File names: `op_noesy_exp.py, op_noesy_res.py`

Applications: measurements of slow exchange, either conformational or chemical, of magnetically non-equivalent spins when the rate of the exchange is greater than or of the same order as T_1 but less than the frequency difference between the spins.

Pulse sequence: $90^\circ - t_1 - 90^\circ - t_m - 90^\circ - Acq$

Phase cycle:

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

For the "cosine" data set, ψ= 0. For the "sine" data set, ψ = –90 (from M.H. Levitt's "Spin Dynamics", 2nd edition, page 530).

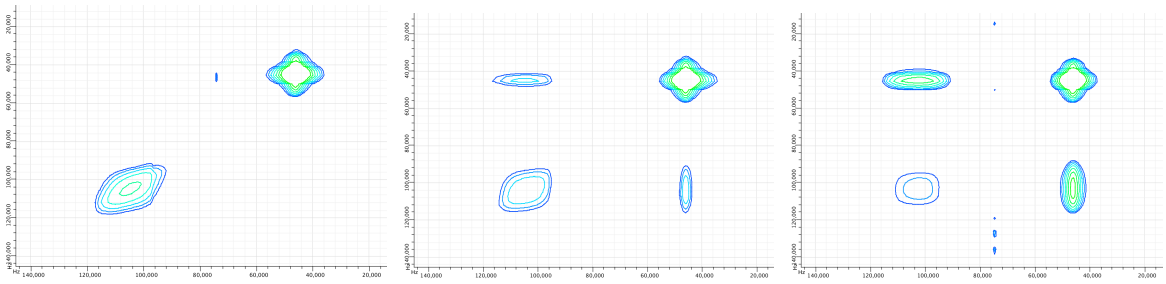
Acquisition parameters:

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 width, in either dimension	1 kHz - 1 MHz
SI1	Number of (complex) data points in F1	32 - 256
SI2	Number of (complex) data points in F2	128 - 16k
D8	Mixing time	from few μs to T_1
NS	Number of scans	≥ 8
DS	Dummy scans	0
RD	Recycle delay	3-5 T_1
DEAD1	Receiver's dead time	4-10 μs
PHA	Receiver's reference phase	to maximize Re

Comments: Technically, 2D EXSY is the same as 2D NOESY (the pulse sequences are identical). The method provides off-diagonal responses for spins under exchange, as the mixing time `D8` is varied. Presently, the indirect dimension F1 is sampled starting from t_1= 2 μs, which in

principle may be set as short as 0.5 µs, and with the increment $\Delta t_1 = 1/\mathtt{SW}$. Two data sets are recorded at each t_1 point with orthogonal `PH1` phases, inverting them each time when t_1 is incremented. This is referred to as the States-TPPI technique of quadrature detection in F1. The data are written in a Tecmag file format and can be processed as a regular 2D dataset with any NMR processing software which accepts Tecmag files (e.g. NMRnotebook). Should there be troubles with the States-TPPI processing option, you can make a dataset pure States by simply replacing 4 with 2 in the line 139 of the experiment script.

Example: The figures below are ^{19}F EXSY spectra of a LaF_3+5% SrF_2 crystal, recorded at three mixing times `D8`: 10, 100, and 1000 µs. The two diagonal peaks correspond to magnetically non-equivalent subgroups of fluorine ions in the crystal, while the cross-peaks appearing at `D8` = 10 µs indicate the chemical exchange between them on the corresponding timescale. The experiment parameters were: `P90` = 1.65 µs, `SF` = 338.7 MHz, `O1` = -57.0 kHz, `SW` = 150 kHz, `SI1` = 32, `SI2` = 128, `NS` = 16, `DS` = 0, `RD` = 2.5 s, `DEAD1` = 4 µs (spectrometer Birgit). The spectra were processed with NMRnotebook software, which included cosinebell apodisation in either dimension, manual phase correction, and automatic (polynomial) baseline correction.



`D8` = 10 µs, 100 µs 1000 µs

5.3. EXSY 2H

Deuteron two-dimensional exchange spectroscopy in solids.

File names: `op_exsy2h_exp.py, op_exsy2h_res.py`

Applications: the study of slow reorientation motion in polycrystalline and amorphous solids.

Pulse sequence: $90^\circ - t_1 - 54.7^\circ - t_m - 54.7^\circ - \Delta - 90^\circ - \Delta - Acq$

Phase cycle:

step:	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9-16*
φ_{90}	0	270	0	270	180	90	180	90	*invert*
$\varphi_{54.7}$	ψ	ψ+90	ψ	ψ+90	ψ	ψ+90	ψ	ψ+90	*repeat*
$\varphi_{54.7}$	ψ	ψ	ψ+180	ψ+180	ψ+270	ψ+270	ψ+90	ψ+90	*repeat*
φ_{90}	90	90	90	90	180	180	180	180	*invert*
φ_{rec}	0	180	180	0	90	270	270	90	*invert*

For the "cosine" data set, or odd `accu`'s, ψ= 0. For the "sine" data set, or even `accu`'s, ψ = 90.

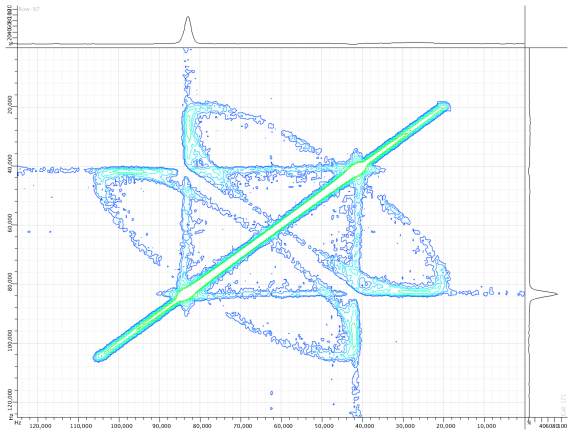
Acquisition parameters:

Parameter	*Description*	*Typical value*
P90	90° pulse length	few μs
SF	Base PTS' frequency	Larmor frequency
O1	Offset from SF	up to ±1 MHz
SW	Spectral width, in either dimension	1 kHz - 1 MHz
SI1	Number of (complex) data points in F1	32 - 256
SI2	Number of (complex) data points in F2	128 - 16k
D3	Position of refocusing 90°-pulse, Δ	10-30 μs
D4	Pre-acquisition delay	few μs
D8	Mixing time	from few μs to T_1
NS	Number of scans	≥ 16
DS	Dummy scans	0
RD	Recycle delay	3-5 T_1
PHA	Receiver's reference phase	to maximize Re

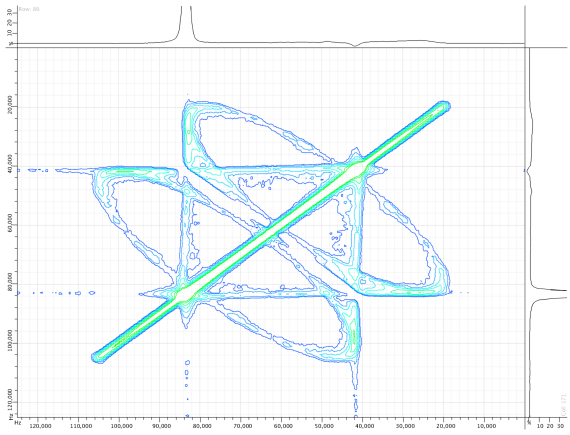
Comments: The experiment is based on the paper [JMR 79, 269-290 (1988)]. The phase lists are different from [JMR 79, 269-290 (1988)] though, but just as good. They have proved optimum for Zeeman-order and spin-alignment four-pulse sequences, which are constituents of ^{2}H EXSY, so I keep using them here. There is also a (formal) difference in the data processing, as suggested in a

GSim documentation. Namely, prior to being saved in a file, the "sine" signal is rotated by 90° so that a regular States algorithm for pure absorption 2D spectrum is applicable (within NMRnotebook or like NMR software). To balance differently relaxing cosine and sine signals, the sine data set is multiplied by a weighting factor varied between 1 and 2. The indirect dimension F1 is sampled starting from $t_1 = 0.5$ µs and with the increment $\Delta t_1 = 1/\mathtt{SW}$. The refocusing 90°-pulse can be placed as close to the 3rd pulse as the probe head dead time, in practice 10-30 µs. For the best performance, make sure that the pulse frequency `SF+O1` is in the centre of the ^{2}H spectrum and that the reference phase `PHA` is adjusted for a maximum Re.

Example: The figures below are ^{2}H EXSY spectra of dimethyl sulfone at 320 K, recorded with different number of scans – 16 and 512. The rest of parameters were: `P90` = 2.7 µs, `SF` = 46.14 MHz, `O1` = 1 kHz, `SW` = 125 kHz, `SI1` = 80, `SI2` = 256, `D3` = 10 µs, `D4` = 2 µs, `D8` = 3 ms, `RD` = 0.2 s, `PHA` = 65 (spectrometer Mathilda). The spectra were processed with NMRnotebook software, which included cosinebell apodisation in either dimension, manual phase correction, and symmetrization. No baseline correction was applied. The spectra exhibit the characteristic ellipse pattern of two-site jumps of CD_3 groups around a molecular C_2 axis on the millisecond scale.



NS= 16, experiment time: 8 min



NS= 512, experiment time: 4.5 h

5.4. 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° – *Acq*

Phase cycle:

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

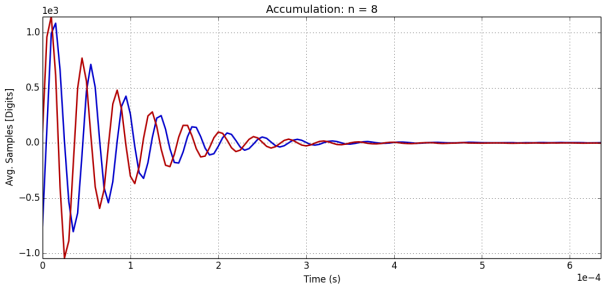
(standard CYCLOPS: cancels DC offset & averages channel imbalance)

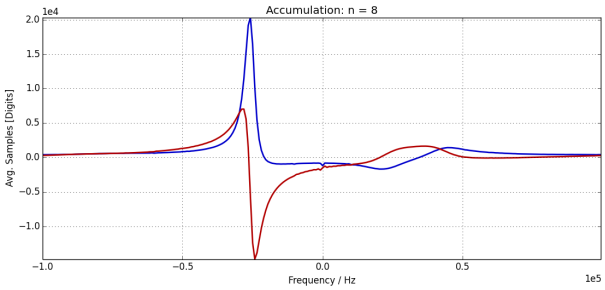
Acquisition parameters:

Parameter	*Description*	*Typical value*
P90	90° pulse length	few μs
SF	Base PTS' frequency	Larmor frequency
O1	Offset from SF	up to ±1 MHz
SW	Spectral 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: In the case of an arrayed experiment, the time signal is phased using the initial phase calculated on the first FID in the array. The time signal's intensity is measured vs. a variable parameter as a sum of either 32 first samples or those within the measurement range specified, on the Re-channel.

Example: FID and spectrum of ^{19}F in a mono-crystal of LaF_3+5% SrF_2, at room temperature. Experiment parameters were: `P90` = 1.7 us, `SW`= 200 kHz, `SI`= 126, `RD` = 3 s, `DEAD1` = 5 us, = 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 ^{19}F in LaF_3, so that an interactive phasing with a third-party software is necessary.


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

5.5. 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: Suppressing 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)]

5.6. 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	variable

Comments: none

5.7. Miscellaneous

"Go-setup" experiment

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

Applications: Interactive adjustment of acquisition parameters; shimming

Pulse sequence: 90° – *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: A non-stop pulse-acquisition without accumulation. Three measurements are reported after each shot: FID amplitude, spectrum integral and spectrum middle frequency (center of gravity).

5.8. Saturation-Recovery

Saturation-recovery experiment

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

Applications: T_1 relaxometry; T_1-filtering of FID signals

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

Phase cycle:

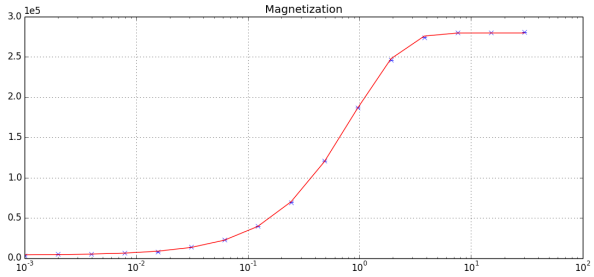
	step:	*1*	*2*	*3*	*4*
	φ_{90}	0	0	0	0
(CYCLOPS)	φ_{90}	0	180	90	270
	φ_{rec}	0	180	90	270

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×T_1
DEAD1	Pre-acquisition delay for NMR coil ringing	5-10 μs
PHA	Receiver phase	arbitrary
NECH	Number of saturation pulses	10-40
D1	First interval in saturation pulse train	100 ms
D2	Last interval in saturation pulse train	100 μs

Comments: The saturation sequence comprises from 10 to 40 pulses with intervals decreasing logarithmically from 100 ms down to 100 μs. The FID amplitude is measured against a variable parameter (usually `TAU`) as a sum of either 32 first samples or those within the measurement range specified, on the Re-channel. For the best performance, maximize Re by adjusting `PHA` and set `SW` high enough to make sure that the measured samples are all positive. If it is `TAU` that varies, the recovery curve will be fitted with a mono-exponential function, the best-fit parameters being reported on the Log page.

Example: T_1-relaxation of ^{19}F in a LaF_3+5% SrF_2 crystal, 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.



5.9. 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 in solids; T_1-weighting of 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

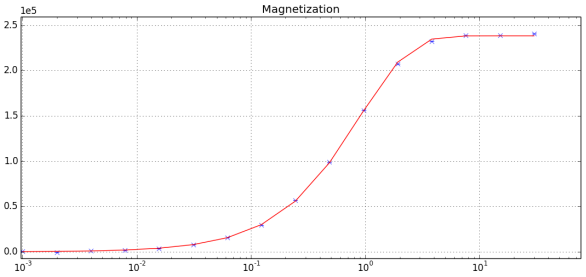
(CYCLOPS; cancellation of *z*-component present before the 3rd pulse [from a Tecmag program])

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 $5\times T_1$
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 from 10 to 40 pulses with intervals decreasing logarithmically from 100 ms down to 100 μs. The solid echo is usually acquired with the echo delay (parameter `D3`) 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 (parameter `D4`) to localize the echo maximum. The echo intensity is measured the same way as in 5.8.

Example: T_1-relaxation of ^{19}F in a LaF_3+5% SrF_2 crystal, 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.



5.10. Solid Echo

Two-pulse solid (quadrupole) echo acquisition.

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

Applications: acquisition of spectra in solids 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

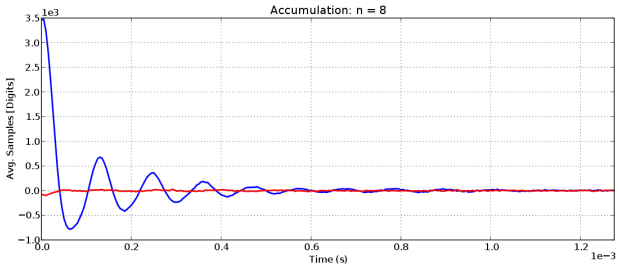
(CYCLOPS; cancellation of *z*-component present after 1st pulse [same as in Tecmag])

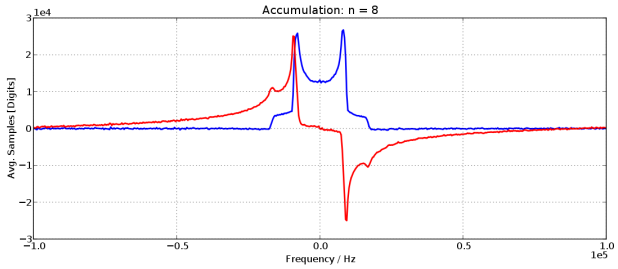
Acquisition parameters:

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 usually acquired with the echo delay (parameter `TAU`) set as short as the receiver dead time. The acquisition starts from the top of the echo. It might be necessary to vary the echo pre-acquisition delay (parameter `D4`) to localize the echo maximum., which is positioned by varying `D4`.

Example: A ^{2}H spectrum of hexamethylbenzene-d_{18}, at room temperature. 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 (spectrometer Eis).


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


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

5.11. Spin Alignment

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

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

Applications: To probe slow molecular motion in solids ("sine-sine" correlation measurements).

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

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

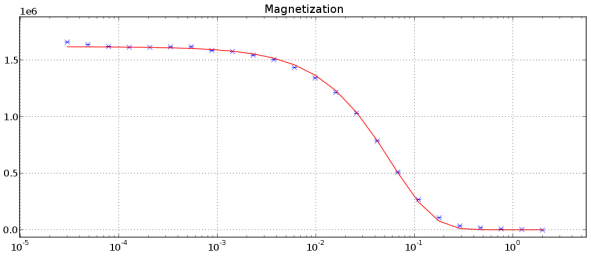
Acquisition parameters:

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: The typical use is to measure intensity of the echo arising at `D1` after the last pulse as of function of the mixing time `D2`. The echo intensity is measured from either 32 first samples in Re-channel or within the range specified in the beginning of the result script. To maximize Re, adjust `PHA` using a reference phase `phi0` (one obtained from the experiment with the shortest `D2`). The echo attains its maximum intensity when `D1` ~ $(2\delta)^{-1}$, δ being a quadrupolar/dipolar spectrum width. However, in practice, one usually keeps `D1` as short as the receiver's dead time. 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 echo-vs-`D2` measurement, the result script performs a KWW (stretched exponential) fit to the data. Not applicable to spins-3/2 and up, for which cases use `spinal32_experiment` pulse program.

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



5.12. Spin Alignment Four Pulses

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

File names: `op_spinal4pulses_exp.py, op_spinal4pulses_res.py`

Applications: To probe slow molecular motion in solids ("sine-sine" correlation).

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

Phase cycle:

step:	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9-16*
φ_{90}	0	270	0	270	180	90	180	90	*repeat*
φ_{45}	90	180	90	180	90	180	90	180	*repeat*
φ_{45}	90	90	270	270	0	0	180	180	*repeat*
φ_{90}	90	90	90	90	180	180	180	180	*invert*
φ_{rec}	0	180	180	0	90	270	270	90	*repeat*

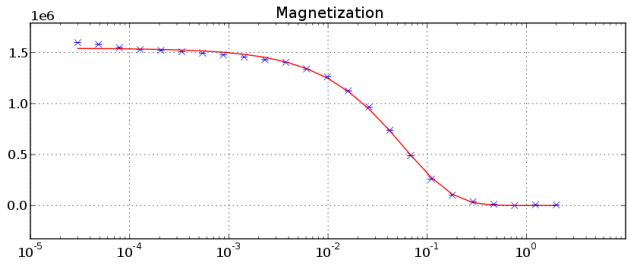
(CYCLOPS; suppresses of single- (SQ) and double-quantum (DQ) coherences present after the second pulse; cancels T_1-recovery)

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	10-100 μs
D2	Mixing time, t_m	from μs to T_1
D3	Refocusing pulse delay, Δ	10-30 μs
PHA	Receiver phase	arbitrary

Comments: The forth 90°-pulse applied at $t = \Delta$ after the 3rd pulse generates two anti-phase echoes at $2\Delta - t_p$ and $2\Delta + t_p$, the latter echo being acquired. Using a pulse extra allows for t_p shorter than the receiver's dead time, which is beneficial for <…>. Note, however, that at t_p shorter than the reciprocal line width, the two echoes overlap severely, thereby making the t_p-dependence of the measured echo's intensity ambiguous.

Example: ^{2}H STE decay in hexamethylbenzene-d_{18}, at room temperature. Parameters of the experiment: `SF` = 46.704 MHz; `P90` = 4.2 us; `D1` = 30 us; `D3` = 20 us; `RD` = 0.5 s; `NS` = 16 (spectrometer Eis). The red line is a mono-exponential fit with the time decay constant of 59 ms.



5.13. Spin Alignment: Spin-3/2

STE acquisition after storing in a quadrupolar order, for spin-3/2 nuclei (^{7}Li, ^{11}B).

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

Applications: To probe slow molecular motion in solids ("sine-sine" correlation).

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]

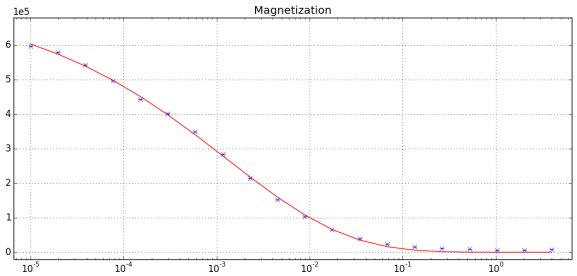
step:	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*
φ_{90}	0	180	0	180	90	270	90	270	90	270	90	270	180	0	180	0
φ_{45}	90	90	270	270	0	0	180	180	180	180	0	0	90	90	270	270
φ_{45}	0	0	0	0	180	180	180	180	90	90	90	90	270	270	270	270
φ_{rec}	180	0	0	180	180	0	0	180	270	90	90	270	270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

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

Acquisition parameters: same as for 5.11

Comments: The only difference from 5.11 is in a phase cycle, which is made here twice as long to cancel also the triple-quantum coherence arising after the second pulse.

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



5.14. 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

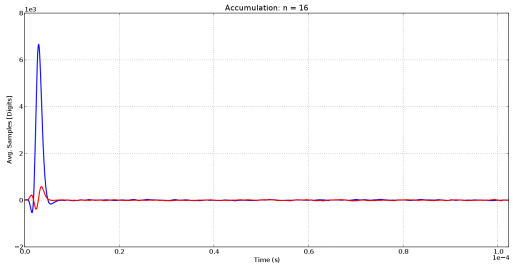
(suppresses SQ coherences present after the second pulse; cancels T_1-recovered magnetization; CYCLOPS)

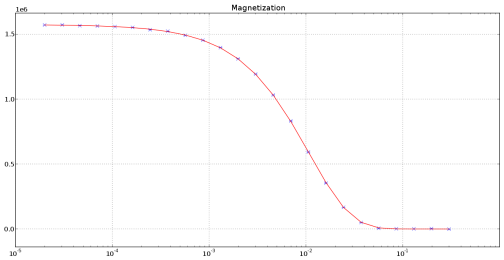
Acquisition parameters:

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: This experiment's scripts are identical to Stimulated Echo (see below). Due to an intrinsically poor SNR and short FID's, an echo intensity is always measured as a sum of all echo points within a given measurement range. When it is `D2` which is varied (a usual scenario), the echo decay is fitted with a mono-exponential function.

Example: (Upper) ^{1}H STE in tetradecane, in the static field gradient of 75 T/m (spectrometer Magnex). Parameters of the experiment: `P90` = 0.8 us, `SF` = 161.75 MHz, `NS` = 16, `RD` = 6 s, `D1` = 20 us, `D2` = 30 us. (Lower) A sum of echo points in the range 2.1 to 4.6 μs vs. `D2`. The red line is a monoexponential fit with the decay time constant of 10 ms.





5.15. Steady Gradient Spin Echo with CPMG Detection

Stimulated-echo sequence with a CPMG readout of the echo 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)

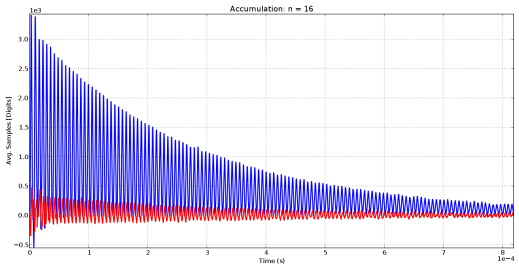
Acquisition parameters:

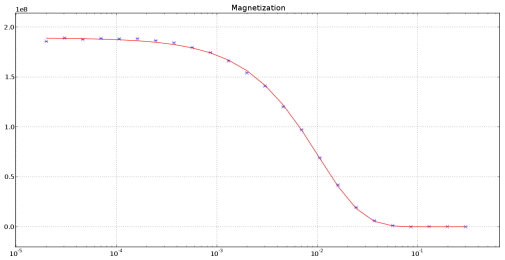
Parameter	*Description*	*Typical value*
P90	90° pulse length	few μs
SF	PTS' frequency	Larmor frequency
O1	Offset from SF	up to ±1 MHz
NS	Number of scans	≥ 16
DS	Dummy scans	0
RD	Delay between scans	3-5×T_1
D1	Delay after the 1st pulse (t_p)	10-20 μs
D2	Delay after the 2nd pulse (t_m)	from few μs to T_1
D4	Pre-acquisition delay	few μs
NECH	Number of 180°-pulses in the CPMG train	up to few hundreds
TAU	Pulse half-period in the CPMG train	> 40 μs
PHA	Receiver phase	arbitrary

Comments: Instead of acquiring a single echo, one refocuses the magnetization over again with a 180°-pulse CPMG train. Multiple echoes are acquired and added up to increase signal-to-noise ratio. When it is `D2` which is varied (an usual scenario), the cumulative echo's decay is fitted with a

mono-exponential function. It is important to position CPMG echoes at the centre of an acquisition window, for which purpose the parameter `D4` is used.

Example: (Upper) A ^{1}H CPMG signal in tetradecane in the static field gradient of 75 T/m (spectrometer Magnex). Parameters of the experiment: `P90` = 0.8 us, `SF` = 161.75 MHz, `NS` = 16, `RD` = 6 s, `D1` = 20 us, `D2` = 30 us, `D4` = 2.5 us, `NECH` = 128, `TAU` = 50 us, , `PHA` = -190. (Lower) A sum of all echoes above a 10% level (the "noise level") vs. `D2`. The red line is a monoexponential fit with the decay time constant of 10 ms.





5.16. Steady Gradient Spin Echo with XY-16 Detection

Stimulated-echo sequence with a XY-16 readout of the echo signal

File names: `op_sgse16_exp.py, op_sgse16_res.py`

Applications: Diffusiometry

Pulse sequence: $90^\circ_x - t_p - 90^\circ_x - t_m - 90^\circ_x - t_p - [t_e - 180^\circ_{XY-16} - t_e - Acq]_n$

Phase cycles:

Outer cycle [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}	0	0	0	0	0	0	0	0	270	270	270	270	270	270	270	270
φ_{rec}	0	180	180	0	180	0	0	180	270	90	90	270	90	270	270	90

Inner cycle (XY-16) [JMR 101A, 320 (1993)]

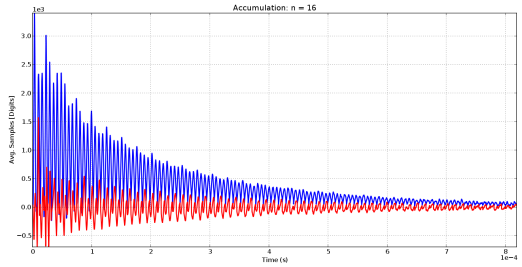
step:	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*
φ_{180}	0	90	0	90	90	0	90	0	180	270	180	270	270	180	270	180
φ_{rec}	90	270	270	90	270	270	90	90	90	270	270	90	270	270	90	90

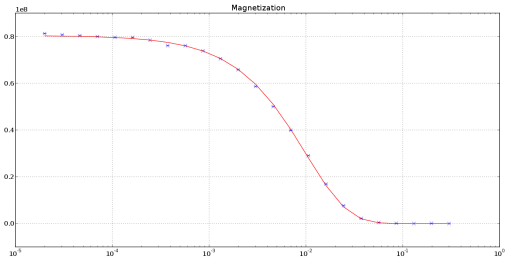
Acquisition parameters:

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
D4	Pre-acquisition delay	few μs
NECH	Number of 180°-pulses in the CPMG train	up to few hundreds
TAU	Half pulse period in the CPMG train, t_e	> 40 μs
PHA	Receiver phase	arbitrary

Comments: A phase of 180°-pulses is no longer constant, as in the CPMG train, but changes from pulse to pulse according to a XY-16 cycling scheme. This allows preserving both *x*- and *y*-components of the transverse magnetization upon refocusing and thus compensates for RF amplitude imbalance [JMR 101A, 320 (1993)]. It is the only script where pulse phases are cycled within a single scan, in addition to the overall phase cycling, hence the two phase lists. Using XY-16 requires keeping track of receiver phases for each echo (line 181 in the experiment script) and rotating them individually on the result script's side (lines 30-41 there).

Example: (Upper) ^{1}H STE in tetradecane as read out by a XY-16 pulse train, in the static field gradient of 75 T/m (spectrometer Magnex). Parameters of the experiment: `P90` = 0.8 μs, `SF` = 161.75 MHz, `NS` = 16, `RD` = 6 s, `D1` = 20 μs, `D2` = 30 μs, `D4` = 2.5 μs, `NECH` = 128, `TAU` = 50 μs. (Lower) A sum of all echoes above a 10% level (the "noise level") vs. `D2`. The red line is a monoexponential fit with the decay time constant of 10 ms.





5.17. 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

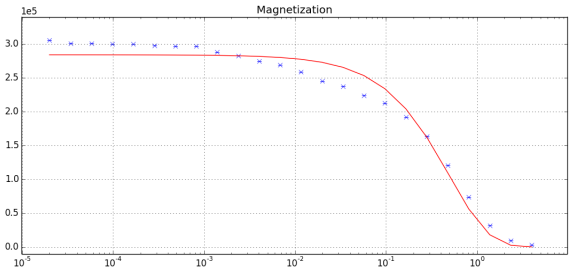
(suppresses SQ coherences present after the 2nd pulse; cancels T_1-recovered magnetization after 3rd pulse; CYCLOPS)

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 employed in diffusiometry (see 5.11 and 5.12) and other experiments that involve a z-storage of the prepared magnetization (a mixing time parameter `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 result in an enhanced signal at short `D2`'s when measuring on rigid-like spectral components.

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.



5.18. 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)

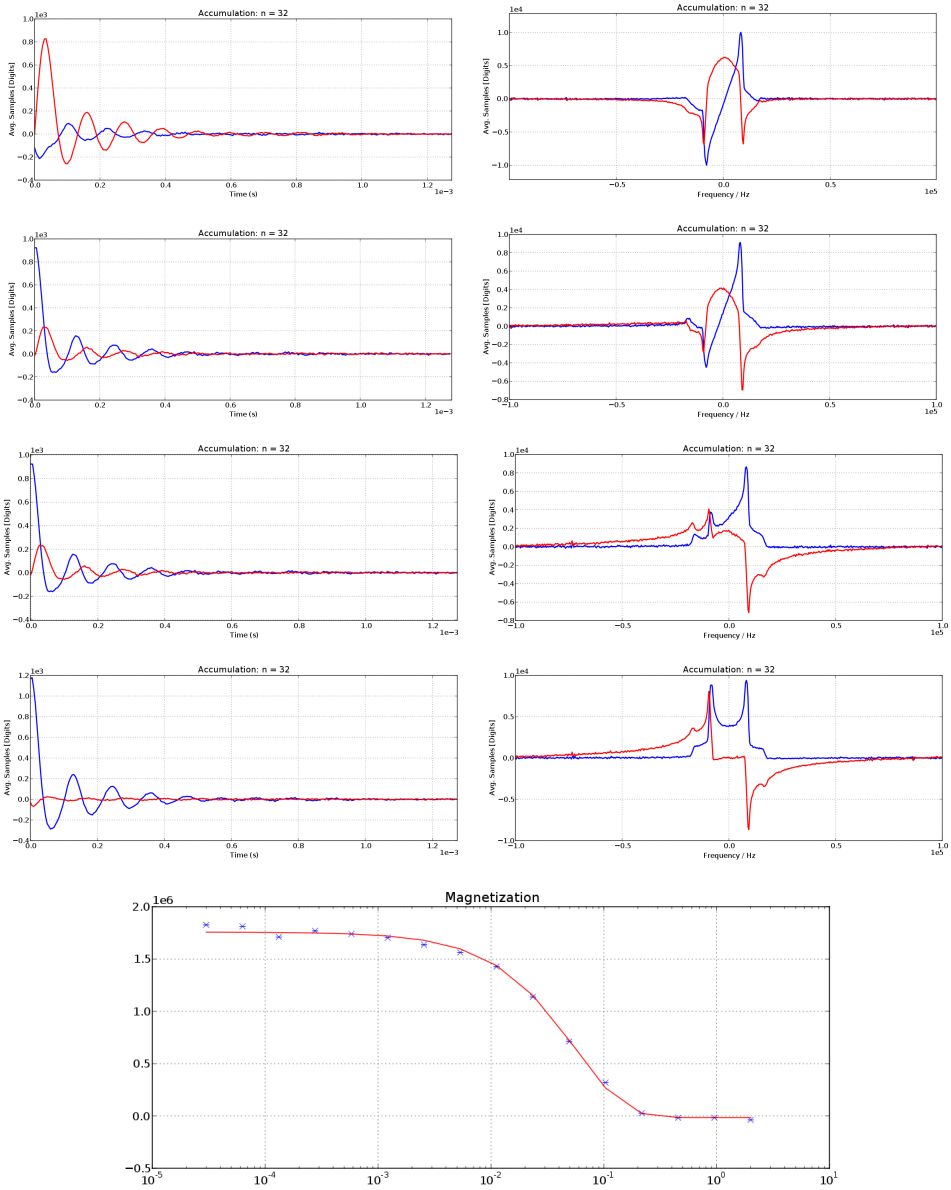
Acquisition parameters:

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 through a progressive 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.



5.19. T_2-Filtered ZZ-Exchange

A preliminarily filtered short-T_2 component is built up during z-storage due to an exchange with other spins.

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, $\beta = 90°$, which is optimum for solid-like (Gaussian) peaks. If, however, the longer-T_2 peak which is supposed to pass through the T_2-filter has a Lorentzian line

shape, use a 180°-pulse instead ($\beta = 180°$). This will minimize the zz-exchange between longer- and shorter-T_2 components during `D1`. Because of T_1-relaxation, the total signal intensity decreases with increasing `D2`. Thus, the measurable exchange rates are limited between $1/T_{2\text{short}}$ and $1/T_1$.

5.20. 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 ("cosine-cosine" 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)

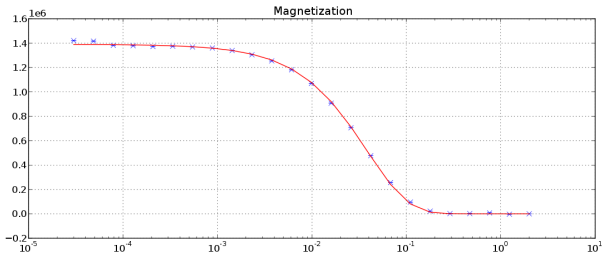
Acquisition parameters:

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, one adjusts `PHA` using a reference phase `phi0` obtained from the first of arrayed experiments (that is, one 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 “echo vs D2“ measurements the result script fits a KWW (stretched exponential) function 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, D2 is varied from 30 us to 2 s. The red line is a mono-exponential fit with the decay time constant of 39 ms.



5.21. Zeeman Order Four Pulses

Stimulated spin-echo after storing in Zeeman order with a refocusing pulse.

File names: `op_zeeman4pulses_exp.py, op_zeeman4pulses_res.py`

Applications: Studying slow molecular motion in solids ("cosine-cosine" correlation)

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

Phase cycle:

step:	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9-16*
φ_{90}	0	270	0	270	180	90	180	90	*repeat*
φ_{90}	0	90	0	90	0	90	0	90	*repeat*
φ_{90}	0	0	180	180	270	270	90	90	*repeat*
φ_{90}	90	90	90	90	180	180	180	180	*invert*
φ_{rec}	0	180	180	0	90	270	270	90	*repeat*

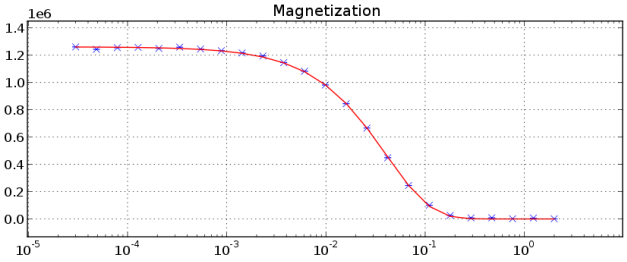
(suppresses single- and double-quantum coherences present after the second pulse; cancels T_1-recovered magnetization; CYCLOPS)

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	10-100 μs
D2	Mixing time, t_m	from μs to T_1
D3	Refocusing pulse delay, Δ	10-30 μs
PHA	Receiver phase	arbitrary

Comments: Same as 5.20 but with an extra 90°-pulse. The pulse is applied at $t = \Delta$ after the 3rd pulse and generates two in-phase echoes at $2\Delta - t_p$ and $2\Delta + t_p$, the latter echo being acquired. This allows using t_p shorter than the receiver's dead time, which is beneficial for <…>. Note, however, that at t_p shorter than the reciprocal line width, the two echoes begin to overlap which may interfere with the studied t_p-dependence of the measured echo.

Example: ^{2}H STE decay in hexamethylbenzene-d_{18}, at room temperature (spectrometer Eis). Parameters of the experiment: `SF` = 46.704 MHz; `P90` = 4.2 µs; `D1` = 30 µs; `D3` = 20 µs, `RD`=0.5 s; `NS` = 16, `D2` is varied from 30 µs to 2 s. The red line is a mono-exponential fit with the decay time constant of 41 ms.



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 its present form for those to whom NMR is a new technique, while experienced users who tend to write their own scripts might 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 both for extending the scripts with new functionality and in combining experiments in one greater script.

Given time, new scripts may 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.