Documentation to SPIDYAN v1.0

June 3, 2015

Stephan Pribitzer
stephan.pribitzer@phys.chem.ethz.ch

Contents

1. SPIDYAN

With the recent developments in the field of fast arbitrary waveform generators (AWGs) in the GHz range, it has become possible to use frequency swept microwave pulses in EPR. Such frequency swept, or passage, pulses are well known in NMR spectroscopy while their effects in EPR are yet to be fully understood.
Common EPR programs were not designed to simulate EPR experiments with arbitrary shaped pulses that may also require consideration of spectrometer hardware parameters, such as a resonator profile or AWG resolution. Waiting for extension by the developers of these packages would have hold back our own developments of experimental techniques and of theory of passage pulses. Likewise, NMR programs would have needed extensions for convenient computation of pulse EPR experiments. In order to simulate spin dynamics during frequency-swept, or generally arbitrary wavefunction pulses, the SPIn DYnamics ANalysis (SPIDYAN) library was developed, which runs on MATLAB. SPIDYAN is free of cost and open source. The first release of SPIDYAN can treat one-spin and two-spin systems with arbitrary spin quantum numbers. At the current stage of development of EPR

spectroscopy with shaped pulses, this appears appropriate. A future version will allow for systems with an arbitrary number of spins.

1.1. Installation

SPIDYAN runs in MATLAB and requires MATLAB R2013a or newer. The current version of SPIDYAN also depends on the MATLAB Signal Processing Toolbox and for improved performance the Parallel Computing Toolbox is recommended.
To install SPIDYAN, the program has to be downloaded from www.epr.ethz.ch/software and extracted into a directory of choice. For testing it is possible to write and run scripts from that directory. To install SPIDYAN permanently and make it accessible from everywhere on the computer, the program's directory has to be added to the MATLAB search path. This is done by clicking on the 'Set Path' button in the 'environment' section of the home tab and picking 'Add Folder…' to navigate to the location of SPIDYAN. Now MATLAB will be looking for the SPIDYAN functions in the specified directory. SPIDYAN scripts can now be run from any directory on your machine.

1.2. Using SPIDYAN

Any SPIDYAN script is divided in three segments:

Set up of Input:

- `system` – a structure which contains the spin system
- `sequence` – a structure with the pulse sequence
- `options` – a structure containing all other parameters required for the simulation

Initializing, Processing and Propagation:

- call of `triple` with `sequence` and `options`, creates the structure `experiment`
- call of `setup` with `system` and `options`, returns the updated structure `system`
- call of `homerun` with `system`, `experiment` and `options` to simulate the `experiment`

Returned Output:

- density matrix `state` and time traces `detected_signal`, which can be plotted

Structure arrays are a data type in MATLAB which allow the user to store several variables in a convenient and easily accessible way. A structure consists of fields, which can be any other data type (integers, doubles, vectors, strings, cell arrays or structures). The fields of structures can be called with '`structure.field`'. For example the field `.sqn1` (which is the spin quantum number of the first spin) in the structure system, can be set with:

```
system.sqn1=0.5;
```

The following sections provide detailed information on how to set up the input and what fields are available. Some example files can be downloaded together with SPIDYAN and are discussed and explained in the Tutorial section of the documentation.

2. User created input

2.1. Structure `system`

The structure `system` provides SPIDYAN with the spin system. The required and optional parameters are resonance frequencies for up to two spins, spin quantum numbers, relaxation times and coupling constants. The structure is processed by `setup` which also does a rudimentary check for completeness.

Table 1: Fields required for the structure system, not all are required for SPIDYAN.

User created fields of `system` with units			
`.nu01`	`GHz`	`.nu02`	`GHz`
`.sqn1`		`.sqn2`	
`.T1`	`ns`	`.T2`	`ns`
`.default_T1`	`ns`	`.default_T2`	`ns`
`.A`	`GHz`	`.B`	`GHz`
`.init_state`			

Table 2: Fields that are created by SPIDYAN from the input.

Fields created from input	
`.sops`	`.spins`
`.ham`	`.eq`
`.R`	`.gamma`

The fields `.nu01` and `.nu02` are the resonance for in the AWG frame. If simulations are to be carried out with one spin only, `.nu02` must not be used, has to be empty or `.spins` set to 1. The value of `.spins` tells SPIDYAN the number of spins in the system. It does not need to be configured since SPIDYAN can usually guess the value from the input.
The spin quantum numbers can be specified with `.sqn1` and `.sqn2`. SPIDYAN assumes spin quantum numbers $S_i = ½$, if not declared otherwise.
With those parameters SPIDYAN calculates the Hamiltonian `.ham` of the system, which is required to propagate the density matrix, from the spin operators `.sops` (created by SPIDYAN). For those not satisfied with the available Hamiltonians in SPIDYAN, it is also possible to create tailored Hamiltonians. To do this, one first has to build the Hamiltonian in matrix form in Hilbert space and then assign it to the field `.ham`. When SPIDYAN discovers an assignment to `.ham`, it skips calculation of the Hamiltonian and propagates with the user's input.
The initial state `.init_state` of the system can be given as matrix or magnetization vector. The matrix is used as it is, while a magnetization vector of the form (S_x, S_y, S_z) is used to build a spin density matrix from the Pauli matrices for a single spin ½. If the initial state is omitted, SPIDYAN assumes thermal equilibrium.
For simulations, which include relaxation effects, the relaxation times for longitudinal and transverse relaxation have to be given in nanoseconds, otherwise they can be omitted. Relaxation times can be provided either through assigning doubles or matrices to `.T1` and `.T2`, or by using a graphical user interface (GUI). The GUI is described in section 2.1.1. Transitions which are not defined by the user are assigned default values which have to be provided with `.default_T1` and `.default_T2`.
SPIDYAN then uses the relaxation times to build the relaxation super operator $\hat{\Gamma}$, which is required for propagation in the Liouville space. If `.spins=1` and `.sqn1=1/2`, SPIDYAN also calculates the relaxation matrix `.R`, which can be used for computationally more efficient simulations including relaxation in Hilbert space, if the system consists of only a single $S = ½$ spin.

An equilibrium state can be set with the field `.eq`, which has to have the form of a density matrix. If the field is not assigned, SPIDYAN assumes the equilibrium state to be $-S_z$ for a single electron or $-S_{1z} - S_{2z}$ for two spins[1].

2.1.1. How to set up relaxation times

SPIDYAN offers two methods to realize relaxation input. The first, very general approach is to apply the same longitudinal and transverse relaxation time to all transitions of the system. To do this the user assigns doubles to `system.T1` and `system.T2`. This can be useful to get a first insight into relaxation dynamics.

A second, more sophisticated method allows to assign relaxation times to individual transitions. For this the input needs to be in matrix form, with dimensions $(2S + 1)$ for a single spin and $(2S_1 + 1)\ (2S_2 + 1)$ for two spins respectively.

Figure 1 shows the transitions for a single spin with $S = 1$, their positions in the matrix of the relaxation times and a MATLAB code. The first column is always the state with the lowest spin projection quantum number m_S. For example a $S = 1/2$ starts with $m_S = -1/2$.

The same is valid for a two-spin system. Here, the first position in the matrix contains both spins in their lowest state. First m_S of the second spin changes (see Figure 2), and after all combinations, the spin quantum number of the first spin is increased. Only the upper triangle of the matrix has to be defined. The user does not need to assign the lower triangle. SPIDYAN will change the relaxation time for every non-defined (0) transition to `.default_T1` or `.default_T2`.

The elements on the diagonal of the matrix do not correspond to relaxation pathways and therefore do not need to be considered.

In addition, SPIDYAN offers the possibility to enter the relaxation times through a simple GUI. This can be done by setting `options.ui` to 1. When the script is executed and relaxation is switched on, a window pops up, where the user can fill in the longitudinal relaxation times in a table. The rows and lines of the table follow the same ordering as described previously. By closing it, the values are stored and a second window pops up for transverse relaxation. Only after the second window is closed, the simulation continues.

[1] Caution, this in not valid for a system with one electron and a nucleus and has to be adapted accordingly in such a case.

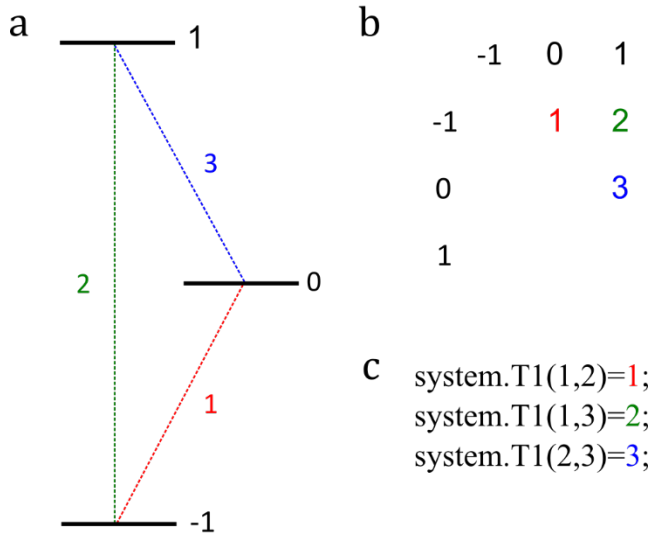

Figure 1: a) Transitions and b) the corresponding positions in the matrix for the relaxation times for a three level system with S=1 with c) A MATLAB code.

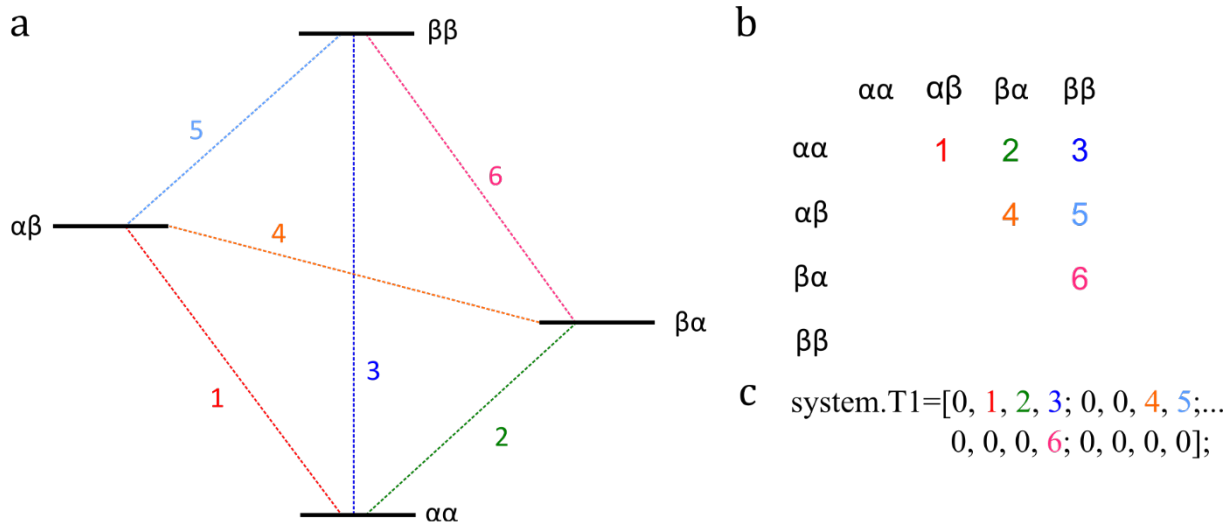

Figure 2: a) Transitions and b) their corresponding positions in the matrix for the relaxation times for a four level system with S=½ and I= ½ and c) MATLAB code for a matrix.

2.2. Structure sequence

The structure `sequence` is used to define the events[2] during a pulse experiment. It contains fields for the event lengths `.tp`, pulse amplitudes `.nu1`, flip angles `.beta`, critical adiabaticity factors `Q` and excitation bandwidths `.frq`. It is also possible to assign rise times to each pulse with the field `.t_rise` and phase cycling can be added with `.pcycle`. An overview of all available fields can be found in Table 3.

The time lengths of all events, regardless of their type, have to be written into the vector `.tp`. SPIDYAN uses the vectors for pulse amplitudes `.nu1`, adiabaticity factors `Q` and flip

[2] SPIDYAN counts pulses, inter-pulse delays and detection as events.

angle .beta to distinguish between pulses and free evolution. If the program finds a non-zero value for an event on the corresponding positions in either .nu1 .beta or .Q (or all), it takes it for a pulse.

Table 3: The structure sequence.

Fields of structure sequence with units				
.tp	ns		.nu1	MHz
.beta	rad		.frq	GHz
.t_rise	ns		.phase	rad
.Q	Vector with critical adiabaticity of pulse.			
.excite	Vector or cell array containing Boolean vector			
.pcycle	cell array containing matrices			
.pcycle{k}	matrix, first dimension phase of pulse, second dimension is sign			

If pulses are to be created with a certain flip angle, the corresponding element in .nu1 is 0, else the flip angle is ignored and the given .nu1 is used. The same holds for creation through adiabaticity factors, in which case .nu1 and .beta have to be zero.
The excitation bandwidth .frq can either be assigned to be the same for every pulse or different for each pulse. The same bandwidth is assigned to every pulse when .frq contains only one (rectangular pulse) or two (chirp) elements. The first element is the initial and the second element the final sweep frequency. For excitation with monochromatic pulses, the first and second element simply have to be the same or the vector must contain only one element. For example

```
sequence.frq=[1.0 1.0];
```

leads to the same behavior, an offset of 1 GHz from the frequency of the simulation frame, as

```
sequence.frq=[1.0];
```

To assign the same chirp bandwidth to every pulse, one would write

```
sequence.frq=[0.5 1.5];
```

which creates a sweep from 0.5 to 1.5 GHz.
For some experiments it is necessary to have different excitation bandwidths for every pulse. This can be achieved by writing .frq as a cell array, with the cells being vectors with initial and final sweep frequency, e.g. for a three-pulse experiment with two rectangular pulses (1st and 4th event), an inter pulse delay (2nd event) and a chirp pulse (3rd event):

```
sequence.frq ={[1.0], [], [0.5 1.5], [1.0 1.0]};
```

The phase of the pulses can be set either for all as the same by typing

```
sequence.phase = pi;
```

or separately for each pulse by making `sequence.phase` a vector (example for the above pulse sequence):

```
sequence.phase = [pi/2, 0, 0, pi/2];
```

The phase has to be given in rad.
Since calculations are usually carried out in a frame that rotates with the local oscillator (LO) frequency (AWG frame [3]), nuclear spins could be erroneously excited in the simulation although they are off resonant by the LO frequency. Therefore it is possible to denote the spin to be excited for each pulse with `.excite`. This will set up the excitation operator accordingly. In case `.excite` is a vector, the same excitation scheme is applied to every pulse. Each pulse can configured individually by writing `.excite` as a cell array, with index being the event number. The required input is a Boolean vector, where the position in the vector corresponds to the number of the spin as defined in `system` where 1 means 'excitation' and 0 'no excitation'. If the field is omitted (either because `.excite` is empty or not defined for individual elements in the cell array), only the first spin is excited. It is also possible to give a custom excitation operator in matrix form, to excite for example only a selected transition in multi-level system.

```
sequence.excite=[1 1];
```

as well as

```
sequence.excite={[1 1], [], [1 1]}
```

excite both spins during both pulses (1st and 3rd event). A custom excitation operator can be given in a similar same way. In the following example both pulses are excited during the first pulse and during the third pulse only the first spin is excited (the excitation operator is in matrix form):

```
sequence.excite=[1 1], [], [0 0 0.5 0; 0 0 0 0.5; 0.5 0 0 0; 0 0.5 0 0]}
```

Phase cycling is controlled with the cell array `.pcycle`. The individual cell elements hereby correspond to the respective event. The cells are matrices with the rows corresponding to phase cycle steps. The first value in each row is the phase of the pulse in radians. The second value is a signal weighting. The density matrix resulting for the corresponding phase is multiplied with this number and added to the weighted density matrices from the other steps. Finally, the sum of the weighted density matrices of all steps is normalized by the sum of the absolute values of all weighting factors. For example the code

```
sequence.pcycle{3}=[0, 1; pi, -1];
```

will, starting from the same initial state, cycle the third event by calculating the outcome of the pulse once without any additional phase and once with a phase of π,. The two resulting spin density matrices (expectation value traces) are processed according to the second

[3] In the frame rotating with the LO frequency, frequencies that are output by the AWG are offset frequencies.

value in the corresponding row of `.pcycle`, i.e. the second signal is subtracted from the first and the total signal is divided by 2.
For an example see the Tutorial section at the end of this documentation.

2.3. Structure `options`

With `options` simulation parameters are adjusted, which are not covered by `system` and `sequence`. It is possible to declare detection operators, tweak SPIDYAN, so that it runs faster, include a resonator and explore the effect of complex excitation or of the choice of the frame.

Table 4: The structure options.

User created fields of structure `options` with units			
`.det_op`	cell array	`.no_detection`	vector
`.relaxation`	0/1	`.propagator`	0/1
`.awg`	structure	`.resonator`	structure
`.direct_evolution`	vector	`.down_conversion`	0/1
`.cutoff_freq`	MHz	`.no_dc`	vector
`.display_filter`	0/1	`.ui`	0/1
`.labframe`	0/1	`.complex_excitation`	0/1
`.LO`	GHz		

The field `.det_op` is a structure which contains the detection operators in string or matrix form. From a string SPIDYAN builds the corresponding detection operator from the spin operators of the system. Not supported detection operators can be added directly as matrices as elements of `.det_op`. For a thorough discussion on the syntax of detection operators in SPIDYAN please refer to section 2.3.1. The function `setup` creates the detection operators in matrix forms and stores them in the cell array `options.detect`.
Detection is computed by multiplication of the density matrix with the detection operator and calculating the trace of the resulting matrix. Calculations can therefore be made more efficient if events, which are not of interest, are excluded from detection by writing the event number into the vector `.no_detection`. If expectation values are of no interest at all, `.det_op` can be left empty. This is useful if only the density matrix describing the state of the system after the sequence is required.
By default SPIDYAN propagates the density operator during all events, including free evolution, on the time grid of the AWG. However, direct propagation in one large time step during such events can drastically reduce computation time. For events where the exact trace is of no interest (e.g. inter-pulse delays) `.direct_evolution` can be set to 1. While saving a lot of time, this can also lead to transients in the detected signals when down conversion is applied, as described below.
Including relaxation into spin dynamics for multiple spins or spins >½ requires that the simulations are run in Liouville space. This in turn leads to larger matrices, slowing down the calculations. Therefore, SPIDYAN neglects relaxation if not explicitly requested by setting `.relaxation` to 1. If the statement is true SPIDYAN calculates the super operators for the Hamiltonian and the relaxation matrix `system.gamma`, which are required for simulations in the Liouville space. By default SPIDYAN uses the Liouville space for

relaxation effects, but for a $S = ½$ system it is also possible to propagate in the Hilbert space. This can be done by setting `.propagator` to 1 while `.relaxation` is true.
To propagate in the Liouville space SPIDYAN requires relaxation matrices as input, and those matrices can easily become very complex to set up. Therefore the program offers the user to assign relaxation times through a simple user interface. To use this, the value for `.ui` has to be set to 1. For more information please refer to section 2.1.1 where the relaxation matrix is explained.
Simulations are usually conducted in the AWG frame, which is a frame rotating at the frequency of `options.LO` and resonance frequencies are given as offsets. Not only does this help to speed up calculations by making it possible to use larger time steps, it also minimizes numerical errors from matrix exponentials by avoiding large off-diagonal matrix elements in the Hamiltonian. The time grid can be changed with the field `.awg`, a structure controlling the arbitrary waveform generator. The default values for the AWG can be found in Table 5. The field `.s_rate` controls the sampling rate in GS/s and `.vert_res` the vertical resolution in bit. The user does not need to assign every field, only the ones he wants to change. SPIDYAN assumes default values for missing fields.
In cases where it is of interest to simulate in the lab frame `.labframe` can be set to true.[4] If so, the simulation is carried out in a static frame and the resonance frequencies of the spin are offset by `.LO`. If the sampling rate of the AWG is not sufficient to fulfill the Nyquist condition for the excitation pulse it is automatically increased.
To remove oscillations at the frequency `system.nu01` from the signal detected in the lab or AWG frame, the signals may be down converted and filtered with almost no phase distortion. This behavior can be switched on and off with the option `.down_conversion`. If true, SPIDYAN returns all processed signals as well as the original traces. To exclude only certain signals from being down converted[5] the user can write the index of the signal (position of the corresponding detection operator in `options.det_op`) or its label (e.g. `'S1p'`) into the vector `.no_dc`. The low pass filter is tuned with the parameter `.cutoff_freq`, which is the cutoff frequency of the filter in MHz. The filter response function can be displayed by setting `.display_filter` to 1.
In some cases it may be interesting to use complex excitation pulses, i.e. a circularly polarized excitation field in the rotating frame. This is possible by setting `.complex_excitation` to 1. Since SPIDYAN cannot use its speed up techniques here and has to calculate the propagator for every time point separately, this has to be handled with care and should not be used as a default option. Complex excitation pulses are implemented for Hilbert space only, which means relaxation has to be switched off.
For simulations that take a resonator profile into account, the resonator has to be given in the structure `.resonator`. It contains a field `.active`, with the indices of events to apply the resonator profile to. Elements which correspond to events other than pulses are ignored. Other fields are the sampling rate of the resonator profile `.s_rate`, the quality factor `.Ql` and the resonance frequency `.f0`. From this SPIDYAN can calculate a simple resonator profile in the frequency domain by modelling an RLC circuit[1]. It is recommended to set

[4] This may be necessary if one is unsure about truncation of off-diagonal terms of the laboratory-frame Hamiltonian.
[5] This can be the case if a system contains an electron and a nuclear spin. At the moment SPIDYAN cannot differentiate between electron and nuclear spins and will apply the same down conversion and filtering to both. In the case when coherence on the nuclei is detected, this will introduce oscillations to the signal, as the simulation frame rotates for the electron spin, but not for the nuclear spin.

`.resonator.s_rate` to at least ten times `.resonator.f0`. An experimentally obtained resonator profile can be used by writing the frequency axis to `.range` and the resonator profile to `.nu1`. SPIDYAN also contains a routine, which allows to compensate the pulse shape for the resonator[1] by setting the field `.comp_bw` in the `.resonator` structure to true. Since the resonator profile is given at the frequency in the lab frame SPIDYAN down-converts it from the lab to the AWG frame. The required frequency is set with `options.LO` and has to be present for simulations including a resonator.

Table 5: Default values of the AWG.

AWG	
`.s_rate`	`12 GS/s`
`.vert_res`	`10`
`.channels`	`1`

Table 6: Fields of the structure resonator.

Resonator fields with input	
`.active`	vector with events
`.Ql`	
`.range`	vector in GHz
`.comp_bw`	0/1
`.s_rate`	GS/s
`.f0`	GHz
`.nu1`	MHz

2.3.1. Configuration of the detection operator

SPIDYAN can recognize and build a broad variety of detection operators from strings. This includes the common Pauli spin matrices, raising and lowering operators, as well as population and coherence operators for individual transitions. It is also possible to provide a detection operator in matrix form.
The input is the cell array `.det` in the structure `options`. There is no limitation on the number of detection operators.
A spin operator starts either with `'-S'` or just `'S'`. This is followed by a letter, where `'x'`, `'y'` or `'z'` stand for the Pauli matrices, `'m'` and `'p'` for the lowering S^- and raising S^+operators and `'e'` represents unity. This leads to the following set of operators for a single spin:

```
Sx, Sy, Sz, Sp, Sm, Se, -Sx, -Sy,…
```

The syntax in a two-spin system is similar to the one for a single spin. The capital `'S'` is followed by two letters for the two spin operators

```
Sxx, Sxy, Sxz, Syy, Syx, Syz, Szz, Szy, Sxz, Spm, Sxp,…
```

where the first letter denotes an operation on the first spin (defined in `system.sqn1`) and the second on the second spin (`system.sqn2`).
To observe populations or magnetization changes on selected transitions or levels for a high-spin system or a two-spin system the detection operators can be further modified. By appending a number to the S_z operator, SPIDYAN detects populations, e.g. in a $S = 3/2$ system `'Sz2'` tracks the $m_S = -1/2$ level. SPIDYAN supports one-digit numbers only, meaning the maximum number of observable levels is nine.
Transitions are detected by adding a second digit. Sticking with the $S = 3/2$ system,

`Sx23`, `Sy23` or `Sp23`

can be written to observe coherence of the central transition ($m_S = -½ \leftrightarrow m_S = ½$) or

`Sz14`

for the polarization between the first ($m_S = -3/2$) and the fourth level ($m_S = 3/2$) with.
For single spin operators in a two-spin system, it is possible to write

`S1x, S2x, S1z, S2p,…`

where the number represents the spin.
Populations can be detected with

`S1z1, Szz3, S2z2,…`

and coherence and raising/lowering operators for selected transitions have the form

`S1x23, Sxx23, Sp23, Sm14, Szz14… .`

3. Output

The output of SPIDYAN consists of two variables `state`, `detected_signal` and an updated version of `experiment`.
The variable `state` is the state of the system after the pulse experiment in form of a spin density matrix.
The structure `detected_signal` contains the time traces of the experiment. If detection was not switched off it contains the fields `.sf`, `.dc`, `.t` and `.signals`, else an empty cell is returned.

Table 7: The fields of the output structure `detected_signal`.

`.sf`	`.signals.sf`
`.dc`	`.signals.dc`
`.t`	`.signals.t`

The vectors `.sf`, `.dc` are *n*×*k*-dimensional matrices, where *n* is the number of detection operators and *k* the length of the signal. They contain the time traces of the signal in the simulation frame and after down conversion (only available if `.down_conversion` is true). The corresponding time axis can be found in `.t`.
In the structure `.signals` the time traces (simulation frame and down converted) and time axes of each individual event are stored in cell arrays.

4. SPIDYAN runtime

A simplified environment scheme of SPIDYAN is displayed in Figure 3. After setup of the three structures `system`, `options` and `sequence`, these are processed with the two functions `setup` and `triple`. While `setup` takes care of the system, by creating spin density matrices and the Hamiltonian in matrix form, building spin operators, detection operators and

setting up the relaxation matrix, `triple` checks `sequence` for consistency and creates the pulse shapes.
Although it would have been possible, these two routines and `homerun` were not combined into a single function. That way more flexibility and insight into the simulation is provided for the user. Furthermore, certain parameters, such as Hamiltonians, pulse strengths or inter-pulse delays can still be changed after building the pulses or `system`, but before the simulation.
For propagation, `homerun` has to be called with the structures as returned from `setup` and `triple`. The output of `homerun` contains the final state as a density matrix, the time traces and the pulse sequence.
The high-level functions `setup`, `triple` and `homerun` augment input variables of the type structure with additional fields, depending on the information that was already provided. This may lead to unintended behavior if these functions are called repeatedly, for instance in a loop. If changes in the structures `system`, `options` and `sequence` are intended, these structures should be cleared and newly initialized.

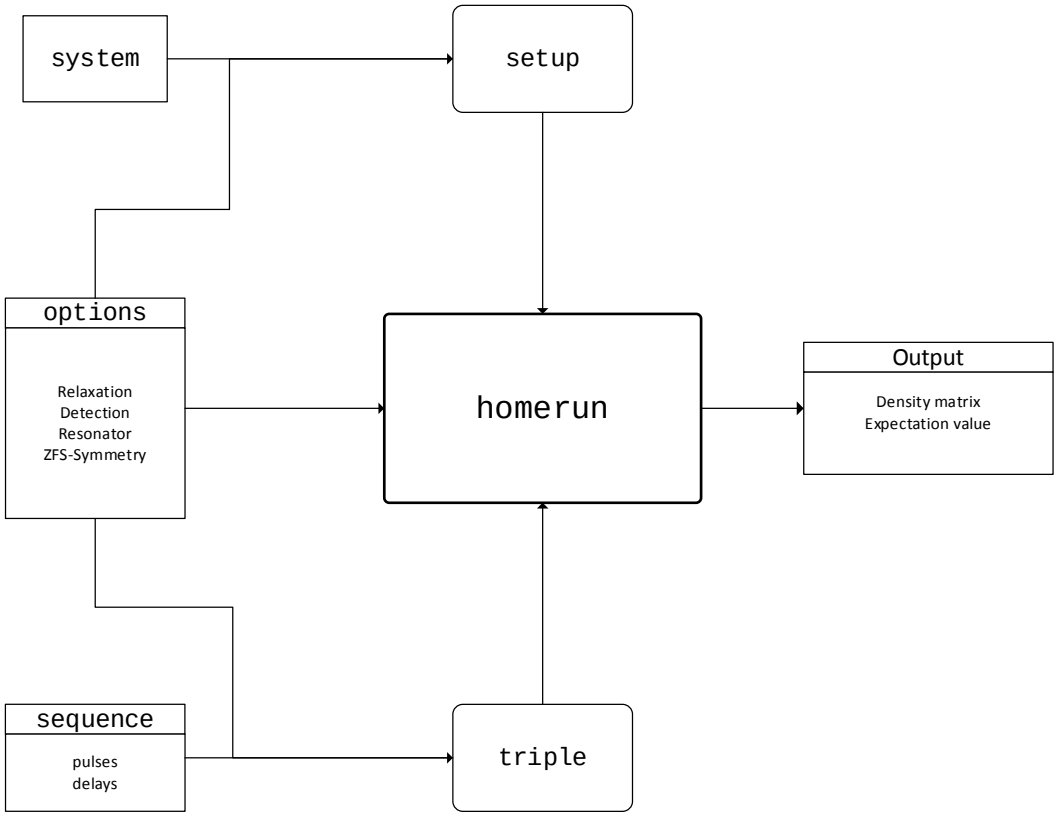

Figure 3: SPIDYAN runtime.

5. Tutorial

5.1. A basic pulse experiment

The script `tweak.m` is a very short and simple SPIDYAN script. It contains a section for the spin `system`, one for the `options` and one for the `sequence`. By default the simulation is carried out for a single spin with spin quantum number ½. This can be extended to a two-spin system where a nucleus couples to the electron spin with the secular coupling constant *A* and pseudo-secular coupling constant *B*, by uncommenting the lines marked as second spin. The script runs a two pulse experiment, with an inter-pulse delay of 20 ns and a detection time of 40 ns.
First the pulse sequence is defined. It is created with the structure `sequence`:

```
sequence.tp=[50,20,40,40];            % event durations
sequence.beta(1)=pi;                  % flip angle of the first pulse
sequence.beta(3)=pi/2;                % flip angle of the 2nd pulse/third event
sequence.frq={[0.5 1.5]};             % excitation band
sequence.phase=0;                     % no phase shift for all pulses
sequence.excite={[1 0],[],[1 0]}      % both pulses excite the first spin only
```

In the example the experiment consisting of 4 events with durations of 50, 20, 40 and 40 ns is run. By creation of the field beta, SPIDYAN is told which of these events are the pulses (1 and 3) and what flip angles they have. First a 180° pulse with a duration of 50 ns is applied to the system (SPIDYAN calculates the pulse amplitudes required for the specified flip angle), which is followed by a 20 ns delay, a 90° pulse of 40 ns duration and a free evolution interval of 40 ns duration, during which the signal is detected. During both pulses frequency is linearly swept from 0.5 to 1.5 GHz.
Next is the structure `options`: The first set of parameters

```
options.relaxation=0;
options.det_op={'Sp','-Sz'};
```

tells SPIDYAN to switch off relaxation, and observe the system through the S^+ and $-S_z$ operator.
The second set of parameters specifies signal processing:

```
options.down_conversion=1;
options.cutoff_freq=100;
options.no_dc={};
options.display_filter=0;
```

Down conversion is on and the cutoff frequency is 100 MHz. All signals are being down converted and the program is not displaying the frequency response of the applied filter.
The following parameters can be used to speed up the simulation.

```
options.no_detection={3};
options.direct_evolution=[2];
options.ui=0;
options.complex_excitation=0;
```

With `.no_detection` it is possible to exclude certain events from detection (in this case the second pulse = 3rd event), which can speed up the calculation by avoiding the computationally expensive computation of the matrix trace. Stepwise evolution is switched off during the second event, which is going to be the inter-pulse delay. This greatly increases performance and should be used whenever possible. The user interface for editing relaxation times is switched off and complex excitation is switched off. Both these choices correspond to default behavior, and would not need to be explicitly defined.
The next step is the definition of the spin system. At first the simulation is to be carried out with one spin only, a second spin will be added later. The resonance frequency of the electron is set to be 1 GHz in the AWG frame. The spin quantum number is ½, longitudinal relaxation time 5000 ns and transversal relaxation time 1000 ns.

```
system.nu01=1;          % resonance frequency of spin in GHz
system.sqn1=0.5;        % spin quantum number
system.T1=5000;         % longitudinal relaxation time in ns
system.T2=1000;         % transversal relaxation time in ns
```

An initial state can be given in form of a magnetization vector:

```
system.init_state=[0 0 1];    % magnetization vector, this corresponds to Mz
```

If no initial state is specified, SPIDYAN automatically assumes thermal equilibrium in the high temperature limit.
Before running the simulation SPIDYAN needs to process the input with

```
[experiment, options] = triple(sequence, options);
[system, state, options] =setup(system, options);
```

where `triple` creates the actual pulse sequence and `setup` builds spin operators, Hamiltonians in matrix form, detection operators in matrix form and an initial state from the magnetization vector.
The returned structures are used to call

```
[state, detected_signal, experiment, options,~]=homerun(state, system,
experiment, options, []);
```

which propagates the system. The output provided are the final state, returned as a density matrix, and a structure which contains the signals with and without down conversion, a time axis and the signals for each event separately. `Homerun` also returns the structure experiment, which is updated during the simulation, the updated structure `options`.
Finally, the signals are plotted, both as detected in the AWG frame and after down conversion and filtering:

```
figure(1),clf           % plots the first figure
hold on
plot(experiment.taxis,real(detected_signal.sf))
legend(options.det_op,'Location','NorthEast')

figure(2),clf           % plots the second figure
```

```
hold on
plot(experiment.taxis,real(detected_signal.dc))
legend(options.det_op,'Location','NorthEast')
plot([experiment.tp(1) experiment.tp(1)],[-1 1],'r--')
plot([experiment.tp(2)+experiment.tp(1) experiment.tp(2)+experiment.tp(1)],[-1
1],'r--')
plot([experiment.tp(3)+experiment.tp(2)+experiment.tp(1)
experiment.tp(3)+experiment.tp(1)+experiment.tp(2)],[-1 1],'r--')
```

The result of this simulated experiment after down conversion is depicted in Figure 4.

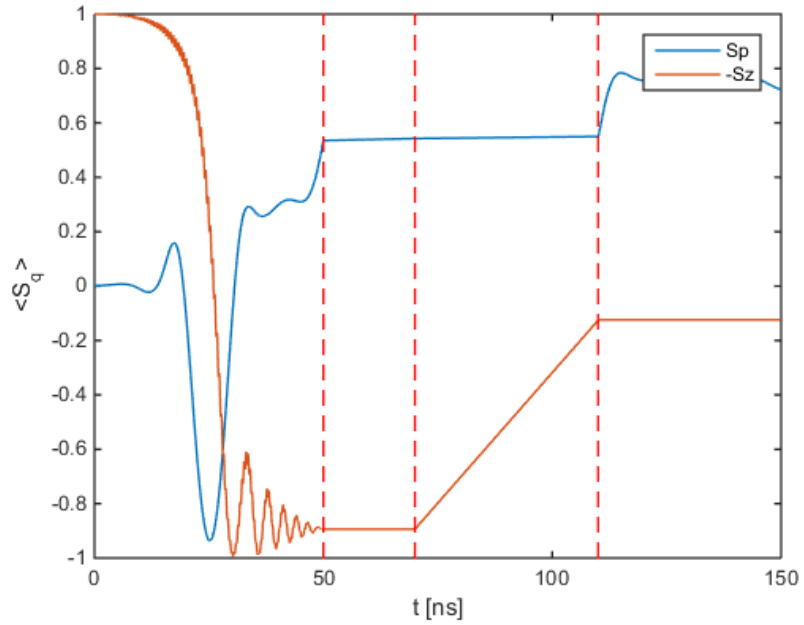

Figure 4: Time trace of a simple two pulse element after down conversion. Events are separated by vertical, dashed, red lines. The blue line is the expectation value over the course of the experiment for the raising operator and the orange line the expectation value for the negative S_z operator.

This example can be extended to a two-spin system (`tweak-2spin.m`), by uncommenting the lines

```
system.sqn2=0.5;
system.nu02=-0.014;
system.A=0.014;
system.B=0.009;
options.det={'Sp', 'S1z','S2p'};
options.no_dc={3};                          % or options.no_dc={'S2p'};
```

before the call of `setup`. Here a second (nuclear) spin with spin quantum number ½ and a resonance frequency of 14 MHz is created. The coupling parameters A and B are 14 MHz and 9 MHz. The detection operator needs to be updated to compute the information of interest for the two-spin system. If spin operators for a single spin system (`Sp`) are used in an experiment with two spins, SPIDYAN can in some cases correct this. In the present case the operator `Sp` is going to be changed to `S1p` by the program. Nevertheless, to avoid unintended behavior, we advise to use the explicit form.

If down conversion is switched on, SPIDYAN applies down conversion to every signal with oscillating components (off diagonal elements in the detection operator), but since the nucleus rotates at a different frequency, down conversion would then introduce a new oscillation or remove the entire signal. Therefore down conversion is switched off for the third signal. Alternatively one can also write the string of the detection operator that is to be excluded from down conversion.
If we want to include phase cycling for both pulses, the line

```
sequence.pcycle{3}=[0, 1; pi/2, -1; pi, 1; 3*pi/2, -1]; % cycling the second pulse
```

has to be added to the section of the pulse definitions. This phase cycle enforces coherence order changes $\Delta p = 1$ for the first and $\Delta p = 2$ for the second pulse, thus selecting only the echo signal. The resulting phase cycle is depicted in Table 8.

Table 8: Example phase cycle for two pulses, specifying the phases of the two pulses and the detector.

$P_1(\pi/2)$	$P_2(\pi)$	detection
x	x	y
x	y	-y
x	-x	y
x	-y	-y

5.2. The echo experiment

The file `echo_template.m` is a script which simulates a simple two-pulse echo.

```
% Define spin distribution for #1 spin
c=1;              % center of Gauss distribution in GHz; resonance frequency
sigma=0.05;       % standard deviation of Gauss distribution
nu0_s=0.9;        % beginning frequency [GHz] of spin distribution
nu0_e=1.1;        % end frequency [GHz] of spin distribution
st=0.0002;        % frequency step [GHz]between two spin packets
nu0_vec=nu0_s:st:nu0_e;
p=exp(-((c-nu0_vec)/sigma).^2); % calculates distribution probability(Gaussian)
p=p/trapz(p);     % normalization of probabilities
```

It assumes a Gaussian distribution of spin packets, centered around 1 GHz, with a standard deviation of 0.05 GHz. The vector `nu0_vec` contains 1001 spin packets, with resonance frequencies ranging from 0.9 GHz to 1.1 GHz, each with individual probability `p`. After the spin packet distribution, next the spin system is set up.

```
system.sqn1=0.5;
system.sqn2=0.5
system.nu02=-0.014;
system.A=0.014;                % secular hyperfine term A in GHz
system.B=0.009;                % pseudo-secular hyperfine term B in GHz
system.T1=[0,10000,5000,5000;0 0 5000 5000; 0 0 0 10000; 0 0 0 0];
system.T2=[0,5000,1000,1000;0 0 1000 1000; 0 0 0 5000; 0 0 0 0];
```

```
system.init_state=[0 0 1];      % starting point of magnetization;
```

This is exactly the same as in the previous example, with the only difference that no resonance frequency for the first spin is required, since it will be added from `nu0_vec` in the loop and the relaxation times are now input in a matrix form. The relaxation times are now input in matrix form, where the elements in the matrix correspond to the transitions as depicted in Figure 2. The same is valid for `options`, only this time relaxation is switched on from the beginning.

```
options.relaxation=1;
options.propagator=0;
options.cutoff_freq=100;
options.det_op={'S1p'};
options.no_detection={};
options.down_conversion=0;
```

This spin echo is a two-pulse experiment, with a 90° pulse followed by a 180° pulse. In order to refocus the coherence of all spin packets with chirp pulses, the second pulse must last half as long as the first pulse and the frequency band needs to be the same.

```
sequence.tp=[200,100,100,500];
sequence.beta(1)=pi/2;
sequence.beta(3)=pi;
sequence.frq={[0.5 1.5]};
sequence.phase=[0 0 0];                    % no phase shifts
sequence.excite=[1 0];                % pulses excite only the first spin

[experiment, options] = triple(sequence, options);
```

After the input is completed, the `for`-loop is processed. It is possible in this example to use a `parfor` loop (parallel-for), which uses the multicore processing ability of MATLAB and decreases the computation times by a factor of almost the number of cores. For more details see the MATLAB documentation on parallel computing. In parallel computing threads have to be independent of each other, meaning that loops cannot communicate with each other. This makes it impossible to calculate the total signal (from all spin packets) within the loop. Therefore a cell structure has to be created first in which the detected signals of all spin packets are stored. Only after the `parfor`-loop is completed the total signal is calculated from the individual traces.

```
signalc=cell(1,length(nu0_vec));          % creates empty cell structure
```

For the loop instances to be independent it is necessary to change the system parameters in every loop separately, by creating a system structure `systeml`, which is, after addition of the resonance frequency of the distributed spin, processed by `setup` within the loop.

```
parfor k=1:length(nu0_vec)

    systeml=system;
    systeml.nu01=nu0_vec(k);

    [systeml, state, optionsl]=setup(systeml,options);
```

```
    [state, detected_signal]=homerun(state, systeml, experimentl, optionsl);

    signalc{k}=detected_signal.sf;
end
```

After simulation of each spin packet, the total signal is created by summing up the weighted traces. At this point the probability comes into play, which acts as a weighting factor for each individual signal.

```
signal=zeros(size(signalc{1}));

for k=1:length(nu0_vec)
    signal=signal+signalc{k}*p(k);
end
```

After creation of a time axis the echo is plotted.

```
figure(22); clf;
hold on
plot(t,real(signal))
xlabel('time [ns]')
plot([experiment.tp(1) experiment.tp(1)],[-1 1],'r--')
plot([experiment.tp(2)+experiment.tp(1) experiment.tp(2)+experiment.tp(1)],[-1
1],'r--')
plot([experiment.tp(3)+experiment.tp(2)+experiment.tp(1)
experiment.tp(3)+experiment.tp(1)+experiment.tp(2)],[-1 1],'r--')
```

To remove oscillations the signal can be down converted by calling `strike` with the signal in the simulation frame and the center frequency of the spin distribution and plot the converted signal:

```
signalf=strike(signal, t, c, options);

figure(23); clf;
hold on
plot(t,abs(signalf))
xlabel('time [ns]')
plot([experiment.tp(1) experiment.tp(1)],[-1 1],'r--')
plot([experiment.tp(2)+experiment.tp(1) experiment.tp(2)+experiment.tp(1)],[-1
1],'r--')
plot([experiment.tp(3)+experiment.tp(2)+experiment.tp(1)
experiment.tp(3)+experiment.tp(1)+experiment.tp(2)],[-1 1],'r--')
```

This results in a non-oscillatory echo (Figure 5).

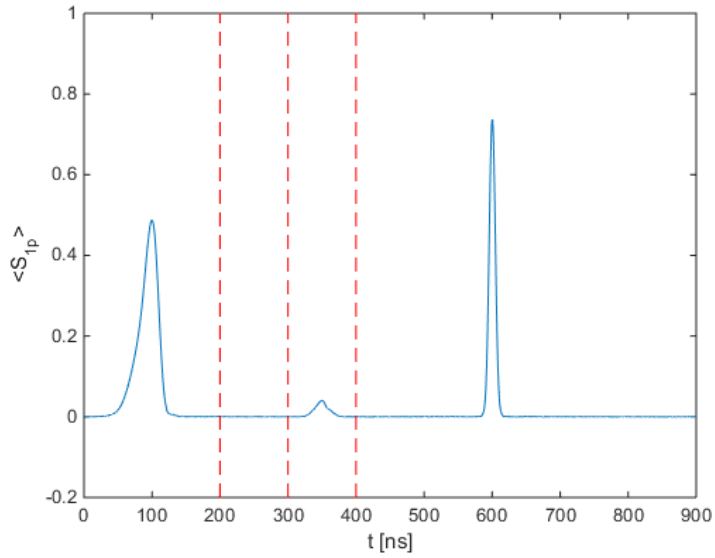

Figure 5: A two-pulse echo after down conversion. The events are separated by vertical, dashed, red lines. Observed was the raising operator for the first spin for the entire experiment. The first segment corresponds to the π/2 pulse and the third element to the π pulse.

6. References

1. Doll, A., Pribitzer, S., Tschaggelar, R. & Jeschke, G. Adiabatic and fast passage ultra-wideband inversion in pulsed EPR. *J. Magn. Reson.* **230,** 27–39 (2013).

2. Schweiger, A. & Jeschke, G. *Principles of pulse electron paramagnetic resonance*. 608 (Oxford University Press, 2001).