# Documentation for SPIDYAN

by 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 [1–8]. Such frequency swept, or passage, pulses are well known in NMR spectroscopy but their effects in EPR are yet to be fully understood [9]. 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 program can treat arbitrary spin system and allows design of custom pulse sequences.
This documentation serves as a help to work with SPIDYAN. A more in depth discussion of frequency swept pulses and the theory behind the program can be found in our paper [10].

## 1.1 Installation

SPIDYAN runs in MATLAB and requires the R2013a version 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 anywhere on your machine.

## 1.2 Using SPIDYAN

Most SPIDYAN scripts can be divided into 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 pulses
  - 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`
  - time traces, which can be plotted and processed

Structure arrays are a data type in MATLAB which allows 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 even structures). The fields of a structure can be called with `structure.fieldname`. For example the field `.sqn` (which is a vector with the spin quantum numbers of the spins in the system) in the structure `system`, can be set with:

```
system.sqn=0.5;
```

Section 2 provides detailed information on how to set up the input and what fields are available. If you are only interested in learning how to run SPIDYAN download the program and head over to Section 5, where a few example simulations are given.

## 1.3 Simulation frames

In our own experimental setup we use a fast AWG with a sampling rate of up to 12 GSa/s. The such generated pulses are usually centered at about 1.3 GHz [5]. The AWG-output is converted up to X-band (or other) frequencies by mixing with a local oscillator (LO) element (8 GHz at X-band). After interaction with the sample, the detected signal is converted down, resulting in spin echoes oscillating around 1.3 GHz, and digitized at 1.5 or 2 GSa/s.
In SPIDYAN, simulations can be run in the laboratory frame or in the LO frame, which rotates at the frequency of the LO.

### 1.3.1 LO-frame

Due to the Nyquist criterion, propagating in the laboratory frame at 9.5 GHz (X-band) would require a step size of 52.6 ps. But since all pulses have a certain bandwidth, it is not advisable to set the sampling rate of the AWG to exactly the double of the required Nyquist frequency and a step size of 40 ps is more beneficial.

Such laboratory-frame propagation has the advantage that aliasing, which may cause apparent excitation of transitions outside the excitation band, is avoided. However, propagating on the time grid of the laboratory frames also comes with certain disadvantages: The time step must be chosen smaller than the AWG time resolution. In cases where the center frequency of the excitation band is much larger than the excitation bandwidth, the time step must be chosen much smaller than the relevant dynamics of the spin system. This can easily increase the computation time by an order of magnitude.
A more severe disadvantage of computing in the laboratory frame are potential roundoff errors during calculation of matrix exponentials. These are caused by large off-diagonal elements in the Hamiltonian, as they occur during pulses. They can be overcome by further reducing the time step, which in turn increases computation time further. Another error arises from amplitude deviations due to time discretization. These can as well be minimized by increasing the sampling rate of the simulation beyond the minimum required Nyquist frequency. An example for this effect can be found in Section 5.3.
By transferring to the LO frame, the minimum required Nyquist frequency is reduced and the errors discussed above can be avoided at a smaller computational effort, as done for instance in [9]. Indeed, comparisons show (Fig. 1) that it is advantageous to conduct simulations in the LO frame. The labo-

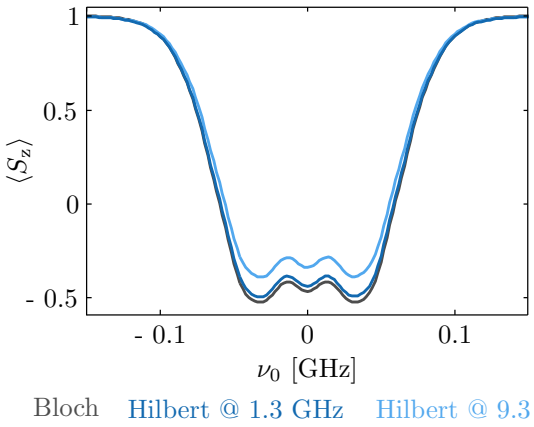


**Figure 1:** Effect of frame selection on computational precision. a) The inversion efficiency of a 64 ns chirp pulse with a $\nu_1$ of 20 MHz and a bandwidth of 100 MHz and b) the trajectory of a single spin in the center of sweep. The laboratory frame simulations are depicted in light blue, LO frame in dark blue and solution of the Bloch equations in black.

ratory frame simulations (light blue) were conducted at 9.3 GHz with a time resolution of 24 ps, while the AWG-frame traces (dark blue) were calculated at 1.3 GHz with a time resolution of 83 ps. Comparison to an accurate solution of the Bloch equations (black), shows that large off-diagonal elements

in the Hamiltonian, as they occur for simulations at 9.3 GHz, as well as amplitude deviation from time discretization, cause a significant numerical error in the computation of matrix exponentials.
Since simulations are usually carried out in the LO frame, resonance and excitation frequencies in SPIDYAN have to be declared relative to the LO frequency. In case a laboratory frame simulation is requested, the user additionally has to specify the frequency of the LO. All required variables are then converted automatically by SPIDYAN. If excitation frequencies fall outside the Nyquist range of the requested AWG setup, the program increases the sampling rate to a suitable value.

### 1.4 Down conversion

In spectroscopy with monochromatic pulses, signals are down converted with the excitation frequency in a mixer and detected at frequencies around zero. In SPIDYAN simulations, independent of the chosen simulation frame, signals oscillate at high frequencies around the offset of pulse center frequency from the LO frequency and are best further down converted digitally. This can either be done automatically during the simulation or, in cases where this is not possible (e.g. simulation of a spin echo), after acquisition with a SPIDYAN function. An example for automatic case can be found in Section 5.4 and for down conversion after signal acquisition in the code for simulation of the primary echo in Section 5.5.

## 2 User-Created Input

This section describes all variables that appear in SPIDYAN, may they be created by the user or generated by the program in the course processing of the input structures. The output of the simulation is described in Section 3. All variables are written as `.variable_name` and can be searched for in that way, which avoids displaying name duplicates that are no variables.

### 2.1 The Spin System

The structure `system`[1] provides SPIDYAN with the spin system, see Table 1. Only two fields are required for SPIDYAN to work.

`.sqn`: A vector which contains the spin quantum numbers of all the spins in the system. The indexing as it is used here, must be used for building the Hamiltonian with an interaction table.

[1] Of course it is possible to name the structure differently, as long as field names are the same.

**Table 1:** Required, optional and generated field of structure `system`. If a field appears under optional and generated it means, that, if omitted a default value is assumed.

| Required fields: | Optional fields: | Generated fields: |
|---|---|---|
| - `.sqn` | - `.T1` | - `.ham` |
| - `.interactions` | - `.T2` | - `.H` |
| | - `.init_state` | - `.spins` |
| | - `.eq` | - `.eq` |
| | - `.dc_freq` | - `.gamma` |
| | | - `.highest_freq` |

`.interactions`: A table which allows to specify single and two-spin interactions. Examples can be found in the example files and in Fig. 2. Each line corresponds to an interaction. Fig. 2a creates only one Hamiltonian

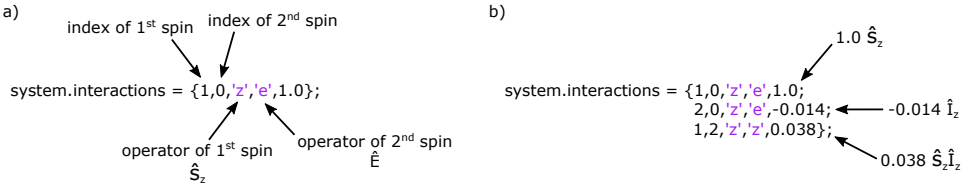


**Figure 2:** Layout of interaction tables. A Zeeman Hamiltonian term a) is created by only providing the index of one spin in `system.sqn` and setting the element in the table, which corresponds to the operator to `'z'`. The string `'e'` equals the identity matrix. For two spin interactions, the indices of both involved spins, and their corresponding operator types need to be provided, see third line b).

term for the first spin ($S$)

$$\hat{\mathcal{H}}_\mathrm{Z} = \Omega_S \hat{S}_\mathrm{z} \tag{1}$$

while the input in Fig. 2b will create a Hamiltonian of the form

$$\hat{\mathcal{H}}_\mathrm{Z} = \Omega_S \hat{S}_\mathrm{z} + \Omega_I \hat{I}_\mathrm{z} + A \hat{S}_\mathrm{z} \hat{I}_\mathrm{z} \tag{2}$$

where $I$ is the second spin and $\Omega_S = 1\,\mathrm{GHz}$, $\Omega_I = -14\,\mathrm{MHz}$ and $A = 38\,\mathrm{MHz}$. The first position always corresponds to the index of a spin in `.sqn`. For a single-spin interaction, the second position should be zero. For a two-spin term, the corresponding index of the second spin has to be provided. The third element denotes the spin operator of the first spin and the fourth the spin operator of the second spin. In case of a single-spin interaction, the fourth position is ignored. The fifth element is the magnitude of the interaction in GHz. The supported spin operators and their corresponding symbols in MATLAB are displayed in Table 2. After processing of the system, the Hamiltonian in matrix form is written to `.ham`. The

**Table 2:** Spin operators and their corresponding symbols in SPIDYAN.

| Operator | $\hat{S}_\mathrm{x}$ | $\hat{S}_\mathrm{y}$ | $\hat{S}_\mathrm{z}$ | $\hat{S}^+$ | $\hat{S}^-$ | $\hat{E}$ |
|---|---|---|---|---|---|---|
| String | `'x'` | `'y'` | `'z'` | `'p'` | `'m'` | `'e'` |

program also stores each individual interaction in matrix form in the cell array `.H`. In cases where the interaction table method is not sufficient, it is also possible to provide a Hamiltonian in matrix form through the field `.ham`.

`.T1`,`.T2`: The longitudinal and transverse relaxation times are only required for simulations with relaxation switched on. The input can either be a double or a matrix. In the former case, the same relaxation time is assigned to all transitions, which is convenient but not necessarily correct. The 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 $(2S_1+1)(2S_2+1)\ldots$ for multi-spin systems respectively. A detailed instruction on how to set up the input correctly can be found in Section 2.1.1. SPIDYAN only considers auto relaxation, other relaxation types such as cross correlation are not included.

`.init_state` The initial state can either be given as matrix or string. In the case of a string the same identifiers for spin operators are allowed (Table 2). In addition it is possible to use + and −. Let us assume a three-spin system. Some possible initial states are described in Table 3. In general it is possible to omit a certain spin by using `'e'` or truncating the string (see $5^{\mathrm{th}}$ and $6^{\mathrm{th}}$ example in Table 3). In such a case the transitions of the spin are considered unpolarized and the corresponding density matrix has only zeros ($\langle \hat{S}_i \rangle = 0$), which for example is used in nuclear spin simulations (see Section 5.5). If no initial state is given, the high-temperature approximation is assumed and all spin (regardless of their type) are taken to be quantized along the $z$-axis.

`.eq`: Defines the equilibrium state of the system, only required for simulations with relaxation. If omitted, the equilibrium state is assumed to be the same as the initial state. It is possible to provide a matrix or a string. The string has the same structure as for the initial state (Table 3). Additionally it is possible to assign `.eq` the string `'init'`, which then uses the initial state.

`.dc_freq`: The frequency for down conversion of the signal in GHz. If not defined, SPIDYAN tries to guess it from the interaction table. Only used if down conversion is switched on. See below for discussion of down conversion.

**Table 3:** Some possible initial states for a three spin system.

| | |
|---|---|
| `'zzz'` | All three spins are quantized along the $z$-axis. |
| `'zz-z'` | Spin 1 and 2 are quantized along the $z$-axis, spin 3 along $-z$. |
| `'z-z-z'` | Spin 1 is quantized along the $z$-axis, spin 2 and 3 along $-z$. |
| `'zzx'` | Spin 3 is in a coherent state. |
| `'zze'` | State of spin 3 is not defined, |
| `'zz'` | State of spin 3 is not defined. |
| `'z'` | States of spin 2 and 3 are not defined. |
| `'zez'` | State of spin 2 is not defined. |

`.ham`: Hamiltonian of the system in matrix form in angular frequency. Calculated from the interaction table. Instead of having the Hamiltonian built from the interaction table it is also possible to provide a custom one by assigning a matrix to `.ham` before call of `setup`. In this case the calculation is skipped and the Hamiltonian is used.

`.H`: A cell array, which contains each interaction defined in `.interactions`. Can be used to check individual Hamiltonian terms.

`.spins`: Number of spins in the spin system, calculated from `.sqn`.

`.gamma`: Relaxation super operator $\hat{\hat{\Gamma}}$ in Liouville space, calculated from `.T1` and `.T2`.

`.highest_freq`: The highest frequency from the interaction table. If `options.dc_freq` is not provided in options and on-the-fly down conversion is switched on, this is used for down conversion.

### 2.1.1 Assigning Relaxation Times Correctly

SPIDYAN offers two methods for setting relaxation times[2]. 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

[2]SPIDYAN only treats auto-relaxation, other relaxation types such as cross relaxation, are not implemented.

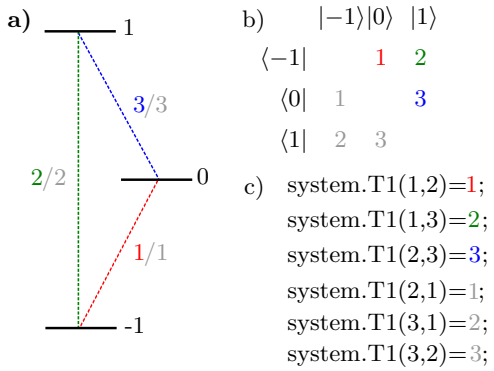


**Figure 3:** a) Transitions and b) the corresponding positions in the matrix for the relaxation times for a three level system with S=1 with c) A possible MATLAB code for this example, showing only $T_1$. For cases when relaxation in the up and down direction differ, you can assign non-zero values to the lower triangle of the matrix (grey). The lower triangle represents the down direction.

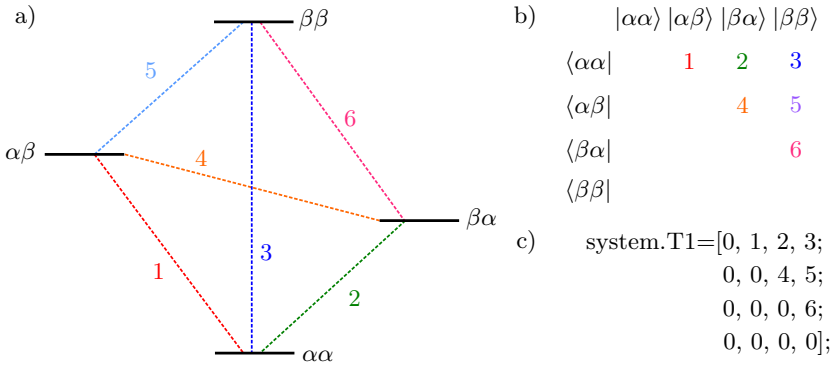


**Figure 4:** a) Transitions and b) their corresponding positions in the matrix for the relaxation times for a four level system with $S = 1/2$ and $I = 1/2$ and c) the MATLAB code for a matrix.

needs to be in matrix form, with dimensions $(2S + 1)$ for a single spin and $(2S_1 + 1)(2S_2 + 1)\ldots$ for multiple spins respectively. Fig. 3 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 Fig. 4), and after all combinations, the spin quantum number of the first spin is increased. Only the upper triangle of the matrix needs to be defined if the high-temperature approximation is valid. The lower triangle will be filled by assuming symmetry.

If you want to use different relaxation times for up and down pathways, you

can assign values to the lower triangle. The upper triangle then corresponds to the up direction (raising of spin state), the lower to the down direction (lowering of spin state). SPIDYAN will set the relaxation time for every non-defined (value of zero) transition to `.default_T1` or `.default_T2`. This has the advantage, that when you are interested in only one (or a a few) specific pathways, you only have to set these and the rest will be automatically assigned by SPIDYAN. The elements on the diagonal of the matrix do not correspond to relaxation pathways and therefore do not need to be considered.

## 2.2 The pulse sequence

The structure sequence is used to define the events of the pulse experiment. It contains fields for the event[3] 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 4.

**Table 4:** Required, optional and generated field of structure `sequence`. If a field appears under optional and generated it means that, if omitted, a default value is assumed.

Required fields:

- `.tp`
- `.frq`

At least one of:

- `.nu1`
- `.beta`
- `.Q`

Optional fields:

- `.t_rise`
- `.detection`
- `.pcycle`
- `.type`
- `.phase`
- `.excite`
- `.stepwise_evolution`
- `.HSbeta`
- `.HSorder`
- `.HSorder2`
- `.WURSTN`

Since setting up a pulse sequence and defining pulse types can be a little confusing at first, a good starting point are the accompanying files `pulses.m` and `echo_template.m` and their discussion in Section 5.

`.tp`: A vector, which contains the lengths of all events in ns. The ordering in this vector is used for assignment of all other (pulse) parameters. For example the third element in `.nu1` corresponds to the third element in `.tp`.

`.frq`: Cell array with microwave freqency (range) of the pulses in GHz

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

in the LO frame (for more details see discussion of LO frame below). Each element in the cell array has to be a vector. The first element of the vector corresponds to the initial frequency, the second to the final frequency. In case `.type` is not provided, SPIDYAN then uses these to make a guess of the pulse type. If both elements are identical a rectangular pulse is assumed and `.type` is ignored. If initial and final frequency are not the same, SPIDYAN then checks if `.type` is provided. In the case that no pulse type is provided, a linear chirp is assumed.
If the length of the *cell array* `.frq` is one, but several pulses are defined in `.nu1`, SPIDYAN takes this one vector to compute all pulses. Find examples in Section 5.

`.nu1`: Vector with pulse amplitudes in MHz. Must be zero for a free evolution event.

For some pulse types it is possible to provide a flip angle `.beta` (rectangular and chirp pulse) or an adiabaticity factor `.Q` (chirp pulse only) instead of a pulse amplitude. SPIDYAN then automatically calculates the correct amplitude. In order for this to work, the corresponding element in `.nu1` must be zero, otherwise `.beta` and `.Q` are ignored. For a free evolution event the corresponding elements in all three vectors (`.nu1`, `.beta` and `.Q`) must be zero.

`.beta`: Vector with flip angles for the pulses in radians. Works only for rectangular and chirp pulses.

`.Q`: Critical adiabaticity. Only available for chirp pulses.

`.detection`: Boolean vector. Expectation values are determined only for events which are 1.

`.stepwise_evolution`: Boolean vector. For free evolution elements which are 1, propagation is performed on the grid of the virtual AWG. If not provided, SPIDYAN assumes it to be the same as `.detection`. Can be used to switch to direct evolution (in one step) during detected free evolution events, to speed up computation.

`.pcycle`: This array controls phase cycling. The individual cell elements herein 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 $\pi$. The two resulting spin density matrices (expectation value traces) are processed according to the second 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.

Each of the following properties of the pulse can be configured in two ways:

- The property is a vector or cell array with length larger than one:
  The value at position $i$ is assigned to the event at position $i$ in `.tp`.
- The property is a vector or cell array with length one:
  The value is taken for all pulses. Be careful when you have several pulses, but want to assign a property only to the first pulse. In such a case the length `.property` should at least be two (second element can be zero). Otherwise the property is added to all pulses!

`.t_rise`: Rise times of the pulses in ns, only for rectangular and chirp pulses. Providing a non-zero rise time to pulses provides pulses with smooth edges. In SPIDYAN this is realized by weighting the beginning and the end of the pulse with a quarter sin wave of length `.t_rise`.

`.type`: Cell array, each element has to be a string. Defines the type of pulse, allowed strings are:

- `'rectangular'`
- `'linear'`, `'chirp'`
- `'HS'`
- `'WURST'`

Can be defined for all pulses (length of cell array is one), or for each pulse separately.

`.phase`: Vector with phase of the pulse in radians. This phase is added to all steps of the phase cycle

`.excite`: A cell array that can be used for manipulation of the excitation operator (exclude certain spins from excitation). Each element can either be a boolean vector, where the index corresponds to `system.sqn` or a matrix. Elements (spins) which are zero or not defined in the boolean vector are not affected by pulses. See Section 5.5 for an example where only one spin is to be excited. If a matrix is given instead, it is interpreted as the excitation

operator.

.HSbeta,.HSorder,.HSorder2: Vectors with parameters for definition of hyperbolic secant (HS) pulses.

.WURSTN: Vector with parameter for definition of WURST pulses.

## 2.3 Options

With options simulation parameters are adjusted, which are not covered by system and sequence. It is possible to declare detection operators, tweak SPIDYAN, include a resonator and explore the effect of complex excitation. A list of available fields can be found in Table 5. Most fields can be omitted and default values are assumed in that case.

**Table 5:** Available fields of the structure options.

| | |
|---|---|
| - .relaxation | - .awg.s_rate |
| - .det_op | - .awg.vert_res |
| - .labframe | - .resonator.active |
| - .LO | - .resonator.Ql |
| - .down_conversion | - .resonator.f0 |
| - .no_dc | - .resonator.comp_bw |
| - .complex_excitation | - .store_density_matrix |

.relaxation: Boolean, switches relaxation on/off. Default is to off.

.det_op: Cell array with detection operators. Can contain strings or matrices. For information on how to set up the string for detection operators, see Section 2.3.1.

.labframe: Boolean, selects the frame for the simulation: lab frame (1) or LO frame (0) (see examples below). Usually it is sufficient to calculate using the LO frame. Default is to 0.

.LO: Frequency of the LO element in GHz, corresponds to the difference of the simulation frame to the lab frame.

.down_conversion: Boolean, skip down conversion (0, default) or down convert obtained signal on the fly (1). The former returns the signal in the simulation frame. The latter adds the signal after down conversion to the output. The frequency for down conversion can be provided through system.dc_freq, if not SPIDYAN tries go guess a frequency from the in-

teraction table, by taking the highest value for an interaction that it can find.

`.no_dc`: Boolean vector which can be used to exclude (1) certain signals from being down converted[4]. The indexing corresponds to the ordering in `.det_op`. See the file `echo_template.m` for an example.

`.complex_excitation`: Boolean, SPIDYAN uses a complex excitation operator if set to 1, default is to 0. Complex excitation reduces computation efficiency drastically.

`.awg.s_rate`: Sampling rate of the virtual AWG in GSa/s. This is related to the size of the time steps. Default is to 12 GSa/s. In cases where the provided (or default) value does not fulfill the Nyquist criterion for the pulse, the sampling rate is automatically increased to a suitable value and a warning is returned to the command prompt.

`.awg.vert_res`: Vertical resolution of the virtual AWG in bit. The default value is 10 bit.

`.resonator.active`: Boolean. If a resonator is defined (`.resonator.Ql` and `.resonator.f0`), the resonator transfer function is only applied to the pulse if `.resonator.active` is set to 1. If any of the resonator parameters is missing the resonator is ignored. Default is 0.

`.resonator.Ql`: Quality factor of the loaded resonator.

`.resonator.f0`: Resonance frequency of the resonator in GHz. Important: This frequency has to be given in the lab frame. The resonator profile is calculated in the lab frame and afterwards down converted to the simulation frame.

`.resonator.comp_bw`: Boolean. If true, the pulses are bandwidth compensated according to [2], to allow for a flat excitation profile over the entire bandwidth of the pulse. Warning for the current version: If a critical adiabaticity value is used for pulse definition, the critical adiabaticity of the compensated pulse will be smaller than the requested one. Default is to zero.

`.store_density_matrix`: Boolean. Stores density matrices at each time

[4]This can be useful if a system contains an electron and a nuclear spin. At the moment SPIDYAN cannot differentiate between electron and nuclear spins and will, by default, apply the same down conversion and filtering to both. In the case when coherence on the nuclei is detected, down conversion of the signal will cause erroneous output. The problem arises, since the nuclear Zeeman frequency term in the Hamiltonian is defined in the laboratory frame, not the LO frame.

step for events that are detected. This can be very memory consuming. Defaults to 0.

### 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 expectation value is formed through

$$\langle S_i \rangle = \mathrm{Tr}\left(\hat{\sigma}_0 \hat{S}_i\right) \tag{3}$$

and normalized by $S_i^{\mathrm{norm}} = \mathrm{Tr}\left(\hat{S}_i \hat{S}_i^*\right)$ .
Detection operators are written to the cell array `.det_op` in the structure `options`. There is no limitation on the number of detection operators and it is possible to use a combination of strings and matrices. On top of the symbols displayed in Table 2, numbers and +/− are allowed. This gives the following set simple detection operators for a single spin system:

```
options.det_op={'x','y','z','p','m','-z',...}
```

Populations are detected by appending numbers to the operator symbol `'i'`. The number hereby corresponds to the row/column in the density matrix of the spin system.
Transition selective operators are created by adding a second number, seperated by a comma `','`, and are not limited to the `'z'`-operator. A few examples for a spin $S = 1$ system are given in Table 6. Treatment of a multi spin

**Table 6:** A few spin operators as string and in their corresponding matrix form for an $S = 1$ system.

| `'z'` | `'-z'` | `'x'` | `'p'` | `'i1'` |
|---|---|---|---|---|
| $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}$ | $\begin{bmatrix} -1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ | $\begin{bmatrix} 0 & 0.7 & 0 \\ 0.7 & 0 & 0.7 \\ 0 & 0.7 & 0 \end{bmatrix}$ | $\begin{bmatrix} 0 & 1.4 & 0 \\ 0 & 0 & 1.4 \\ 0 & 0 & 0 \end{bmatrix}$ | $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ |
| **`'z1,2'`** | **`'z2,1'`** | **`'x1,2'`** | **`'p1,3'`** | |
| $\begin{bmatrix} 0.5 & 0 & 0 \\ 0 & -0.5 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ | $\begin{bmatrix} -0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ | $\begin{bmatrix} 0 & 0.5 & 0 \\ 0.5 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ | $\begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ | |

system follows the same general rules, with the addition, that for product operators several letters need to be combined. A few product operators of a three-spin system are shown in Table 7. The ordering is once again taken

**Table 7:** Examples for multi-spin operators in a three-spin system.

| `'zzz'` | `'-zzz'` | `'zxz'` | `'zze'` | `'zz'` |
|---|---|---|---|---|
| $\hat{S}_{1,z}\hat{S}_{2,z}\hat{S}_{3,z}$ | $-\hat{S}_{1,z}\hat{S}_{2,z}\hat{S}_{3,z}$ | $\hat{S}_{1,z}\hat{S}_{2,x}\hat{S}_{3,z}$ | $\hat{S}_{1,z}\hat{S}_{2,z}$ | $\hat{S}_{1,z}\hat{S}_{2,z}$ |

from `system.sqn`. Spins at the end of the list can be omitted (see last example in Table 7) and are then treated as `'e'`. If coherence of the first spin is to be detected, either of the following two detection operators can be used:

```
options.det_op = {'pee'}
options.det_op = {'p'}
```

By appending digits to the multi-spin operators, transition selective operators are created. If you are uncertain about whether you created the desired detection operator, you can always have a look at the detection matrix of it by typing

```
options.detect{i}
```

into the command prompt, *after* `options` has been processed with `setup`.

# 3 Output

The input structures `system`, `sequence` and `options` are processed with the two functions `setup` and `triple`. This adds missing fields, as well as computes Hamiltonian and waveforms which are needed for propagation. The function `homerun` can then be called with the updated structures, returning (among others) the following two variables: `state` and `signal`. The variable `state` is the state of the system after the pulse experiment in form of a spin density matrix and the structure `signal` contains the fields `.sf`, `.dc`, `.t` and `.signals`. Only events for which detection was switched on are available. If detection was switched off entirely, `signal` is empty. Also returned are an updated version of the structure with the pulse sequence, options and a structure `tables` and a cell array `sigmas`.

`state`: Spin density matrix of the final state of the spin system after propagation.

`.sf`: An $n \times k$ dimensional matrix where $n$ is the number of detection operators provided in `options` and $k$ the length of the signal. It contains the expectation values at each time step as detected in the simulation frame.

`.dc`: Same as `.sf` but for the signal after down conversion. Available only if `.down_conversion` was switched on.

`.t`: Time axis, can be used for plotting of `.sf` and `.dc`.

`.signals.sf`: Cell array, containing the expectation values of each individually detected event in the simulation frame.

`.signals.dc`: Cell array, only available if down conversion is switched on. Stores down converted expectation values of each individually detected event.

`.signals.t`: Cell array with time axes for `.signals.sf` and `.signals.dc`.

`tables`: Structure which contains propagators. Can be used to make consecutive calculation of the identical pulse sequence with the same static Hamiltonian but changed interpulse delays run faster.

`sigmas`: A cell array with density matrices for each detected time point. Only available if `options.store_density_matrix` was set to true. Can be plotted with the accompanying function `barscroll` which allows to scroll through a visualization of the state of the spin system.

# 4 The SPIDYAN Environment

A simplified environment scheme of SPIDYAN is displayed in Fig. 5. After setup of the three structures `system`, `options` and `sequence`, these are processed with the two functions `setup` and `triple`. While `setup` adds spin density matrices, the Hamiltonian in matrix form, detection operators and sets up the relaxation superoperator if requested, `triple` checks the provided pulse sequence for consistency and creates the pulses.
Though this would have been possible, these two routines were not combined with `homerun` into a single function. This gives more flexibility and allows deeper insight into the simulation. Furthermore, certain parameters, such as Hamiltonians, pulse strengths or inter-pulse delays can still be changed after building the pulses or system. This comes in handy when simulations are run in loops with only one varied parameter.
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 as well as the time traces and an updated version of 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

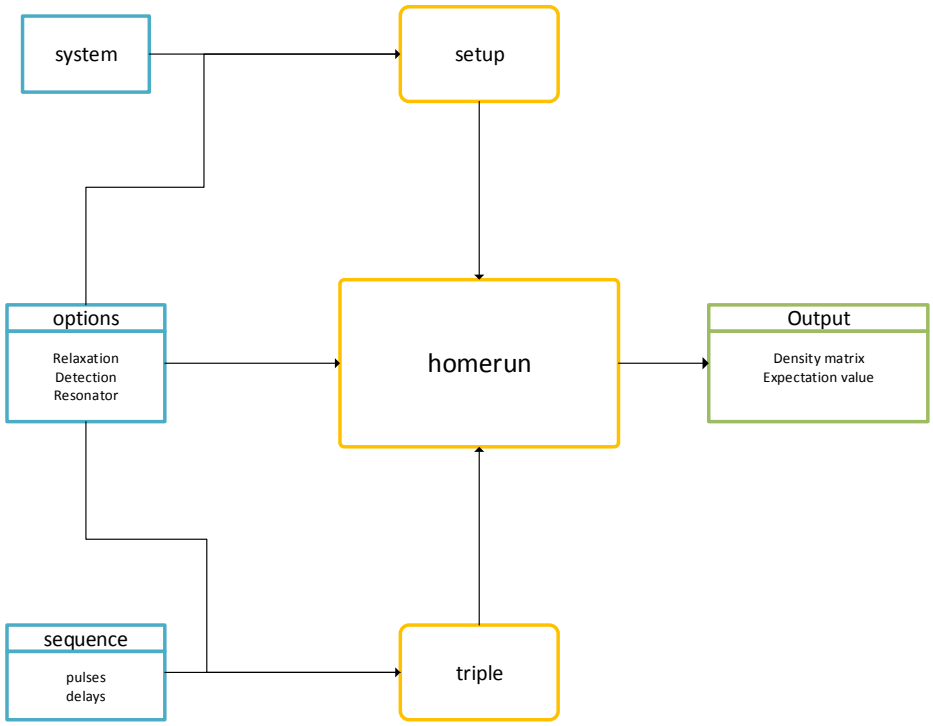


**Figure 5:** The SPIDYAN environment.

structures `system`, `options` and `sequence` are intended, these structures should be cleared and newly initialized.

# 5 Examples

The following sections concern themselves with example files, to give a few starting points for working with SPIDYAN. Not all of the online provided example files are discussed in depth in this documentation.
The scripts can be downloaded together with the program. Most of the MATLAB code in this file can also be copy-pasted into the command line or editor of MATLAB, though it is possible that some of the special characters are not recognized correctly. MATLAB will then report an error. In such a case, you will to have to correct the errors yourself or just use the original scripts.

## 5.1 A Simple System

The script `a_simple_system.m` is a good way to get familiar with SPIDYAN. It treats a single electron spin and visualizes what happens during a pulse (no complicated pulse sequences yet). If the script is executed, five figures

are created in total. The first graph (Fig. 6a) shows $\langle -\hat{S}_z \rangle$ and $\langle \hat{S}^+ \rangle$ for a rectangular pulse and the second (Fig. 6b) for a chirp pulse with a critical adiabaticity of $Q_{\text{crit}} = 5$. The third plot (Fig. 6c) is a three-dimensional representation of the spin trajectory during the chirp pulse. The chirp pulse with an effective flip-angle of $\beta = \pi/2$ is depicted in the fourth figure (Fig. 6d). The scroll bar on the right of the last figure (not shown here) allows to follow the changes of the density matrix during the chirp pulse with $Q_{\text{crit}} = 5$.

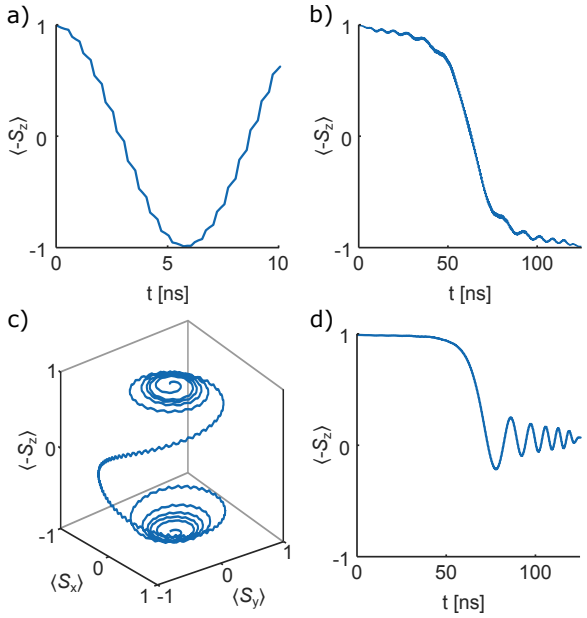


**Figure 6:** Four of the five figures created by the script `a_simple_system`. In a) a spin during a resonant rectangular pulse, in the b) and c) performance of chirp pulse with $Q_{\text{crit}} = 5$ and in d) a chirp pulse with a flip angle of $\beta = \pi/2$ are depicted. a), b) and d) only show $\langle -\hat{S}_z \rangle$.

The first block of the script defines the spin system:

```
system.interactions={1,0,'z','e',1}; % Zeeman-term in the LO-frame
system.sqn=0.5;          % spin quantum number
system.T1=2000;          % relaxation times in ns
system.T2=1000;
system.init_state='-z'; % initial state
```

This creates a single electron with $S = 1/2$ and the system Hamiltonian consists solely of Zeeman-Hamiltonian term which is given in the LO frame (see Section 1.3). Although relaxation is switched off in this case, relaxation times are defined (You can try switching relaxation on by setting `options.relaxation = 1` and see what happens). The last line defines the initial state. Since we are treating an electron spin, the initial state has to be along the $-z$ direction. Without relaxation it is not necessary to

define an equilibrium state (`system.eq`). This can be added for relaxation, or, if relaxation is switched on without a definition of the equilibrium state, SPIDYAN assumes the initial state to also be the equilibrium state.
Next comes the definition of the first pulse. Since we are simulating the effect of three different pulses on the same initial state, it is necessary to define three pulse structures and run each of them seperately. For the rectangular pulse we can write:

```
rectangular.tp=10;        % length of the pulse in ns
rectangular.nu1=[80];     % flip angle for pulse
rectangular.frq={[1.0]}; % this creates a rectangular pulse
rectangular.t_rise=[0];  % the rise time of the pulse
rectangular.detection=1; % detection  switched on
```

In fact all of the above variables are vectors. But since we only consider a single pulse (therefore one event) it is safe to write the variables not in vector form (brackets). The first field (`.tp`) tells SPIDYAN about the number of events (length of the vector) and their durations. We are creating a 10 ns pulse with an pulse amplitude of 80 MHz (`.nu1`). In this case the event is recognized as a pulse by SPIDYAN, because `.nu1` contains a non-zero value. Next comes the microwave frequency in the LO-frame in GHz, followed by the rise time of the pulse in ns. The last field tells SPIDYAN to switch detection on during the simulation.
Next we have to define a few more options:

```
options.relaxation=0;        % with relaxation (1) or not (0)
options.down_conversion=1;   % downconversion (1) or not (0)
options.det_op={'-z','p'};   % cell-array with detection operators
options.LO=8;                % frequency of the local oscillator
options.labframe=0;          % simulate in labframe (1) or not (0)
options.store_density_matrix=1; % stores density matrices
```

A complete list of all available options with their default values can be found in Section 2.3. The first line tells SPIDYAN to simulate without taking relaxation into account and the second activates down conversion on the fly. This effects signals which have off-diagonal elements in their detection operators (see Sections 1.4 and 5.4 for details). Probably the most important option is the setting of the detection operator. In this case we are detecting $\langle -\hat{S}_z \rangle$ (`'-z'`) and $\langle \hat{S}^+ \rangle$ (`'p'`). More detection operators (Section 2.3.1) can be added by using commas as separators. The field `.LO` is the frequency of the local oscillator (Section 1.3). With `.store_density_matrix` it is possible to store the density matrix at each time step during detection periods. These can then be plotted (see below).
Before we can run the simulation the input now needs to be processed:

```
[rectangular, options] = triple(rectangular, options);
[system, state0, options]=setup(system,options);
```

The function `triple` builds the waveform from the input and adds an additional field `.pulses` to the structure `rectangular`. With `setup` missing

fields, such as the Hamiltonian `.ham`, equilibrium state `.eq` and relaxation superoperator `.gamma` (if relaxation is requested) are generated. The initial state of the spin system is returned as `state0`. Both functions also modify the structure `options` and add some default values.
Now it is finally time to propagate the system:

```
[state, signal_rectangular]=homerun(state0, system, ...
    rectangular, options, []);
```

This returns the state of the system `state` after the pulse sequence and the time traces `signal_rectangular`. We can now have a look at the final state with

```
disp(state)
```

which prints the density matrix to the command line and can plot the time traces (after down conversion), Fig. 6a):

```
figure(1),clf
title('Rectangular pulse')
hold on
box on
plot(signal_rectangular.t,real(signal_rectangular.dc))
xlabel('t [ns]')
ylabel('<S_q>')
legend(options.det_op)
```

If you are interested in what the time traces look like before down conversion go ahead and change the plot command to:

```
plot(signal_rectangular.t,real(signal_rectangular.sf))
```

This will display the signals in the simulation frame.

Let us now create a chirp pulse:

```
chirp.tp=120;     % length of the pulse in ns
chirp.nu1=0;      % pulse strength in MHz
chirp.Q=5;        % crit. adiabaticity, .nu1 and .beta must be zero
% chirp.beta=pi; % flip angle for pulse
chirp.frq={[0.8 1.2]};  % sweep range
chirp.t_rise=10; % rise time of the pulse
%chirp.detection=1; % detection is switched on
```

This creates a new structure `chirp`, which again contains only one event: A pulse with a length of 120 ns. Before we used the pulse amplitude to identify the event as pulse. In the case of chirp pulses we can also define the pulse through adiabaticity: `.Q = 5` requests a chirp pulse with a critical adiabaticy of 5, which corresponds to a flip angle of almost $\pi$. SPIDYAN then adapts the pulse amplitude for the provided pulse length and sweep width (`.frq`). Alternatively we could also define the flip angle with `.beta`. This would lead to the same result. Flip angles also work for rectangular pulses.

Important: If `.nu1`, `.beta` and `.Q` are given for the same event, only one can be used. The priorities in SPIDYAN are as follows: `.nu1` > `.beta` > `.Q`. Hence, if for one event the pulse amplitude and critical adiabaticity are defined, the program will ignore $Q_{\text{crit}}$. In the script this is avoided by setting `.nu1` to 0. Alternatively, we could just omit `.nu1`. As soon as `chirp.beta=pi;` is uncommented, the statement `chirp.Q=5` is ignored. In a multi-pulse sequence it is of course possible to define some pulses with $Q_{\text{crit}}$ and others through pulse amplitude or flip angle. In such a case the parameters must be vectors and all parameters for a given pulse with higher priority must be zero.
The sweep width is defined through `.frq`: Since we want a chirp pulse, we must now provide an initial and a final frequency. The first element of the vector is the starting frequency and the second element the end frequency. To ensure a flat excitation profile and avoid artifacts, the edges of the chirp pulse need to be flattened. This can be done by setting the rise time of the pulse to 10 ns. For a visualization of the chirp pulse in the time and frequency domain have a look at Section 5.2, which also covers other pulse types.
Since the system is still available from the first simulation (unless you cleared the workspace) we only need to process `chirp` this time:

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

You might notice, that we now store the output as `experiment`. We do that because we want to run yet another simulation with the same parameters, but a different adiabaticity factor. So instead of rewriting the whole pulse definition we will only have to change one field (see below).
Again, we can run the simulation and plot the outcome.

```
[state, signal_chirp, experiment, options, tables, ...
    sigmas]=homerun(state0, system, experiment, options);
```

This time we added a lot of new variables to the output, most of which, except for `sigmas`, we do not need. First let us have a look at a new plot:

```
figure(3),clf
title('3D plot of spin trajectory after downconversion')
Sx=real(signal_chirp.dc(2,:));
Sy=imag(signal_chirp.dc(2,:));
Sz=signal_chirp.dc(1,:);
plot3(Sx,Sy,Sz)
xlabel('<S_x>')
ylabel('<S_y>')
zlabel('<S_z>')
```

This will create a three-dimensional plot of the spin trajectory, Fig. 6c). For this we use that $\hat{S}^+ = \hat{S}_x + i\hat{S}_y$. The figure nicely depicts adiabatic inversion of the electron.
The next plot is even more interesting:

```
barscroll(sigmas{1},signal_chirp.t,4)
```

Now the density matrices at each time step are plotted and you can use the bar on the right to scroll through them. You can see how first the diagonal elements are being populated and that coherence is at its maximum, when the pulse is on resonance with the spin system.
The last thing that remains to do is change the critical adiabaticity from 5 to $2\ln(2)/\pi$, which corresponds to a flip angle of $\beta = \pi/2$, see [9], Fig. 6d).

```
chirp.Q=2*log(2)/pi;    % adiabaticity for a 90 degree flip
% chirp.beta = pi/2;    % equivalent statement

[experiment, options] = triple(chirp, options);

[state, signal_chirp]=homerun(state0, system, experiment, ...
    options, []);
```

## 5.2 Pulses

To define a pulse, the pulse length `tp`, microwave frequency `frq`, rise time `t_rise` and either pulse amplitude `nu1`, flip angle `beta` or critical adiabaticity (for chirp pulses) `Q` must be given. The following shows examples for a rectangular, a chirp, a HS [11] and a WURST (wideband uniform-rate smooth truncation) [12] pulse.
The combination of several pulses and delays into a pulse experiment is shown in Sections 5.5 and 5.6.
The code below illustrates how a rectangular pulse can be defined with SPIDYAN:

```
rectangular.tp = 10;      % pulse length in ns
rectangular.frq = 1;      % microwave freqency in GHz
rectangular.nu1 = 50;     % pulse amplitude in MHz, alternatively
% rectangular.beta = pi; % alternative to nu1
rectangular.t_rise=0;     % rise time in ns
rectangular.type={'rectangular'};   % type of pulse

options.plot_domain = 'both';       % option for plotting
rectangular = triple(rectangular,options);  % processing building

plot_pulses(rectangular,options)    % plotting
```

The above defined pulse has a pulse length of $t_\mathrm{p} = 10\,\mathrm{ns}$, a microwave frequency in the LO frame $\Omega = 1\,\mathrm{GHz}$, a pulse amplitude $\nu_1 = 50\,\mathrm{MHz}$ and no rise time. Additionally also the type of the pulse is defined (`'rectangular'`).
The field `options.plot_domain` tells the function `plot_pulses` to display the pulse in the time and the frequency domain (s. Fig. 7). Before plotting, the pulse parameters have to be processed with the function `triple`.
A linear chirp can be defined as follows:

```
chirp.tp = 100;
```

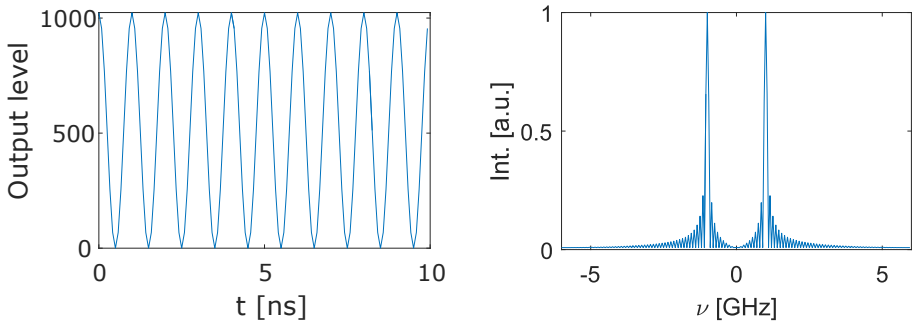


**Figure 7:** Output of `plot_pulses` for the rectangular pulse.

```
chirp.frq = [0.8 1.2];    % frequency, sweeps from 0.8 to 1.2 GHz
chirp.Q = 5;              % critical adiabaticity of pulse
chirp.t_rise = 20;
chirp.type = {'chirp'};

chirp = triple(chirp,options);
plot_pulses(chirp,options)
```

This will create a chirp with a pulse length of $t_\mathrm{p} = 100\,\mathrm{ns}$, an initial microwave frequency in the LO frame $\Omega_\mathrm{init} = 0.8\,\mathrm{GHz}$, an final microwave frequency in the LO frame $\Omega_\mathrm{final} = 1.2\,\mathrm{GHz}$, a critical adiabaticity $Q_\mathrm{crit} = 5$ and a rise time of $t_\mathrm{rise} = 20\,\mathrm{ns}$. Again, the type of the pulse is not required for the definition, as SPIDYAN assumes a linear chirp by default if it finds two different values in `.frq` but no `.type`. The pulse in the time and frequency domain is displayed in Fig. 8.

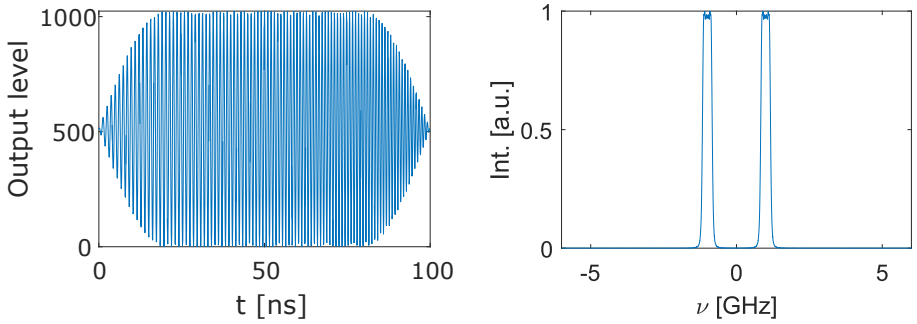


**Figure 8:** Output of `plot_pulses` for the chirp pulse.

A HS pulse [13] requires a few additional fields :

```
HS.tp = 100;
HS.nu1 = 10;
HS.HSbeta = 10;     % apodization parameter
HS.HSorder = 2;     % order of HS pulse
HS.frq = [0.8 1.2];
```

```
HS.type = {'HS'};

HS = triple(HS,options);
plot_pulses(HS,options)
```

The apodization parameter $\beta$ of the HS pulse is controlled through `.HSbeta` and the order is set with `.HSorder`. For the HS pulse it is mandatory to specify `.type` with `'HS'`. Otherwise the HS parameters are ignored and a linear chirp is assumed. The time trace of the pulse and the excitation profile are displayed in Fig. 9.

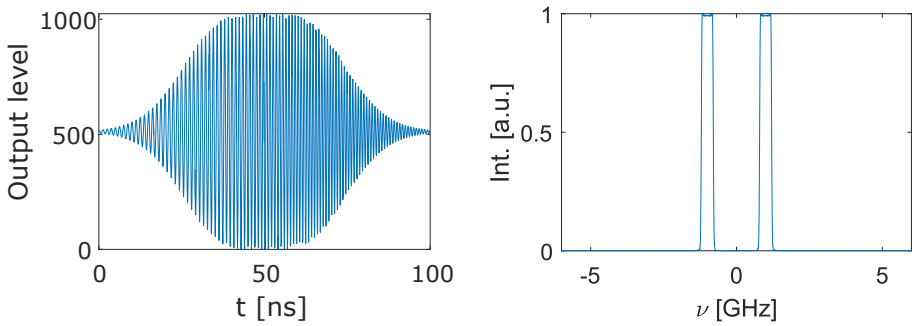


**Figure 9:** Output of `plot_pulses` for the HS pulse.

Like a HS pulse, the WURST pulse requires setting of additional parameters:

```
% WURST pulse
wurst.tp = 100;
wurst.nu1 = 30;
wurst.frq = [0.8 1.2];
wurst.type = {'WURST'};
wurst.WURSTN = 10;     % steepness of pulse, WURST-10

wurst = triple(wurst,options);
plot_pulses(wurst,options)
```

The field `WURSTN` controls the order of WURST pulse, e. g. `WURSTN=10` is a WURST-10 pulse. If the pulse type is not specified, the WURST parameters are ignored and a linear chirp is assumed. The above defined WURST-10 pulse is illustrated in Fig. 10.

## 5.3 Simulation Frames

In SPIDYAN simulations can be carried out in different rotating frames (Section 1.3). The following example propagates a $S = 1/2$ system during a rectangular pulse. Frequencies always have to be defined in the LO frame. We can take the resonance frequency of the spin as $\Omega_S = 1\,\text{GHz}$ and the pulse is on resonance with $\Omega = 1\,\text{GHz}$. The frequency difference to the lab frame is provided through `options.labframe`.

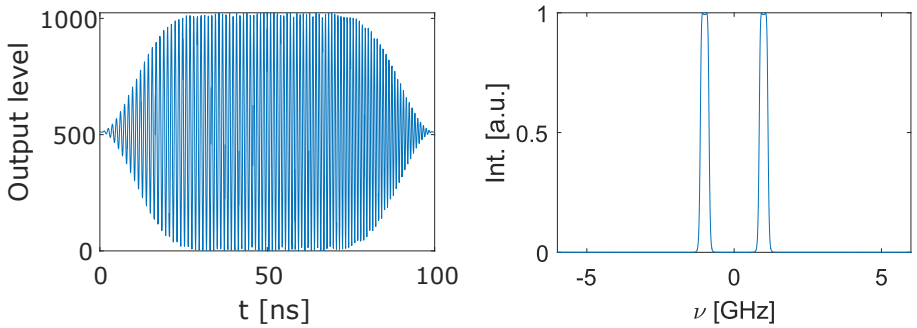


**Figure 10:** Output of `plot_pulses` for the WURST-10 pulse.

The code below runs two simulations. First the spin is propagated in the lab frame (Fig. 11a,b). Then the simulation is repeated in the LO frame (Fig. 11c).

```
% simulation frames

% definition of spin system
system.sqn = 1/2;         % spin quantum number
system.interactions = {1,0,'z','e',1.0};  % interaction table ...
    for the spin Hamiltonian
system.init_state = '-z'; % initial state of the spin system

% pulse sequence
rectangular.tp = 10;     % pulse length in ns
rectangular.frq = 1;     % microwave freqency in GHz
rectangular.beta = pi;   % alternative to nu1
rectangular.t_rise=0;    % rise time in ns
rectangular.type={'rectangular'};    % type of pulse
rectangular.detection = 1;

% options
options.relaxation = 0;    % relaxation is switched off
options.LO = 8;            % frequency of the LO in GHz
options.det_op = {'-z','y'};      %detection operators, -Sz and Sx

%% simulation in the lab frame %%%%%%%%%%%

options.labframe = 1;      % lab frame simulation is on
% options.awg.s_rate = 40;  % can be switched on to improve ...
    accuracy

[LFsystem,state0,options] = setup(system,options); % ...
    processing of system

[experiment,options] = triple(rectangular,options); % ...
    processing of pulse
```

```
[state1, LFsignal] =  homerun(state0, LFsystem, experiment, ...
    options, []); % propagation

figure(1)
plot(LFsignal.t,real(LFsignal.sf))
xlabel('t [ns]')
ylabel('<S_i>')

%% simulation in the LO frame %%%%%%%%%%

clear options
options.relaxation = 0;    % relaxation is switched off
options.LO = 8;            % frequency of the LO in GHz
options.det_op = {'-z','y'};      %detection operators, -Sz and Sx
options.labframe = 0;      % lab frame simulation is off

[LOsystem,state0,options] = setup(system,options); % ...
    processing of system

[experiment,options] = triple(rectangular,options); % ...
    processing of pulse

[state2, LOsignal] =  homerun(state0, LOsystem, experiment, ...
    options, []); % propagation

figure(2)
plot(LOsignal.t,real(LOsignal.sf))
xlabel('t [ns]')
ylabel('<S_i>')
```

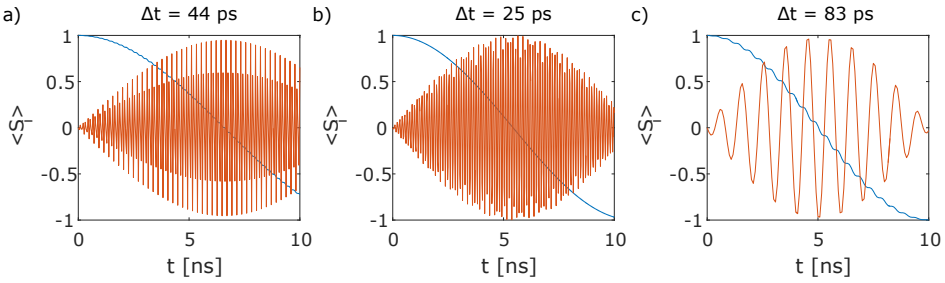


**Figure 11:** Three different simulations of a rectangular pulse with an effective flip angle of $\beta = \pi$. In a) the simulation is carried out in the lab frame at 9 GHz with a sampling rate of the virtual AWG of $s_{\text{rate}} = 22.5\,\text{GSa/s}$. Since this is rather low compared to the required Nyquist frequency $\nu_{\text{nyquist}} = 18\,\text{GHz}$, errors are introduced due to large off-diagonal elements in matrix exponentials and amplitude deviations due to time discretization errors. Subfigure b) shows how this can be improved with a higher sampling rate of 40 GSa/s. In simulations in the LO frame c), these deviations are not as pronounced.

In this example no pulse amplitude $\nu_1$ but a flip angle $\beta$, `.beta` was provided. SPIDYAN then calculates the correct pulse amplitude from the requested flip angle and the pulse length. This is possible for chirp and rectangular

pulses only.
Note that for the simulation in the lab frame, the accuracy strongly depends on the sampling rate. Fig. 11a displays the case where the sampling rate $s_{\text{rate}}$ is just a little larger than the required Nyquist frequency:

$\Omega_{\text{S}} = 1\,\text{GHz}$ in the LO frame
$\omega_{\text{S}} = 9\,\text{GHz}$ in the lab frame
$\nu_{\text{nyquist}} = 18\,\text{GHz}$
$s_{\text{rate}} = 22.5\,\text{GSa/s}$
$\Delta t = 44\,\text{ps}$

In Fig. 11b the sampling rate has been increased to $s_{\text{rate}} = 40\,\text{GSa/s}$ which corresponds to a time step size of $\Delta t = 25\,\text{ps}$.
The observed error is due to larger off-diagonal elements in the matrix exponential and amplitude deviations which arise from time discretization errors. At the present this error can be minimized by increasing the sampling rate of the simulation, as for instance in [9]. When changing from the lab frame to the LO frame, the relevant dynamics evolve at a smaller frequency and sampling beyond the Nyquist frequency therefore requires less computational effort. Alternative ways to minimize these deviations are being investigated. What can be observed as well, are oscillations of the $\langle \hat{S}_{\text{y}} \rangle$-signal with the frequency of the simulation frame, which can be removed through digital down conversion (see below).

### 5.4 Down Conversion

To down convert the obtained signal from the previous example, the following line has to be added to the structure `options`:

```
options.down_conversion = 1;
```

This adds an additional field `.dc` to the returned signal structure, which contains the down converted signal:

```
figure(3)
plot(LOsignal.t,real(LOsignal.dc))
xlabel('t [ns]')
ylabel('<S_i>')
```

Fig. 12 shows the signal before (in the simulation frame) and after down conversion. In general, SPIDYAN applies down conversion to all signals, which have off-diagonal elements in their corresponding detection operators. It is possible to set the down conversion frequency manually with `options.dc_freq`.

### 5.5 Spin Echo

Next let us have a look at a two-pulse sequence. In the previous examples we restricted ourselves to single spins and pulse sequences with one event.

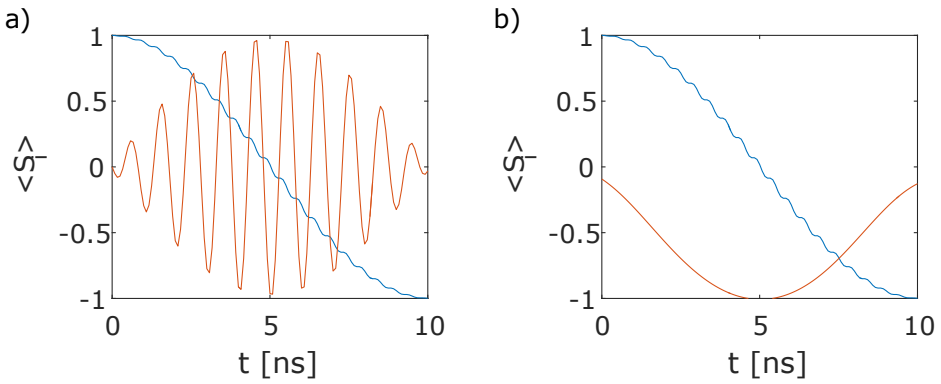


**Figure 12:** Output of `plot_pulses` for a WURST-10 pulse.

But in order to form a primary echo (refocusing of coherence), some kind of distribution of an Hamiltonian parameter is required. This can be realized by assuming a powder or a Gaussian distribution of resonance frequencies in a single crystal.
Since we did not want to limit ourselves (and potential users) to only use a certain set of preconfigured experiments, SPIDYAN does not provide you with any top-level functions to simulate spectra of such compounds right away. Instead it is up to the user to loop over SPIDYAN functions and combine time traces. Though more complex than calling a single function with a few parameters, keep in mind that SPIDYAN has been designed for a deeper insight into spin dynamics, and not fitting of spectra. In the future, scripts, which provide a simple way to calculate such spectra will become available for download on the program homepage.
Here we use, for the simulation of the primary echo, a simple Gaussian distribution of the resonance frequency (or $g$-values) of the electron spin in an electron-nucleus system. In order to obtain a time trace which contains the echo, our script has to do he following:

- Define Gaussian distribution of the resonance frequency
- Definition of spin system, pulse sequence and options
- Process the pulse sequence, but not the spin system
- Start a `for`-loop over the Gaussian distribution
  - Load a value for the resonance frequency
  - Replace the previously defined value in the interaction table
  - Process the spin system now
  - Propagate
  - Store time trace in simulation frame (no down conversion!)
  - Repeat
- Sum up individual time traces
- Down convert the accumulated traces

- Plot the echo

Let us have a look at the code:

```
% the spectral sampling
c=1.4;      % center frequency of Gaussian distribution, in GHz
g=0.01;     % width of the Gaussian distribution in GHz
nu0_s=1.3;  % starting value for sampling of Gaussian distribution
nu0_e=1.5;  % final value for sampling of Gaussian distribution
st=0.001;   % stepsize for sampling of Gaussian distribution
nu0_vec=nu0_s:st:nu0_e; % vector with resonance frequencies
p=exp(-((c-nu0_vec)/g).^2); % probabilities for each spin
p=p/trapz(p); % normalization
```

We define a Gaussian distribution of resonance frequencies with $\sigma = 35\,\mathrm{MHz}$ centered at 1.4 GHz, with a total of 201 spin packets between 1.3 and 1.5 GHz. Next, define the spin system and pulse sequence: In the interaction table we can use any value for the Zeeman term, since it will be replaced during the loop anyway. The rest of interaction table uses parameters from one possible orientation of $\gamma$-irradiated malonic acid ($\omega_I = -14\,\mathrm{MHz}$, $A = -45\,\mathrm{MHz}$, $B = 8\,\mathrm{MHz}$). We create two pulses with lengths 50 and 25 MHz sweeping from 1.3 to 1.5 GHz.

```
% the system
system.sqn=[0.5 0.5];       % spin quantum numbers
system.interactions={1,0, 'z','e',1.4;     % Zeeman term of ...
    electron, value is replaced in loop
                     2,0,'z','e',-0.014;   % Zeeman of nucleus
                     1,2, 'z','z',-0.045;  % secular part of ...
                         hyperfine interaction , A*SzIz
                     1,2, 'z','x',0.008};  % pseudo secular ...
                         part of hyperfine interaction , B*SzIz
system.init_state='-ze'; % only the electron spin is ...
    quantizised along z, nucleus is not defined, same ...
    statement would be '-z'
system.eq = 'init';  % same as initial state

% the options
options.relaxation=0;       % no relaxation
options.down_conversion=0;  % downconversion of signal off
options.det_op={'pe'};      % detection operator

% the sequence
sequence.tp=[50,300,25,600];  % vector with event lengths in ns
sequence.beta(1)=pi/2;        % flip angle for first event
sequence.beta(3)=pi;          % flip angle for third event
sequence.frq={[1.3 1.5]};     % initial and final frequency
sequence.detection=ones(1,length(sequence.tp)); % detects all ...
    events
sequence.t_rise=10;           % rise time of chirp pulses
sequence.excite={[1 0]};      % only excite the first spin

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

A few things to point out:

- Since both spins have the same spin quantum number, the discrimination between electron and nucleus is made in the first two entries of the interaction table, where we decide to make the first spin the electron and the second the nucleus.
- `system.init_state='-ze'`: In the initial state only the electron is quantized along the $z$-axis of the system. The nucleus is not defined (`'e'`). An equivalent statement would be `.init_state='-z'`.
- `options.det_op={'pe'}`: This is the detection operator we are using. In the experiment we can only detect electron coherence, hence the first symbol has to be `'p'`. All other spins should be ignored. Therefore we can add the `'e'`, which means that the nucleus does not contribute to the expectation value. Another possible way to write this detection operator would be: `.det_op={'p'}`. Omitted spins at the end of the list are automatically set to `'e'`.
- `sequence.detection=ones(1,length(sequence.tp))`: We want to observe electron coherence through the entire experiment. Therefore detection has to be switched on for each event. This is done by creating a vector with 4 ones in it.

Before running the loop we can initialize a cell array in which we will store the time trace of each spin packet This allows us to use the much faster `parfor` instead of the common `for`-loop. For parallel processing, you need the Parallel Computing Toolbox to be installed. If you have no access to it, just change the `parfor` to `for`. It will take longer, but give the same results.

```
signalc=cell(1,length(nu0_vec)); % creates an empty cell to ...
    store results

parfor k=1:length(nu0_vec)

    systeml=system;                          % copy system ...
        into loop
    systeml.interactions{1,end}= nu0_vec(k); % add loop ...
        resonance frequency

    [systeml, state, optionsl]=setup(systeml,options); % build ...
        system

    [state, signal, ...
        experimentl]=homerun(state,systeml,experimentl, ...
        optionsl,[]);  % propagate

    signalc{k}=signal.sf;   % keep signals from simulation frame
end
```

Due to the way parallel computing is implemented, it is not possible to change variables, that are used in other instances of the loop, during the loop. Each loop therefore needs its own copy of the spin system, called `systeml` here. After it has been cloned to each worker, we can now freely assign a resonance frequency from the Gaussian distribution to `systeml` and process and propagate it. The detected signal is then stored in `signalc`. Next we accumulate all signals. We have to recreate the time axis since all unsaved variables from the `parfor`-loop are lost.

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

for k=1:length(nu0_vec)
    signal=signal+signalc{k}*p(k);   % combines all signals
end

t=linspace(0,sum(experiment.tp),length(signal)); % creates ...
    time axis
```

We can now plot the result and observe high frequency oscillations on our trace.

```
figure(1); clf; plot(t,real(signal))
title('Signal in simulation frame')
xlabel('time [ns]')
ylabel('<S_{1p}>')
```

These can be removed by down conversion with the function `strike`. It is important that the signal is down converted after accumulation. If the signals are down converted on the fly and combined, refocusing can not be observed.

```
signaldc=strike(signal, t, c, options);   % down conversion of ...
    merged signals
signaldc=signaldc/max(max(signaldc));

figure(2); clf;
title('Signal after down conversion')
hold on
plot(t,real(signaldc))
xlabel('time [ns]')
ylabel('<S_{1p}>')
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--')
```

After plotting of the down converted signal we can see how coherence is created during the first pulse, dephases and is refocused with the $\pi$-pulse (Fig. 13a)).

Now let us reduce the number of spin packets (less spin packets mean less

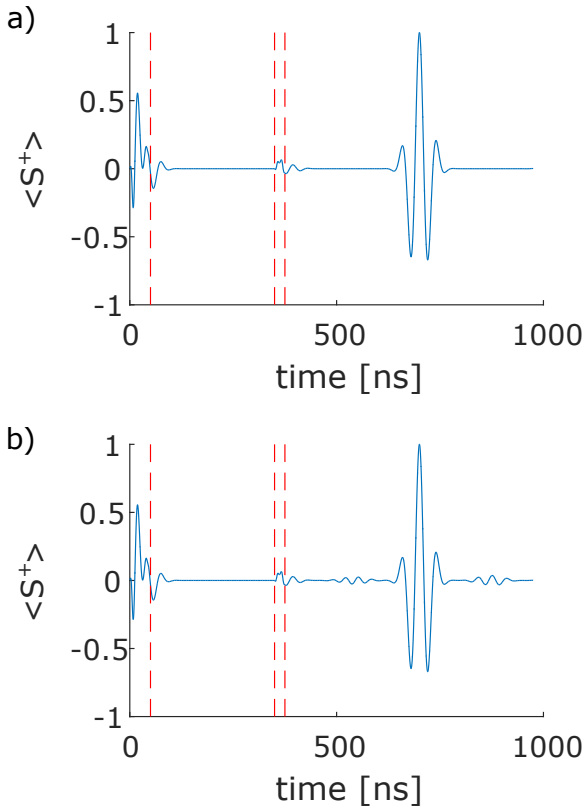


**Figure 13:** A two-pulse echo simulation with a) 201 and b) 101 spin packets. The wiggles in b) arise from back folding from the frequency (sampling of the resonance frequencies) into to the time domain.

loops, means faster simulation, right?). Set `st=0.001` in the preamble to `st=0.002`. This reduces the number of spin systems from 201 to 101. If you run the simulation again, you will notice, that it runs much faster. But you should also observe some wiggles in the trace (Fig. 13b)). These are in fact aliasing artifacts from a too large sampling interval in the frequency domain (where we sample spin packets) transformed into the time domain. You can prove this by elongating the detection time of your first echo simulation (set `.tp` from `[50,300,25,600]` to `[50,300,25,800]`) and you will see another, much smaller echo. If you increase detection time further, you can see even more, evenly spaced, echoes. You can make them disappear by reducing the sampling interval of the frequency distribution by a magnitude. But they are not really gone: by decreasing $\Delta f$, we increased the $\Delta t$ in the time domain, which means the virtual echoes will appear later, but have not vanished.

The Nyquist time range of a simulation is determined by the sampling interval in the frequency domain. If sufficiently fine sampling would be too time consuming, Monte Carlo sampling can be used. This distributes the artifact

as white noise over the whole time trace and the noise level is proportional to the square root of the number of trials.

### 5.6 Simulation of an ESEEM trace

Due to the pseudo-secular hyperfine coupling, the $\pi/2$ pulse of the echo sequence excites coherence on the allowed and the forbidden electron transitions [14], which then evolves under the Hamiltonian

$$\hat{\mathcal{H}} = \Omega_S \hat{S}_z + \omega_I \hat{I}_z + A\hat{S}_z\hat{I}_z + B\hat{S}_z\hat{I}_x \tag{4}$$

during the inter pulse delay $\tau$. After refocusing the modulation of the primary echo as a function of $\tau$ is related to nuclear transition frequencies within an electron spin manifold and the sum and difference of basic nuclear frequencies (Fig. 14). Fourier Transformation of these traces give an

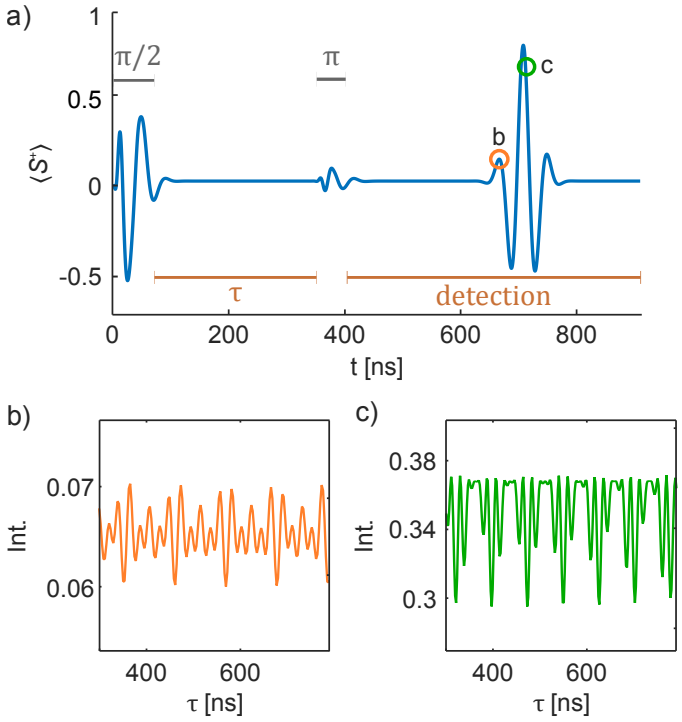


**Figure 14:** Spin echo and ESEEM simulation for the malonic acid system ($\omega_I = -14\,\mathrm{MHz}$, $A = -45\,\mathrm{MHz}$, $B = 8\,\mathrm{MHz}$) using two chirp pulses. The echo in a) was created using a Gaussian distribution of resonance freqencies with $\sigma = 35\,\mathrm{MHz}$ centered at 1 GHz. A total of 201 spin packets was used between 1.3 and 1.5 GHz. By recording a series of echos with variable $\tau$, ESEEM spectra can be obtained. The frequencies in an ESEEM trace (b) and c)) depend on the observer position, shown in a).

ESEEM spectrum which allows extraction of Hamiltonian parameters. To

obtain ESEEM traces it is necessary to repeat the experiment for a series of inter pulse delays.
In our simulation we now not only have to loop over a distribution resonance frequencies, but also over different interpulse delays:

- Define Gaussian distribution of the resonance frequency
- Vector with $\tau$-values
- Definition of spin system, pulse sequence and options
- `for`-loop over $\tau$-vector:
  - Load a $\tau$-value
  - Replace $\tau$ in pulse sequence
  - Process the pulse sequence, but not the spin system
  - Start a `for`-loop over the Gaussian distribution
    - Load a value for the resonance frequency
    - Replace the previously defined value in the interaction table
    - Only now process the spin system
    - Propagate
    - Store time trace in simulation frame (no down conversion!)
    - Repeat
  - Sum up individual time traces
  - Down convert the accumulated traces
  - Store them
- Process the signals to a 2D spectrum

This would work, but there is a more efficient way in SPIDYAN. Let us take a step back. For propagation the program solves the Liouville equation or Master equation under the assumption of stepwise time-constant Hamiltonians (for more details see the SPIDYAN paper [10]). The program takes the Hamiltonian at each time step, calculates the matrix exponential and applies it to the density matrix. Since the number of Hamiltonians is limited (our virtual AWG has a limited number of levels), it is possible to store the matrix exponentials and reuse them (as long as the spin system and the pulses are identical).
One of the output variables of the function `homerun` is `tables`, which contains all the required propagators. With this, we can now do the following:

- Define Gaussian distribution of the resonance frequency
- Vector with $\tau$-values
- Definition of spin system, pulse sequence and options
- `for`-loop over the Gaussian distribution:
  - Load a value for the resonance frequency
  - Replace the previously defined value in the interaction table
  - Process the spin system
  - Load the *first* $\tau$-value

  - Replace $\tau$ in pulse sequence
  - Process the pulse sequence
  - Propagate the system for the *first* $\tau$-value
  - Store time trace and lookup-tables
  - Start a `for`-loop over $\tau$, starting with the second value:
    * Load the $\tau$-value
    * Process the pulse sequence
    * Propagate using the lookup-tables from above
    * Store time trace
    * Repeat
- Accumulate time traces correctly
- Process the signals to a 2D spectrum

The loop over the resonance offset must be the outer loop, since the stored propagators depend on this offset and can be reused only as long as this offset remains constant. This will put a slightly higher-load on the memory, since more traces need to be stored before processing, but will increase the computation performance significantly. Another trick to speed up computation is to switch on detection *only* during the primary echo. Since we can calculate the position of the echo and its width, we can optimize the length of the detection event (event 5 in the below code) and center it around the echo position. With the same information we can set the lenght of event 4, which is the evolution of the spin system between the pulse and the echo, which does not need to be detected. It is therefore sufficient to write `sequence.detection(5)=1;`, which will create a vector with five elements of which all but the fifth are zero. The entire script for the simulation of an ESEEM spectrum of the malonic acid system can be found below:

```
t2_vec=550:3:800;

% the spectral sampling
c=1.4;      % center frequency of Gaussian distribution of ...
    spins, in GHz
g=0.01;     % width of the Gaussian distribution in GHz
nu0_s=1.3;  % starting value for sampling of Gaussian distribution
nu0_e=1.5;  % final value for sampling of Gaussian distribution
st=0.001;   % stepsize for sampling of Gaussian distribution
nu0_vec=nu0_s:st:nu0_e;     % creates vector with resonance ...
    frequencies, according to above
p=exp(-((c-nu0_vec)/g).^2); % probabilities for each spin
p=p/trapz(p);


% the system
system.sqn=[0.5 0.5];       % spin quantum number of first spin
system.interactions={1,0, 'z','e',1.4;     % Zeeman term of ...
    electron, value is replaced in loop
                     2,0,'z','e',-0.014;   % Zeeman of nucleus
```

```
                    1,2, 'z','z',-0.045;  % secular part of ...
                        hyperfine interaction , A*SzIz
                    1,2, 'z','x',0.008};  % pseudo secular ...
                        part of hyperfine interaction , B*SzIz
system.init_state='-z'; % only the electron spin is ...
    quantizised along z, nucleus is not defined, same ...
    statement would be '-ze'
system.eq = 'init';  % equilibrium state is the same as ...
    initial state

% the options
options.relaxation=0;       % tells SPIDYAN whether to include ...
    relaxation (1) or not (0)
options.down_conversion=0;  % downconversion of signal (1) or ...
    not (0)
options.det_op={'pe'};      % detection operator

% the sequence
sequence.tp=[50,300,25,500,10]; % vector with event lengths in ns
sequence.beta(1)=pi/2;        % flip angle for first event, ...
    tells SPIDYAN that 1st event is a pulse
sequence.beta(3)=pi;          % flip angle for second event, ...
    tells SPIDYAN that 2nd event is a pulse
sequence.frq={[1.3 1.5]};     % initial and final frequency of ...
    the chirp
sequence.t_rise=10;           % rise time of chirp pulses
sequence.excite={[1 0]};      % only excite the first spin ...
    during the pulses
sequence.detection(5)=1;

echos=zeros(size(t2_vec));

%Send to 'triple'
[experiment,options] = triple(sequence, options);
%%
tmax=t2_vec+0.5*sequence.tp(1);

n_echo_detection = ceil(300/experiment.dt);    % number of ...
    points to detect the echo (=length in time domain)

toplot=zeros(length(t2_vec),3601);       %creates empty matrix ...
    for spins at all t2 delays

% 'for loop' to calculate time trace for each frequency ...
    'nu0_vec', running 'k'
for k=1:length(nu0_vec)

     system.interactions{1,end}= nu0_vec(k);              % ...
         add loop resonance frequency
     [system1, state0, options]=setup(system,options);     % ...
         calls 'setup': calculates Hamiltonians, etc.
     po=p(k);
     experiment1=experiment;                               % ...
         experiment1: every loop is defined independently
```

```
    experiment1.tp(2)=t2_vec(1);
    experiment1.tp(4)=tmax(1)-n_echo_detection/2*experiment.dt;
    experiment1.tp(5)=n_echo_detection*experiment.dt; ...
                        % duration of detection time t5

    op=options.detect{1};

    [state, signal, ¬, ¬...
        ,tables]=homerun(state0,system1,experiment1,options,[]); ...
        % 'homerun' calculates timetrace

    toplot(1,:)=toplot(1,:)+signal.sf*p(k);                ...
        %sum up signal of different spin packets

    pk=p(k);

    % 'parfor loop' to calculate varying echo delays ...
        't2_vec', running 'm'
    parfor m=2:length(t2_vec)

        experiment1=experiment;
        experiment1.tp(2)=t2_vec(m);
        experiment1.tp(4)=tmax(m)-n_echo_detection/2*experiment.dt;
        experiment1.tp(5)=n_echo_detection*experiment.dt;

        [state, ...
            signall]=homerun(state0,system1,experiment1,options,tables); ...
            % 'homerun' calculates timetrace

        toplot(m,:)=toplot(m,:)+signall.sf*pk;             ...
            %sum up signal from different t2 values
    end
end

t=linspace(0,experiment.tp(end),length(toplot)); % creates ...
    time axis
signaldc=strike(toplot, t, c, options);   % down conversion of ...
    merged signals
signaldc=signaldc/max(max(signaldc));

figure(3)
plot(t2_vec(1:end),abs(toplot(1:end,1300)))
xlabel('\tau [ns]')
ylabel('Int.')

figure(4)
plot(t2_vec(1:end),abs(toplot(1:end,1849)))
xlabel('\tau [ns]')
ylabel('Int.')
```

## 5.7 Polarization enhancement

Another interesting aspect of passage pulses is observed for high-spin systems $S > 1/2$. During a chirp pulse each transition in the spectrum is passed separately, which makes it possible to move polarization between levels. Fig. 15a shows that for single crystal of Gd(III) with $S = 7/2$, two consecutive chirp pulses (orange and blue) with opposite sweep directions can be chosen such that the populations from the $m_S = -7/2$ and $m_S = 7/2$ levels are transfered to the central transition (highlighted in red).

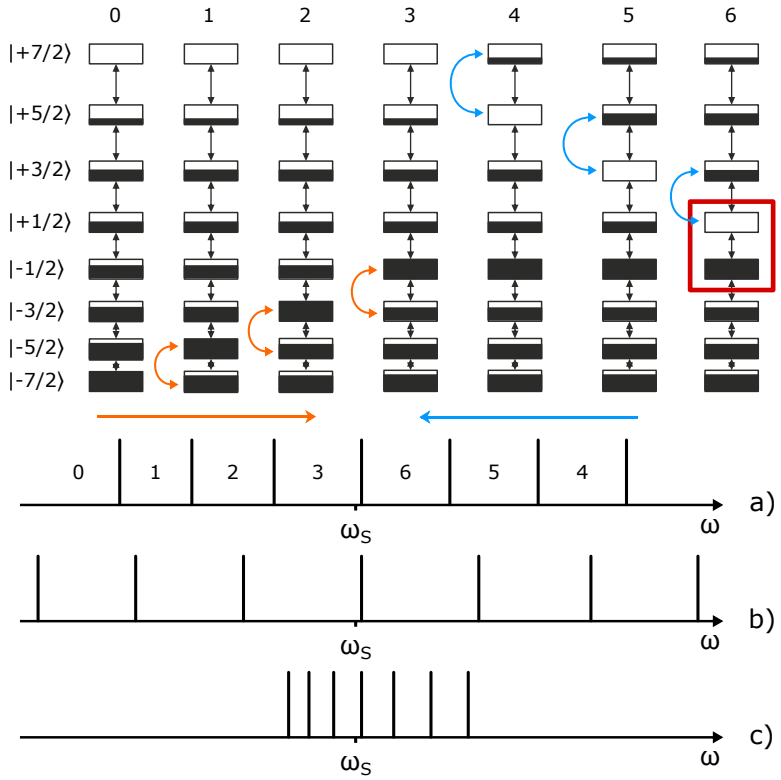


**Figure 15:** By using two consecutive chirp pulses (orange and blue) each transition in a high-spin system can be inverted selectively. This leads to an enhancement of the central transition (red).The labelling 0 - 6 corresponds to the progression of the chirp pulses. For example at point 0 neither chirp has hit a transition yet. At point 1, the up sweep has inverted the outermost levels. Since zero-field splitting frequencies depend on the orientation of the spin system to the magnetic field and a distribution of $D$ and $E$ values, not all transitions may be passed by a pulse in a powder. These two effects can lead to the three depicted cases a), b) and c) which are illustrated in Fig. 16.

Starting from equilibrium polarization in the high-temperature limit (situation 0 in the energy scheme), the polarization difference between each pair of neighboring transitions is $\epsilon$. After adiabatic passage of the first level (situation 1) with an up sweeping chirp pulse (from low to high frequency),

polarization on transition ($-5/2 \leftrightarrow -3/2$) is $2\epsilon$. This can be repeated for every passage of the first pulse until population from $m_S = -7/2$ is completely transferred to $m_S = -1/2$ (situation 3 in Fig. 15).
The same procedure is then repeated from the other side: A second chirp sweeps from high frequencies to lower frequencies (down sweep). Now population is transferred from the $m_S = 7/2$ level to the $m_S = 1/2$. This maximizes the polarization of the central transition ($-1/2 \leftrightarrow 1/2$).
It is important to choose the two chirp pulses such that they do not affect the central transition, otherwise the polarization enhancement is strongly reduced.
Using transition selective operators this shuffling of polarization is depicted in Fig. 16 in a Gd(III) system. In the case of a single crystal of a Gd(III) compound in a suitable orientation with respect to the static magnetic field, where all transitions are within the accessible excitation band, the pulses can be configured such that an enhancement of almost 7 can be reached by passage of all 6 transitions, Fig. 16a. For orientations where not all transition fall inside the excitation band or have different $D$ and $E$ values, the enhancement might be reduced (Fig. 16b) or not happening at all (Fig. 16c).
Compared to a suitably aligned single crystal. the situation is less favorable for a powder sample, where one has to consider the orientation dependence of the zero-field splitting and, for a Gd(III) complex with organic ligands, a distribution of $D$ and $E$ values.
The MATLAB code for the simulation of polarization enhancement as in Fig. 16 is given below. Again we start by defining the spin system.

```
%system
system.interactions={1,0,'z','e',1.5};  % Zeeman term
system.sqn=7/2; % spin quantum number
system.D=0.1;   % ZFS parameters in GHz
system.E=0.01;
system.phi=0;   % orientation of the crystal
system.theta=pi;
system.init_state='-z';  % initial state
```

Whenever SPIDYAN comes across the Zeeman term of a spin with spin quantum numbers larger than 1/2, the program looks if the vectors `.D` and `.E` and an orientation (`.phi` and `.theta`) are provided. The zero-field splitting is then applied through perturbation theory to the Zeeman-Hamiltonian term *in the lab frame*. Since perturbation terms need to be calculated in the lab frame, it is necessary to also provide the difference between the LO and the lab frame with `options.LO`. For simulations that are conducted in the LO frame, the Hamiltonian term is then down converted again.
If the system contains several spins with zero-field splitting, `.D` and `.E` can be assigned for all of spins to be the same (length = 1) or separately (length is the same as number of spins).

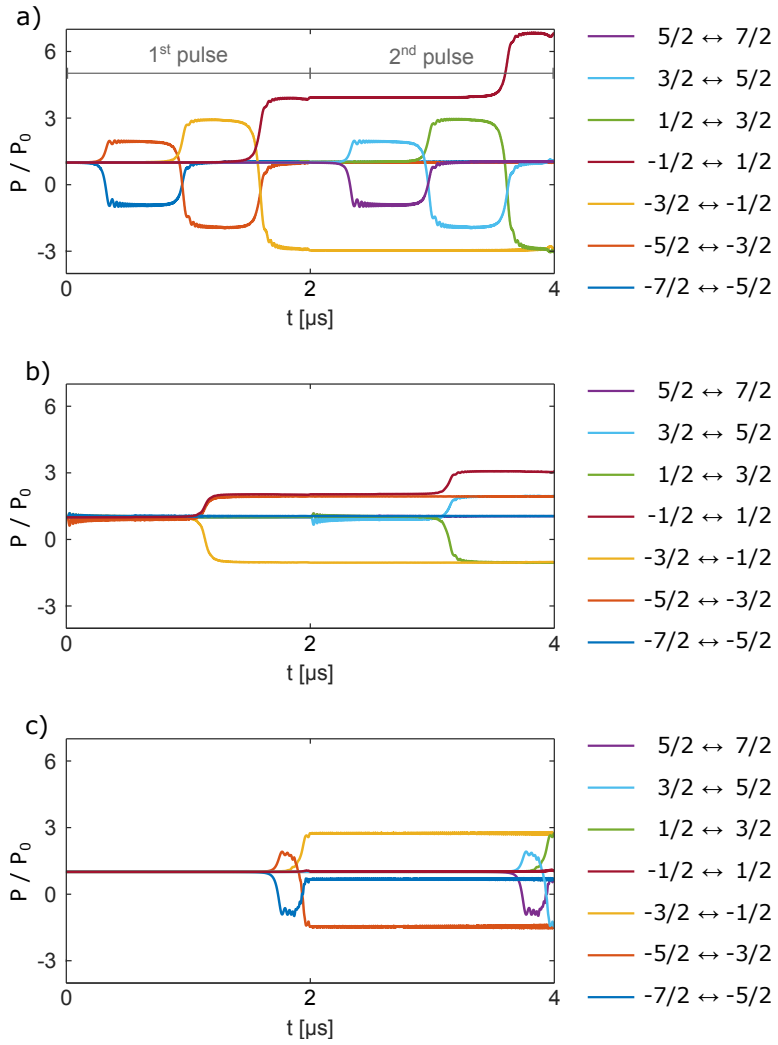


**Figure 16:** Population transfer simulation for $S = 7/2$ with a) $D = 100\,\mathrm{MHz}$ b) $D = 180\,\mathrm{MHz}$, c) $D = 30\,\mathrm{MHz}$ and $E = D/10$, compare to Fig. 15. The first chirp sweeps from -0.70 to -0.07 GHz, the second from 0.70 to 0.07 GHz. Both pulses have an irradiation strength of 4 MHz and a length of 2 $\mu$s, as depicted in a). a) When the pulse passes all transitions in the system, an enhancement of the central transition ($1/2 \leftrightarrow -1/2$) of almost 7 is possible. b) In the case when the chirp does not invert all outer satellite transitions, less polarization is moved to the central transition. c) For a very small zero-field splitting, the pulse can not invert the innermost transitions, and therefore the central transition is not enhanced at all.

If no `.D` and `.E` are found, or the corresponding values are zero, zero-field splitting is ignored. Alternatively it is also possible to provide the zero-field splitting through the interaction table, by providing the term symbols and interaction quantities.

Next we can define the pulse sequence:

```
sequence.tp=[2000 2000]; % length of the pulses in ns
sequence.nu1=[4 4];      % amplitude of the pulses
sequence.frq={[0.8 1.43] [2.2 1.57]};
sequence.detection = [1 1]; % detect during both pulses
sequence.t_rise = 30;    % rise time in ns
```

This creates two consecutive pulses, each with a length of 2000 ns and a pulse amplitude of 4 MHz. The first pulse sweeps from 0.8 to 1.43 GHz (up sweep) and the second one from 2.2 to 1.57 GHz (down sweep). Since we are interested in what happens with the spin system during the pulses, detection is switched on for both events.
Before processing it we need configure some options:

```
options.det_op={'-z1,2','-z2,3','-z3,4','-z4,5',...
                '-z5,6','-z6,7','-z7,8'}; % transition ...
                    selective detection operators
options.LO=8;  % frequency of the local oscillator
```

In order for us to detect the polarization enhancement, we have to use transition selective operators (Section 2.3.1). They allow us to observe changes between each pair of levels (black arrows in Fig. 15). The number after `'z'` corresponds to the position in the density matrix. For example `'z12'` measures the population difference between the first and second diagonal element.
In addition we have to provide the frequency difference of the LO frame (1.5 GHz as defined in the interaction table) to the lab frame (9.5 GHz for example for X-band) with `.LO`.
The last step is processing input, propagation and plotting of the time traces.

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

[system, state, options]=setup(system,options);

[state, signal]=homerun(state,system,experiment, options, []);

figure(1),clf
title('Polarization on detected transitions')
plot(signal.t(1:10:end),real(signal.sf(:,(1:10:end))))
legend(options.det_op,'Location','NorthWest')
xlabel('t [ns]')
ylabel('<S_q>')
xlim([0 sum(sequence.tp)])
ylim([-4 7])
```

Now go ahead and change some parameters of the script and observe their influence on the polarization enhancement. For example you could start by reducing the sweep width:

```
sequence.frq={[1.0 1.43] [2.0 1.57]};
```

What happens if the pulses finish closer to the center frequency of the spectrum

```
sequence.frq={[0.8 1.5] [2.2 1.5]};
```

or farther away from it

```
sequence.frq={[0.8 1.2] [2.2 1.8]};
```

Or you can change the orientation of the spin single crystal with `.phi` and `.theta` or the zero field splitting parameters `.D` and `.E`.

# References


[1] P. E. Spindler, Y. Zhang, B. Endeward, N. Gershernzon, T. E. Skinner, S. J. Glaser, T. F. Prisner, *J. Magn. Reson.* **2012**, *218*, 49–58, DOI 10.1016/j.jmr.2012.02.013.

[2] A. Doll, S. Pribitzer, R. Tschaggelar, G. Jeschke, *J. Magn. Reson.* **2013**, *230*, 00015, 27–39, DOI 10.1016/j.jmr.2013.01.002.

[3] P. E. Spindler, S. J. Glaser, T. E. Skinner, T. F. Prisner, *Angew. Chem. Int. Ed.* **2013**, *52*, 3425–3429, DOI 10.1002/anie.201207777.

[4] T. Kaufmann, T. J. Keller, J. M. Franck, R. P. Barnes, S. J. Glaser, J. M. Martinis, S. Han, *J. Magn. Reson.* **2013**, *235*, 95–108, DOI 10.1016/j.jmr.2013.07.015.

[5] A. Doll, G. Jeschke, *J. Magn. Reson.* **2014**, *246*, 18–26, DOI 10.1016/j.jmr.2014.06.016.

[6] P. Schöps, P. E. Spindler, A. Marko, T. F. Prisner, *J. Magn. Reson.* **2015**, *250*, 55–62, DOI 10.1016/j.jmr.2014.10.017.

[7] J. M. Franck, R. P. Barnes, T. J. Keller, T. Kaufmann, S. Han, *J. Magn. Reson.* **2015**, DOI 10.1016/j.jmr.2015.07.005.

[8] P. E. Spindler, I. Waclawska, B. Endeward, J. Plackmeyer, C. M. Ziegler, T. Prisner, *J. Phys. Chem. Lett.* **2015**, DOI 10.1021/acs.jpclett.5b01933.

[9] G. Jeschke, S. Pribitzer, A. Doll, *J. Phys. Chem. B* **2015**, *119*, 13570–13582, DOI 10.1021/acs.jpcb.5b02964.

[10] S. Pribitzer, A. Doll, G. Jeschke, *Journal of Magnetic Resonance* **2016**, *263*, 45–54, DOI 10.1016/j.jmr.2015.12.014.

[11] M. S. Silver, R. I. Joseph, D. I. Hoult, *Journal of Magnetic Resonance (1969)* **1984**, *59*, 347–351, DOI 10.1016/0022-2364(84)90181-1.

[12] E. Kupce, R. Freeman, *Journal of Magnetic Resonance Series A* **1995**, *115*, 273–276, DOI 10.1006/jmra.1995.1179.

[13] A. Tannús, M. Garwood, *J. Magn. Reson. A* **1996**, *120*, 133–137, DOI 10.1006/jmra.1996.0110.

[14] A. Schweiger, G. Jeschke, *Principles of pulse electron paramagnetic resonance*, 01150, Oxford University Press, Oxford, UK ; New York, **2001**, ISBN: 0-19-850634-1.