A Rigorous Calculation of Pulsed EPR SECSY and Echo-ELDOR Signals: Inclusion of Static Hamiltonian and Relaxation during Pulses

Sushil K. Misra and Hamid Reza Salahi

Physics Department, Concordia University, 1455 de Maisonneuve Boulevard West, Montreal, Quebec H3G 1M8, Canada

Abstract

A procedure is developed to calculate pulsed electron paramagnetic resonance (EPR) signals with relaxation rigorously including the static Hamiltonian and relaxation during pulses by solving Liouville von Neumann (LVN) equation numerically in Liouville space. It can be carried out within a reasonable time on a PC using Fortran or Matlab. It is illustrated here numerically, as coded in Matlab, to calculate the spin echo correlation spectroscopy (SECSY) and echo-electron-electron double resonance (echo-ELDOR) signals for a coupled electron-nuclear system with the electron spin (S = ½) and nuclear spin (I = ½) for the experimental results of Lee, Patyal and Freed [1] in a malonic acid single crystal.

Keywords: Electron Paramagnetic Resonance (EPR), Pulsed EPR, Two-dimensional spin-echo-correlation spectroscopy (SECSY), echo-electron-electron double-resonance (echo-ELDOR), Liouville von Neumann equation (LVN).

Introduction

The technique of pulsed electron paramagnetic resonance (EPR) is very powerful in that it can be exploited to reveal the electronic and geometric structures of the environment around paramagnetic centers in detail [1-5]. Even weak interactions between electron spins, as well as those between electron and nuclear spins not resolved by continuous wave (CW) EPR, can be distinguished by pulsed EPR. A quantitative analysis to extract precise information on electronic and geometric structure from pulsed EPR data, therefore, requires a rigorous simulation of pulsed EPR spectra.

In a previous publication by Misra and Li [6] (hereafter ML), calculations were made of pulsed EPR without taking into account the static Hamiltonian and relaxation during the pulses. As well, the spin Hamiltonian used had inherited some significant typographical errors from the original source, Lee, Patyal and Freed [1] (hereafter LPF) which led to erroneous results. As a consequence, the theoretical calculations were not quite in good agreement with the experimental data. It is the purpose of the present paper to extend the calculations of Misra and Li [6] to calculate pulsed EPR signals in the presence of relaxation as well as the static spin Hamiltonian during the pulses using the correct expressions.

Although, there are available two open-source packages that are implemented in Matlab, which are much more general than the approach introduced in the present manuscript, namely SPINACH [7] and SPIDYAN [8], the present procedure, on the other hand, offers an advantage in that within the framework of its applicability it can be carried out on a laptop within a short time using Matlab or Fortran, and can be applied to calculate echo-signals of a user-defined pulse sequences, including any kind of relaxation processes. From this point of view, it is felt that this work serves a useful purpose to practitioners of pulsed EPR spectroscopy needing to treat relaxation rigorously in a straightforward manner.

In order to take into account relaxation rigorously, it is imperative to use Liouville von Neumann (LVN) equation, which is an exact quantum-mechanical equation of motion for the density matrix, in Liouville space. This equation is valid even for relatively slow random processes, and is therefore especially suitable for EPR, where the natural time scale is short so that the random processes are not usually fast on this time scale. The time-dependent LVN equation used here includes a relaxation term and a time-dependent, but not stochastically time dependent, Hamiltonian, e.g. a Hamiltonian representing the pulses.

The general procedure for setting up the simulation for the LVN equation taking into account the relaxation and static Hamiltonian during pulses for pulsed EPR experiments is briefly given in Sec 4. (For details, refer to [6]). Thereafter, the details of calculation of spin-echo-correlation spectroscopy (SECSY) and echo-electron-electron double-resonance (echo-ELDOR) signals, including selection of coherent electron pathways, are given in Section 5. The results are discussed in Section 6. The conclusions are presented in Section 7. For illustration, simulations of pulsed EPR spectra for SECSY and echo-ELDOR experiments are carried out in the Appendix for the cases investigated by LPF in a malonic acid single crystal. The Matlab source code is included at the end.

Solution of the LVN equation

The time dependence of the density matrix, \( \rho \), taking into account the relaxation effects is expressed in Liouville space as follows [6, 9-13].
\[
\frac{d}{dt} \rho(t) = -\{\hat{H}, \rho(t)\} - \hat{\Gamma} (\rho(t) - \rho_0)
\]
where \(\hat{H} = \hat{H}_s + \hat{H}_r\) is the Hamiltonian operator; \(\hat{H}_s\) and \(\hat{H}_r\) are its time-independent and time-dependent parts, respectively; \(\hat{\Gamma}\) is the relaxation superoperator, and assumed to be time independent here. (Throughout the paper, the single and double carets \(^\ast\) and \(^\ast\ast\) will be used to denote the time and the superoperator, respectively.) In Eq. \(\rho_0 \propto S_e\) is the initial thermal equilibrium density matrix, as discussed in by ML [6].

The difference between the time-dependent density matrix and the equilibrium density matrix, i.e. the reduced density matrix, is expressed as follows:

\[
\frac{d}{dt} \chi(t) = -\{\hat{H}, \chi(t)\} - \hat{\Gamma} \chi(t)
\]

The operator equation (2.2), can be expressed as a matrix equation in a given set of operators, where \(\chi\), \(\hat{H}\), \(\hat{\Gamma}\) in the expansion of the operators \(\chi\) and \(\hat{H}\) in this basis, respectively.

**Evolution of the density matrix in the absence of a pulse (free evolution)**

Because the Liouville equation in the matrix form is a differential equation in Liouville space in a chosen basis defined in the preceding section, where the operators are represented as matrices, can be considered as a column, as follows:

\[
\frac{d}{dt} \chi = \text{Col}[-\{\hat{H}, \chi(t)\}] - \hat{\Gamma} \chi(t)
\]

Here \(\chi = \text{Col}(\chi(t))\) denotes the column of the matrix \(\chi(t)\) formed by stacking the columns of \(\chi_0\) into a single column vector as a matrix.

The Liouvillian

\[
\hat{L} = (I_n \otimes \hat{H} - \hat{H}^T \otimes I_n)
\]

with the dimension \(n\), where \(I_n\) is the unit matrix, and \(\hat{H}^T\) denotes the matrix transpose of \(\hat{H}\).

As for the second term on the right-hand side of Eq. (2.3), it can be expressed in the matrix form as:

\[
\hat{R} \chi = \text{Col}(\hat{\Gamma} \chi(t))
\]

where the double-subscripted matrix \(\hat{R}\) is derived from the 4-subscripted matrix \(\hat{\Gamma}\) in Liouville space as

\[
\hat{R}_{\alpha \times N + \beta} = \hat{\Gamma}_{\alpha \beta \alpha' \beta'}; \alpha, \beta, \alpha', \beta' = 1, 2, \ldots n
\]

and \(\hat{R}\) is the relaxation superoperator in Liouville space, introduced phenomenologically.

It is shown in ML [6] that the LVN equation is expressed in matrix form as follows:

\[
\frac{d}{dt} \chi = -\hat{L} \chi
\]

where the generalized Liouvillian superoperator \(\hat{L}\), includes also the relaxation superoperator:

\[
\hat{L} = \hat{L} + \hat{R}
\]

The solution of Eq. (2.7) is

\[
\chi(t) = e^{-t_{0}-t_{t}} \chi(t_{0})
\]

In Eq. (2.9), \(\chi(t_{0})\) is the reduced density matrix at the beginning of the evolution.

**Evolution of the density matrix under the action of a pulse including the static Hamiltonian and Relaxation**

During the pulse, to be rigorous, one should take into account the combined action of \(\hat{H}_s\), \(\hat{H}_r\) and \(\hat{\Gamma}\) where \(\hat{H}_s\) is the static Hamiltonian, that given by Eq. (A.2) in Appendix below, and the pulse operator, \(\hat{H}_r(t)\), is expressed as

\[
\hat{H}_r(t) = \hat{\vec{e}}(t) = B_\gamma e^{i(\omega \phi + S_x \sin(\phi) + S_y \cos(\phi) + S_z \sin(\phi))}
\]

where \(\gamma\) is the gyromagnetic ration of the electron, and \(\hat{\vec{e}}(t)\) is the irradiating microwave pulse with the intensity \(B_\gamma = e^{i\omega t}\) and phase \(\phi\). The tip angle \(\theta\) over time \(t\) during the pulse is, \(\theta = \omega t = B_\gamma e^{i\omega t}\).

Finally, following the same procedure that led to Eq. (2.7), the density matrix can be expressed as follows:

\[
\hat{\rho}(t) = e^{-t_{0}-t_{t}} \hat{R} \hat{\rho}(t_{0})
\]

The solution of Eq. (2.11) is

\[
\hat{\rho}(t) = e^{-t_{0}-t_{t}} \hat{R} \hat{\rho}(t_{0})
\]

Here \(\hat{\rho}(t_0)\) is the initial density matrix when the pulse is applied.

**Simulation procedure**

The algorithm for the calculation of the pulsed EPR signal is described in detail in [6]. Here only the outline is presented. The time evolution of the density matrix includes evolution under the action of the static Hamiltonian, \(\hat{H}_s\) (free evolution) in the absence of pulses, and under the action of the pulse \(\hat{H}_r\) including the static Hamiltonian and relaxation, is discussed in the Appendix.

The calculation of the final density matrix, \(\hat{\rho}_f\), corresponding to the coherence pathways \(S_{\text{coh}}\) for obtaining the SECSY and echo-ELDOR signals, as shown in Figures 1 and 2, uses Eqs. (2.9) and (2.13), respectively. In this paper, the signal is calculated over the photon pathway \(S_{\text{coh}}\) in accordance with that used by LPF [1] (for more details of coherence pathways, see [6]). In these experiments, the time \(t\) between the pulses, for time-domain signals, is stepped in the experiment and the time \(t_0\) is measured from the top of the echo, as shown in Figures 1 and 2.
The 2D time-domain signal is calculated from \( \rho f \) as follows:

\[
S(t_1,t_2) = Tr(S_{t_2} \rho f) = Tr\left( S_{t_2}^\dagger S_{t_1} \rho f \right)
\]

The Fourier transform (FT) of the two-dimensional time domain signal \( S(t_1,t_2) \), is the corresponding 2D-FT signal, \( S(\omega_{t_1},\omega_{t_2}) \). The calculations are carried out in the rotating frame, in which the effective magnetic field becomes equal to zero at resonance, as described in [6].

**Simulation of SECSY and echo-ELDOR signals**

The technique developed here is used to illustrate the cases for SECSY and echo-ELDOR signals obtained in an irradiated malonic acid single crystal [1], which is a radical produced by irradiation, so the system possesses an electron spin \( S=1/2 \) coupled to a nuclear spin \( I=1/2 \). The calculated spectra are compared with the experimental results of LPF [1].

Here one consider the case of an unpaired electron spin \( S=1/2 \), interacting with a single nucleus \( I=1/2 \), by hyperfine (HF) interaction, wherein the principal axes of the HF matrix \( A \) and those of the \( g \) matrix are coincident. More details of the procedure used by LPF, including the spin Hamiltonian and basis vectors used, are given in Appendix 1. In this experiment, the various parameters are as follows: the \( \pi/2 \) pulse is of duration \(-5\mu s\) [1]; nuclear Zeeman frequency \( \omega_0=14.5MHz \); the spin-Hamiltonian parameters:

\[
\vec{g}=(g_{xx},g_{yy},g_{zz})=(2.0026,2.0035,2.0033) \quad \text{and} \quad A=(A_{xx},A_{yy},A_{zz})=(-61.0MHz,-91.0MHz,-29.0MHz).
\]

The input values used in the simulation of the time-domain signals, as described in Appendices A.1 and A.2, are as follows [1]: Gaussian inhomogeneous broadening \( \Delta=4MHz \); electron spin-spin relaxation time, \( T_{2e}=0.900\mu s \); nuclear spin-spin relaxation time, \( T_{2n}=22\mu s \); inverse electron spin-spin relaxation time, \( W_{pe}=0.0167\mu s^{-1} \); inverse nuclear spin-spin relaxation time, \( W_{pn}=0.00714\mu s^{-1} \); inverse electron nuclear spin-spin relaxation time, \( W_{en}=0.00617\mu s^{-1} \); inverse electron-nuclear spin-spin relaxation time \( W_{en}=W_{ej} \); inverse Heisenberg exchange relaxation time, \( \omega_{HE}=0.0\mu s^{-1} \).

The direction of the external static field \( B_0 \) which is defined by the angles \( \theta \) and \( \phi \), where \( \theta \) is the angle between \( B_0 \) and the z axis, and \( \phi \) is the angle between the x axis and the projection of \( B_0 \) on the xy plane is depicted in Figure 3.

The calculated SECSY time-domain signals are shown: (i) in Figure 4, corresponding to the experiment of LPF [1] for three orientations of the external magnetic field with respect to the crystal axes: \( (\theta,\phi)=(\pi/2,0) \), \( (\pi/4,\pi/2) \), and \( (0,0) \) in the zy-quadrant, so that the corresponding Euler angles are \( (\alpha,\beta,\gamma)=(0,-0,0) \) and \( (45,0,90) \) in the zy-quadrant, which correspond to \( (\alpha,\beta,\gamma)=(0,0,90) \). [The relationship between (\( \theta,\phi \)) and (\( \alpha,\beta,\gamma \)) is described by Misra et al. [18].]

As for the echo-ELDOR signal, the time-domain signals, were calculated for the orientation \( (\theta,\phi)=30,0 \) in the zx-quadrant, so that \( (\alpha,\beta,\gamma)=(0,-0,0) \) [1], with four mixing times \( T_m \) (the fixed time interval between the second and third \( \pi/2 \) pulses; see Figure 2): (a) 5\( \mu s \); (b) 20\( \mu s \); (c) 40\( \mu s \); (d) 60\( \mu s \). They are shown in Figure 6.

One can now compare the Fourier transforms (FT) of the signals as obtained experimentally by LPF [1] with those simulated here for SECSY and echo-ELDOR signals, as shown in Figures 4–6. It
is found, in general, that they look quite the same to the eye. In particular, for echo-ELDOR signals the corresponding positions of the main frequency peaks, i.e. the nuclear modulation frequency $\omega_\alpha$ and $\omega_\beta$, are the same in all four cases shown in Figure 6. Their values are the same as those reported in LPF [1], i.e. $\omega_\alpha \approx 7.0 \text{MHz}$ and $\omega_\beta \approx 32.0 \text{MHz}$. The shapes of the simulated spectra in the frequency domain as calculated here and those calculated by LPF [1] appear to be in excellent agreement with each other. It takes about 10-15 secs to carry out the calculations on a laptop for the two types of pulse sequences considered here.

The overlapped contour plots in the Fourier-transform domain for SECSY and echo-ELDOR signals as obtained using the procedure used in this work, including both the static Hamiltonian and relaxation during the pulses, and those of LPF [1], without inclusion of the static Hamiltonian and relaxation during the pulses, are shown in Figures 7 and 8, respectively. It is clear from these figures that adding the static Hamiltonian and relaxation during the pulses modifies significantly the peaks in the FT spectra; this fact is more evident in the coherence cross peaks.

Furthermore, one can determine the effects of inclusion of static Hamiltonian and relaxation during the pulses separately from Figures 9 and 10. In these figures, the 1D spectra along $f_2$ shown with the slice along $f_1=0$ in the Fourier domain for SECSY and echo-ELDOR are compared for four different cases: (i) with the static Hamiltonian and relaxation terms included during the pulses; (ii) with the static Hamiltonian included but without the relaxation during the pulses; (iii) without the static Hamiltonian but with the relaxation included during the pulses; and (iv) without both the static Hamiltonian and relaxation included during the pulses. It is seen clearly from these two figures that the effect of inclusion of the static Hamiltonian during the pulses is very significant, whereas that of the relaxation is negligible, as explained in Section 6 below.

**Discussion of results**

In the theoretical expressions presented in LPF, Eqs. (5) and (7), the static Hamiltonian and relaxation were not taken into account during the pulses. This was justified by the fact that the duration of the pulses was short ($\sim 5 \text{ ns}$) and the pulses were intense. However, for a more rigorous simulation, one should indeed include them during the pulses.

It is seen from the present simulations that inclusion of the static Hamiltonian during the pulses does change the spectra significantly, which enhances as the number of pulses in an experiment increases (SECSY-2 pulses versus echo-ELDOR-3 pulses). On the other hand, taking into account the relaxation during the pulses does not have any significant effect on the spectra, since the spin-lattice and spin-spin relaxation times ($T_1$ and $T_2$) are several orders of magnitude longer than the duration of the pulses.

**Concluding remarks**

This paper deals with a rigorous calculation of pulsed EPR signals in the presence of relaxation using the LVN equation in Liouville space, providing a comprehensive theoretical treatment, taking into account relaxation processes, both during pulses and in their absence. As well, the static Hamiltonian is included during the pulses. The procedure of how to implement the theoretical approach...
numerically has been illustrated, and the algorithm required
has been thoroughly discussed, and illustrated by examples. The
Matlab source code is included here in the Appendices A-L, which
can be used for both polycrystalline (powder) and single-crystal
simulations. (We are grateful to Lin Li for providing us with a
Matlab source code for pulsed EPR calculations.) The algorithm is
illustrated here using MATLAB to calculate SECSY and echo-ELDOR
signals for the system of an electron-nuclear spin coupled system
in a malonic acid crystal and compared with experimental results
of LPF [1]. These calculations can be carried out on a commonly
available lap top within a reasonable time, on the order of 10-15
seconds

The numerical calculations show that the effect of inclusion of the
static Hamiltonian during the pulses does have a significant on the
signals, whereas that of the relaxation is negligible.

Acknowledgments

We are grateful to NSERC (Natural Sciences and Engineering
Research Council of Canada) for financial support.

Appendix: An electron-nuclear spin-coupled system in
an irradiated malonic acid crystal

This appendix describes the corrected theoretical expressions for
the calculation of pulsed EPR experiments as discussed by LPF [1],
relevant to the simulations presented in this paper.

Static spin Hamiltonian and parameters

For the specific case of a single nucleus (I = ½) interacting with an
unpaired electron (S = ½) by the hyperfine (HF) interaction tensor,
where the HF has the same principal axes as the g matrix, the total
Hamiltonian can be expressed as the sum of static Hamiltonian and
pulse Hamiltonian [1]

\[
\hat{H} = \hat{H}_0 + \hat{H}_1
\]

where

\[
\hat{H}_0 = C S_z - \omega_0 I_z + A S_z I_z + \frac{1}{2} B S_z I_+ + \frac{1}{2} B^* S_z I_-
\]

The coefficients in Eq. are expressed as follows [1]:

\[
C = \frac{\beta_e B_0}{\hbar} \left[ \frac{1}{2} \left( 3 \cos^2 \beta - 1 \right) + F(2) \sin^2 \beta \cos(2\gamma) \right]
\]

\[
A = 2 \pi \left[ \frac{B}{2} \left( 3 \cos^2 \beta - 1 \right) + D(2) \sin^2 \beta \cos(2\gamma) \right]
\]
Figure 6: Experimental FT (LPF Figure 11[1]) (first column) and simulated time-domain with relaxation taken into account (second column) of the echo-ELDOR spectra at $(\theta, \phi)$ orientations of $(30^\circ, 0^\circ)$ in the $zx$-quadrant $[(\alpha, \beta, \gamma) = (0, -\theta, 0)]$ [1], with the mixing times $T_m$: (a) 5 μs; (b) 20 μs; (c) 40 μs; (d) 60 μs. The corresponding FT figures are shown in the last column. A Gaussian inhomogeneous broadening width $\Delta=4$ MHz, Eq. (A.14), is used in the simulation.
Here $\Omega(\beta,\gamma)$ are the Euler angles which describe the orientations of the principal axes of the $g$-matrix with respect to the static magnetic field. (It is noted that in LPF [1], there were misprints, which have been corrected in the above equations, and this led to significantly erroneous numerical results in ML [6]. In particular, instead of having the factors of $2\pi$ in Eq. (A.4) and $4\pi$ in Eq. (A.5), there was the factor $\frac{2\pi}{h}$ in both places in LPF [1].)

$$F=\frac{2}{3}\left(g_{zz}-\frac{1}{2}(g_{xx}+g_{yy})\right);$$  \hspace{1cm}  (A.8)  

$$D=\frac{2}{3}\left(2A_{zz}-\frac{1}{2}(A_{xx}+A_{yy})\right);$$  \hspace{1cm}  (A.9)  

$$F^{(2)}=\frac{1}{2}(g_{xx}-g_{yy});$$  \hspace{1cm}  (A.10)  

$$D^{(2)}=\frac{1}{2}(A_{xx}-A_{yy});$$  \hspace{1cm}  (A.11)  

Here $\mathbf{g}$ and $\mathbf{A}$ are the isotropic parts of $\tilde{g}$ and $\tilde{A}$ matrices, respectively.
The Relaxation matrix

The effect of spin relaxation is taken into account by the use of the phenomenological relaxation matrix based on Redfield theory [12] as outlined in LPF [1]:

$$\frac{d}{dt} \rho_{aa'}(t) = -i \omega_{aa'} \rho_{aa'} - \sum_{\beta \beta'} \widetilde{R}_{a \beta} \beta \beta'(\rho_{\beta \beta'} - \rho_{\alpha \beta'}) \quad (A.12)$$

Where

$$\omega_{aa'} = E_a - E_{a'}$$

Here $E_a$, $E_{a'}$ are the eigenvalues of the static Hamiltonian, $\hat{H}_0$, for the electron-nuclear coupled system ($S = \frac{1}{2}$, $I = \frac{1}{2}$).

In Eq. (A.12), $\widetilde{R}_{a \beta} \beta \beta'$ are relaxation matrix elements, where $\alpha, \alpha'$, $\beta, \beta'$ designate the eigenstates of the Hamiltonian $\hat{H}_0$. The following specific values for the matrix elements, as given by Freed [14-16], can be found in [6].

Gaussian inhomogeneous broadening effect

In accordance with LPF [1], the Gaussian inhomogeneous broadening effect, over and above the relaxation effect, in the frequency-domain along $\omega_2 = 2 \pi \nu$, corresponding to the step time $t_2$, as depicted in Figures 1 and 2, is taken into account by the following time-domain dependence:

$$f(t_2) = \int_{-\infty}^{\infty} \frac{\omega}{2 \Delta^2} \exp\left\{-\frac{\omega^2}{2 \Delta^2}\right\} e^{-i 2\pi \nu_2 t_2} dt_2 \exp\left\{-2 (\pi \Delta t)^2\right\} \quad (A.14)$$

where $f(t_2)$ is the Gaussian-broadened signal along $t_2$ and $\Delta$ is the Gaussian inhomogeneous broadening parameter expressed in frequency.

Matlab source code: It consists of the main program and the accompanying input classes as included in the various appendices below. (Note: Here the “powder” is used synonymously to “poly-crystalline”)

Appendix A. Main function: Single&Powder_Main

Appendix B. Hamiltonian factors: HamiltonianFactor_class

Appendix C. Hamiltonian operator: Hamiltonian_class

Appendix D. Density matrix: Density_class

Appendix E. Free evolution operator: FED_class

Appendix F. Pulse operator: Pulse_class

Appendix G. Relaxation matrix: Relaxation_class

Appendix H. Pathway operator: Pathway_class

Appendix I. Calculation of the time domain signals according to the pulse sequence and the pathway defined in Signal_class

Appendix J. Simulated image plot: Plot_class

Appendix K. Input and output of data: IO_class

Appendix L. Input file for parameters: Data/Single&Powder_Data.txt

Procedure: Create a directory, e.g. ELDOR_SECSY. Open matlab, create a new matlab script (file), copy inside it the text given in Appendix A, and save it in the directory just created, e.g. ELDOR_SECSY, naming it “Single&Powder_Main”. This will create a file named “Single&Powder_Main.m”. Then add to this directory, e.g. ELDOR_SECSY, successively matlab scripts (files) with the texts in the various appendices, giving them the same names as those of the appendices. Finally, in that directory, e.g. ELDOR_SECSY, create two new folders and name them “Data” and “Figures”. Now create a “Single&Powder_Data.txt” file inside the “Data” folder and copy the contents of Appendix L in it. Next, click on the matlab file “Single&Powder_Main.m” and then click on the green “Run” button in the Matlab ribbon on top to execute the source code. This will result in questions on the monitor, asking (i) the number of pulses (enter 2 for SECSY and 3 for echo_ELDOR); (ii) nthetas: enter the number of theta values on the unit-sphere grid (enter 1 for single-crystal orientation simulation); and (iii) nphips: enter the number of phi values on the unit-sphere grid (enter 1 for single-crystal orientation simulation). (i) For the choice nthetas=nphips=1, you will be asked first ‘Please enter the rotate angle theta (0 - 180)’: . Thereafter you will be asked ‘Please enter the rotate angle phi(0 - 360): ‘; (ii) For the only choice nthetas=1, you will be asked ‘Please enter the rotate angle theta (0 - 180): ‘; (iii) For the only choice nphips=1, you will be asked ‘Please enter the rotate angle phi(0 - 360): ‘. Enter these values with a carriage return. After this the simulation will start, showing the number of (theta and or phi) loop as it is completed on the monitor. At the end of the calculation, there will be indicated “completed” on the monitor. The resulting figures (The Matlab --.fig file and the PDF --.pdf file for the time-domain (TD) and Fourier-transform (FT) plots); as well, the numerical values of the TD signal for the various $t_2, t_2$ values and the input parameters (.txt files) will be included in the “Figures” folder.

Please note that before running the Matlab code, one must remove the “XXX” present in the beginning of each such line in the source code and join it to the end of the line just above it. Furthermore, some lines in the source code have “…” (as indicated by the diagnostics), then remove the “…” at the end of each of these lines and join at its end the line next to it. The Matlab code should now work.

Appendix A. Main function(name: Single&Powder_Main):

% 2d-ELDOR (Two-dimensional electron-electron % double resonance) Pulse EPR Matlabssimulation % Lin Li, Department of Physics Concordia, % 20 18-05-03

%function Single&Powder_Main()
fclose('all'); % Close all open files
% Input data
myIO = IO_class();
dataFileName = 'Data\Single&Powder_Data.txt';
myIO.ReadData(dataFileName);

prompt = 'Edit input parameters in data/
(XXX) Single&Powder_Data.txt, Y/N? ';
editParameters =...
myIO.InputYes('EditParameters', prompt);

fs1 = myIO.GetData('fs1');
fs2 = myIO.GetData('fs2');

%Only plot the backuped time-domain data
plotExistData = myIO.GetData('plotExistData');
plot_class=Plot_class(fs1, fs2,plotExistData);

prompt = 'Please enter number of pulses, 2(for
(XXX) SECSY) or 3(for Echo-ELDOR): ';
inPulses = myIO.InputInt('pulses', prompt,...
min, max);

% rotateDegree: rotate the plot a z-axis,
%unit:degree, not radian
rotateDegree = myIO.GetData('rotateDegree');
prompt = 'Please enter number of thetas (XXX) (1 - 720): ';
min = 1;
max = 720;

nThetas = myIO.InputInt('nThetas', prompt, min, max);

prompt = 'Please enter number of phis (XXX) (1 - 720): ';

nPhis = myIO.InputInt('nPhis', prompt, min, max);

withRelaxation = myIO.GetData('withRelaxation');

if (nThetas == 1) 
    thetas(nThetas) = thetaMin;
else 
   for i = 1:nThetas 
       thetas(i) = acos(cosThetaMin +...
          (i-1)*(cosThetaMax - cosThetaMin)/...
          (nThetas-1));
   end
end

phiMin = myIO.GetData('phiMin');
phiMax = myIO.GetData('phiMax');

if (nPhis == 1) 
    phis(nPhis) = phiMax;
else 
   for i = 1: nPhis 
       phis(i) = phiMin +...
          (i-1)*(phiMax - phiMin)/(nPhis-1);
   end
end

withGaussian = myIO.GetData('withGaussian');

if ((nThetas == 1) && (nPhis == 1))
    % prompt = 'With relaxation, Y/N: ';
    % withRelaxation = myIO.InputYes('Relaxation', prompt);
end

thetaMin = myIO.GetData('thetaMin');
thetaMax = myIO.GetData('thetaMax');

if (nThetas == 1)
    thetas(nThetas) = thetaMin;
else
   for i = 1 : nThetas
%In unit sphere in spherical coordinates
    thetas(i) = acos(cosThetaMin +...
       (i-1)*(cosThetaMax - cosThetaMin)/...
       (nThetas-1));
   end
end

prompt = 'Please enter the rotate angle (XXX) theta (0 - 180): ';

Theta = myIO.InputInt('Theta', prompt, min, max);

prompt = 'Please enter the rotate angle phi (XXX) (0 - 360): ';

Theta = myIO.InputInt('Theta', prompt, min, max);
%GetData('rotateAngle');
Phi = myIO.InputInt('Phi', prompt, min, max);
end

if ((nThetas == 1) && (nPhis!= 1))
    prompt = 'Please enter the rotate angle
(XXX)theta (0 - 180): ';
    min = 0;
    max = 180;
    %rotationNote = myIO.GetData('rotateAngle');
    Theta = myIO.InputInt('Theta', prompt,...
        min, max);
end

%rotationAngle = myIO.GetData('rotateAngle');
Theta = myIO.InputInt('Theta', prompt,...
        min, max);
end

if ((nThetas!= 1) && (nPhis == 1))
    % prompt = 'Please choose the rotation axis
%for the goniometer, 1(x-axis), 2(y-axis) or
%3(z-axis)): ';
    % min = 1;
    % max = 3;
    %rotationAxis = %myIO.
    %GetData('rotationAxis');
    % rotationAxis = myIO.InputInt('the rotation
%axis', prompt, min, max);
end

if (0<=Theta<pi/2) % x-axis
    if(0<=Phi<pi/2 || pi<=Phi<3*pi/2)
        thetas(1) = -Theta * pi/180.0d0;
        phis(1) = -Phi;
        rotationNote = strcat(' (0, ',...
            num2str(-Theta), ',', num2str(-Phi),')');
    else  % y-axis
        thetas(1) = Theta * pi/180.0d0;
        phis(1) = pi-Phi*pi/180;
        rotationNote = strcat(' (0, ',...
            num2str(Theta), ',', num2str(pi-Phi),')');
    end
else
    if(0<=Phi<pi/2 || pi<=Phi<3*pi/2)
        thetas(1) = Theta * pi/180.0d0;
        phis(1) = pi-Phi*pi/180;
        rotationNote = strcat(' (0, ',...
            num2str(Theta), ',', num2str(pi-Phi),')');
    else  % y-axis
        thetas(1) = -Theta * pi/180.0d0;
        phis(1) = -Phi;
        rotationNote = strcat(' (0, ',...
            num2str(-Theta), ',', num2str(-Phi),')');
    end
end

if (0<=Phi<pi/2 || pi<=Phi<3*pi/2)
    thetas(1) = Theta * pi/180.0d0;
    phis(1) = pi-Phi*pi/180;
    rotationNote = strcat(' (0, ',...
        num2str(Theta), ',', num2str(pi-Phi),')');
else  % y-axis
    thetas(1) = -Theta * pi/180.0d0;
    phis(1) = -Phi;
    rotationNote = strcat(' (0, ',...
        num2str(-Theta), ',', num2str(-Phi),')');
end

%if(withGaussian)
%   rotationNote = %strcat(rotationNote, ', with Gaussian');
%else
%   rotationNote = %strcat(rotationNote, ', without Gaussian');
%end

prompt = 'Please enter the rotate angle phi
(XXX)(0 - 360): ';
min = 0;
max = 360;
%rotationAngle = %myIO.
%GetData('rotateAngle');
Phi = myIO.InputInt('Phi', prompt, min, max);
% end
With Gaussian, Y/N: 

withGaussian = ...

myIO.InputYes('WithGaussian', prompt);

With Relaxation During the (XXX) Pulses?, Y/N: 

withRelaxationPulse = ...

myIO.InputYes('WithRelaxationPulse', prompt);

With H0 During the Pulses?, Y/N: 

withH0Pulse = ...

myIO.InputYes('WithRelaxationPulse', prompt);

if(~exist('Figures', 'dir'))
   mkdir('Figures');
end

prompt='Please enter the file name: ';
signalFile=['figures\ input(prompt,'s')]
%signalFile = 'figures\signalEldorp'
signalFile = myIO.OutputFileName(...
signalFile, nThetas, nPhis,...
withRelaxation, '.dat');

B0 = myIO.GetData('B0'); % unit: Gauss
w_n = myIO.GetData('w_n'); % w_n the nuclear %Larmor frequency for the nucleus unit: MHz
g_tensor = myIO.GetData('g_tensor');
hyperfine_tensor = ...
myIO.GetData('hyperfine_tensor');

hamiltonianFactor=HamiltonianFactor_class(...
B0, w_n, g_tensor, hyperfine_tensor);

% T2e Off-diagonal electron spin-spin relaxation, unit: microsecond
t2e = myIO.GetData('t2e');

% T2n Off-diagonal spin-nuclear relaxation, %unit: microsecond
t2n = myIO.GetData('t2n');

% Diagonal Lattice %induced electron-spin flip relaxation rates, %unit: 1/ microsecond
we = myIO.GetData('we');

% Diagonal Lattice %induced nuclear-spin flip relaxation rates, %unit: 1/ microsecond
wn = myIO.GetData('wn');

% Diagonal Cross %induced Cross relaxation, unit: 1/ microsecond
wx = myIO.GetData('wx');

% Diagonal Diagonal %Cross relaxation, unit: 1/ microsecond
wy = myIO.GetData('wy');

% Heisenber spin %exchange, unit: 1/ microsecond
whe = myIO.GetData('whe');

% round off relaxation = Relaxation_class(t2e, t2n, we,...
wn, wx, wy, whe);

random = ...
Hamiltonian_class(hamiltonianFactor,...
relaxation, withRelaxation, rndoff);

density = Density_class(hamiltonian);

% pulse time of a pi/2 pulse, unit: Microsecond
pulsetime = myIO.GetData('pulsetime');

%p (phase, tipAngle) pair
pulsetype1 = myIO.GetData('pulsetype1');

%p (phase, tipAngle) pair
pulsetype2 = myIO.GetData('pulsetype2');
pulse = Pulse_class(hamiltonian,...
relaxation, density, pulsetime, pulsetype1,...
pulsetype2, withrelaxationpulse, withh0pulse);

fED = FED_class(hamiltonian, density);

pathway = Pathway_class(density);

%signalType = 1; % 1 - S+, 2 - S-
signalType = myIO.GetData('signalType');

delta = myIO.GetData('Delta');

% Gaussian
%inhomogeneous broadening effect, unit: MHz

% In the coherence pathway, coherence order
%for 2 pulses
spinPathway2 = myIO.GetData('spinPathway2');

% In the coherence pathway, coherence order
%for 3 pulses
spinPathway3 = myIO.GetData('spinPathway3');

signal = Signal_class(hamiltonian,...
density, pulse, fED, pathway, signalType,...
plot_class.Tv1, plot_class.Tv2, thetas,...
phis, withGaussian, delta, signalFile);

disp('Please wait...');

tic % starts a stopwatch timer to measure
%performance
if nPulses == 2
    pulses2 = {pulsetype1, pulsetype1};
signal.Output2Pulses(pulses2, spinPathway2);
endif

if ((nThetas == 1) && (nPhis == 1))
    plotTitle = myIO.PlotTitle2('SECSY',...
rotationNote);
else
    plotTitle = myIO.PlotTitle('SECSY',...
nThetas, nPhis, withRelaxation);
end
elseif nPulses == 3
    t = myIO.GetData('FEDTime'); %5, 20, 40,
    %60; %unit: microsecond
    pulses3 = {pulsetype1, pulsetype1, pulsetype1};
signal.Output3Pulses(pulses3, t, spinPathway3);
    if ((nThetas == 1) && (nPhis == 1))
        plotTitle = myIO.PlotTitle2('Echo-ELDOR', ...
         rotationNote);
    else
        plotTitle = myIO.PlotTitle('Echo-ELDOR',...
         nThetas, nPhis, withRelaxation);
    end
else
    error('Error: Pulse number must be 2 or 3, not
    (XXX)%d.', nPulses);
end

plot_class.SignalToFFT(signalFile);
plot_class.Plot(plotTitle, withTitle, ...
rotateDegree, withTranspose, plotAll);

% Output messages to screen
fprintf('nPulses = %d
',nPulses);
fprintf('(nThetas,nPhis) = (%d,
(XXX)%d\n',nThetas,nPhis);
end

if(withRelaxation)
fprintf('With relaxation: %s\n','yes');
else
fprintf('With relaxation: %s\n', 'no');
end
fprintf('Output data file name: %s\n', ...
signalFile);

elapsedTime = toc; % reads the elapsed time
% from the stopwatch timer started by the tic
timeStr = datestr(elapsedTime/(24*60*60),...
'DD:HH:MM:SS.FFF');
fprintf('Total time: %s\n', timeStr);
myIO.RecordItem('Total time', timeStr);
myIO.RecordItem('End', datestr(now));
myIO.SaveParameters();
disp('Complete!');

%end %function ELDORPowder_Main()

Appendix B. Hamiltonian factor (name: HamiltonianFactor_class):

% Calculate spin Hamiltonian parameters, LPF %Eqs.2 and 4
classdef HamiltonianFactor_class< handle
    properties
        B0; % static magnetic field, unit: Gauss
        Wn; % the nuclear Larmor frequency for the %nucleus, unit: MHz
        G_tensor; % g_tensor(3)
        Hyperfine_tensor; % hyperfine_tensor(3)
    end
    methods
        function this =...
            HamiltonianFactor_class(B0, wn, g_tensor,...
                hyperfine_tensor)
                % B0 !static magnetic field, unit: Gauss
            % wn !the nuclear Larmor frequency for the %nucleus, unit: MHz
                % g_tensor(3)
                % hyperfine_tensor(3) !hyperfine tensor, unit: %Gauss
            this.B0 = B0;
            this.Wn = wn;
            this.G_tensor = g_tensor;
            this.Hyperfine_tensor = hyperfine_tensor;
        end

        function Init(this, euler_angles)
            % euler_angles(3) !euler angles of the rotating %frame, unit: rad
            bohr = 9.27400968D-28; %Bohr magneton:
                %\mu_B=9.27400968\times10^{-28} J*Gauss^{-1}
            planck = 6.62606957D-34; %Planck constant:
                %h=6.62606957\times10^{-34} J*S
            planck = planck/(2*pi); % reduced Planck
                %constant J*S/rad
        end

% end classdef HamiltonianFactor_class
%gyromagneticRatio = 2.802495266D0; %Electron
%gyromagnetic ratio: r_e=2.8024952 %66(62)
%MHz*Gauss^-1

wn = this.Wn;

hyperfine_tensor = this.Hyperfine_tensor;

f0 = (bohr * this.B0 / planck) * 1.0D-6; %
%uint: MHz*rad

this.F0 = f0;

gAverage = (1.0/3.0)*(g_tensor(1) +
+ hyperfine_tensor(2) + hyperfine_tensor(3)) * f0;

this.GAverage = gAverage;

this.W0 = gAverage; % the electron Larmor
%frequency for the nucleus, uint: MHz

aAverage = (1.0/3.0)*(hyperfine_tensor(1)...
+ hyperfine_tensor(2) + hyperfine_tensor(3));
%unit: Gauss

this.AAverage = aAverage;

f=(2.0/3.0)*(g_tensor(3)-0.5*(g_tensor(1)... + g_tensor(2))) * f0;

this.F = f; % uint: MHz

% For calculation of orientation-dependent spin
%relaxation matrix

% alpha = euler_angles(1);
  beta = euler_angles(2);
  gamma = euler_angles(3);

% alpha = euler_angles(1);
  beta = euler_angles(2);
  gamma = euler_angles(3);

% gyromagneticRatio = 2.802495266D0; %Electron
%gyromagnetic ratio: r_e=2.8024952 %66(62)
%MHz*Gauss^-1

wn = this.Wn;

hyperfine_tensor = this.Hyperfine_tensor;

f0 = (bohr * this.B0 / planck) * 1.0D-6; %
%uint: MHz*rad

this.F0 = f0;

gAverage = (1.0/3.0)*(g_tensor(1) +
+ hyperfine_tensor(2) + hyperfine_tensor(3)) * f0;

this.GAverage = gAverage;

this.W0 = gAverage; % the electron Larmor
%frequency for the nucleus, uint: MHz

aAverage = (1.0/3.0)*(hyperfine_tensor(1)...
+ hyperfine_tensor(2) + hyperfine_tensor(3));
%unit: Gauss

this.AAverage = aAverage;

f=(2.0/3.0)*(g_tensor(3)-0.5*(g_tensor(1)... + g_tensor(2))) * f0;

this.F = f; % uint: MHz

d = (2.0/3.0)*(hyperfine_tensor(3)-0.5...
*(hyperfine_tensor(1)+hyperfine_tensor(2)));

this.D = d;

f2 = 0.5*(g_tensor(1) - g_tensor(2)) * f0;

this.F2 = f2;

d2 = 0.5*(hyperfine_tensor(1)-...
hyperfine_tensor(2));

this.D2 = d2;
% this.C2 = 1 - (this.A/2 - wn)/wa; \% Modified by H R Salahi
% this.C2 = -sqrt(this.C2/2);
% this.C3 = 1 + (this.A/2 + wn)/wb;
% this.C3 = sqrt(this.C3/2);
% this.C4 = 1 - (this.A/2 + wn)/wb;
% this.C4 = -sqrt(this.C4/2);
% this.M1 = 1 - (wn^2 - (this.A/2)^2 - (abs(this.B1/2)^2))/(wa*wb);
% this.M1 = sqrt(this.M1/2);
% this.M2 = 1 + (wn^2 - (this.A/2)^2 - (abs(this.B1/2)^2))/(wa*wb);
% this.M2 = sqrt(this.M2/2);
% LPF' Eq.(A3)
% this.Ea = this.C/2 + wa;
% this.Eb = this.C/2 - wa;
% this.Ec = -this.C/2 + wb;
% this.Ed = -this.C/2 - wb;
end
% HamiltonianFactor_class constructor
end %methods
end %classdef HamiltonianFactor_class

Appendix C. Hamiltonian operator (name: Hamiltonian_class):

% 2d-ELDOR (Two-dimensional electron-electron double resonance) Pulse EPR Matlab simulation
% Lin Li, Department of Physics Concordia, 2018-05-03

classdef Hamiltonian_class< handle
properties
HamiltonianFactor;
Relaxation;
end %properties
methods
% constructor
function this = Hamiltonian_class(...
hamiltonianFactor, relaxation,...
withRelaxation, rndoff)
this.HamiltonianFactor = hamiltonianFactor;
this.Relaxation = relaxation;
this.WithRelaxation = withRelaxation;
this.Rndoff = rndoff;
end

function Init(this, euler_angles)
% HamiltonianH = zeros(4,4);
hamiltonianFactor = this.HamiltonianFactor;
hamiltonFactor.Init(euler_angles);
C = hamiltonianFactor.C; % C, A, B1, B2: uint: MHz
A = hamiltonianFactor.A;
B1 = hamiltonianFactor.B1;
B2 = hamiltonianFactor.B2;
wn = hamiltonianFactor.Wn; % thenuclear Larmor %frequency for the nucleus, uint: MHz
 rndoff = this.Rndoff;

% Initialize Hamiltonian operator matrix in %Hilbert space
sz = eye(2); % sz of 1/2 spin
sz(2,2) = -1;
sz = 0.5 * sz;
Iz = sz;
Iplus = zeros(2,2); % s+ of 1/2 spin
Iplus(1,2) = 1;
Iminus = zeros(2,2); % s- of 1/2 spin
Iminus(2,1) = 1;
hamiltonianSzIzH = C * kron(sz, eye(2)) - ...
wn * kron(eye(2), Iz) + A * kron(sz, Iz) + 0.5 ...
* B1 * kron(sz, Iplus) + 0.5 * B2 * kron(sz, Iminus);
this.HamiltonianSzIzH = hamiltonianSzIzH;

% Diagonalize the Hamiltonian operator matrix %in Hilbert space
[V, D, W] = eig(A);
% D: diagonal matrix of generalized eigenvalues
% V: Right eigenvectors: A*V = V*D
% W: Left eigenvectors: W'*A = D*W
[V, D] = eig(hamiltonianSzIzH);
%Swap ascending order to descending order
eigenvalueH = diag(D);
eigenvalueH([1,3], :) = eigenvalueH([3,1], :);
eigenvalueH([2,3], :) = eigenvalueH([3,2], :);
this.EigenvalueH = eigenvalueH;
rightEigenvectorH = V;
rightEigenvectorH(:, [1, 3]) = ...
-1 * rightEigenvectorH(:, [3, 1]);
rightEigenvectorH(:, [2, 3]) = ...
rightEigenvectorH(:, [3, 2]);
this.RightEigenvectorH = rightEigenvectorH;

% Initialize Liouville superoperator matrix in %Liouville space
eigenvalueH = diag(this.EigenvalueH);
n = size(this.EigenvalueH, 1);
this.HamiltonianH0L = kron(eye(n), ...
eigenvalueH) = kron(eigenvalueH, eye(n));
h0 = SzIzToH0BasisH...
{this, this.HamiltonianSzIzH};
relaxation0 = this.Relaxation.Relaxation;
liouville0 = kron(eye(4), h0) - ...
 kron(transpose(h0), eye(4));
liouville = round((1i * liouville0 + relaxation0) ...
*rndoff)/rndoff;
[V1, D1] = eig(liouville, 'vector');
rightEigenvectorL = V1;
rightEigenvectorH(:, [1, 3]) = ...
rightEigenvectorH(:, [3, 1]);
rightEigenvectorH(:, [2, 3]) = ...
rightEigenvectorH(:, [4, 3]);
this.RightEigenvectorL = rightEigenvectorL;
liouvilleH = H0ToNormalBasis(this, liouville);
this.LiouvilleH0 = liouvilleH;
function [A] = SzIzToH0BasisH(this, operatorH)
A = ctranspose(this.RightEigenvectorH) * operatorH * this.RightEigenvectorH;
end

function [A] = H0ToSzIzBasisH (this, operatorH)
A = this.RightEigenvectorH * operatorH * ctranspose(this.RightEigenvectorH);
end

function [A] = H0ToNormalBasis (this, operatorH)
A = ctranspose(this.RightEigenvectorL) * operatorH * this.RightEigenvectorL;
end

function [A] = NormalToH0Basis (this, operatorL)
A = (this.RightEigenvectorL) * operatorL * ctranspose(this.RightEigenvectorL);
end

function [A] = ToH0BasisDensity (this, operator)
A = ctranspose(this.RightEigenvectorH) * operator;
end

function [A] = ToSzIzBasisDensity (this, operator)
A = this.RightEigenvectorH * operator;
end

function Print(this)
disp('HamiltonianH:');
disp(this.HamiltonianSzIzH); % Hamiltonian %operator in Hilbert space and |SzIz> basis
disp('EigenvalueH:');
disp(this.EigenvalueH);
end % function Print

classdef Hamiltonian_class

properties
Hamiltonian;
EquilibriumSzIzH; % Equilibrium density
end %classdefHamiltonian_class

Appendix D. Density matrix (name: Density_class):
% 2d-ELDOR (Two-dimensional electron-electron double resonance) Pulse EPR Matlab simulation
% Lin Li, Department of Physics Concordia, %2018-05-03
classdef Density_class< handle
properties
Hamiltonian;
EquilibriumSzIzH; % Equilibrium density
% operator in SzIz basis in Hilbert space

EquilibriumHOH; % Equilibrium density operator
% in eigenvector basis of H in Hilbert space
EquilibriumH0L; % Equilibrium density operator
% in eigenvector basis of H in Liouville space
DensitySzIzL; % Current density operator in %SzIz basis in Liouville space

%properties

methods

% Suppose, at initial state, the spin is in %z-axis
function this = Density_class(hamiltonian)

this.Hamiltonian = hamiltonian;

equilibriumH = zeros(4,4);
% Equilibrium density operator in Hilbert space %and |SzIz> basis
e0 = complex(0.50e0, 0.0e0);
equilibriumH(1,1) = e0;
equilibriumH(2,2) = e0;
equilibriumH(3,3) = -e0;
equilibriumH(4,4) = -e0;

this.EquilibriumH0H = this.Hamiltonian... .SzIzToH0BasisH(this.EquilibriumSzIzH);
this.EquilibriumH0L = this.EquilibriumHOH(:,:,);

this.DensitySzIzL = this.EquilibriumSzIzH(:,:,);

end % properties

% Suppose, at initial state, the spin is in %z-axis
function this = Density_class(hamiltonian)

this.Hamiltonian = hamiltonian;

equilibriumH = zeros(4,4);
% Equilibrium density operator in Hilbert space %and |SzIz> basis

end % function GetDensityMat

function SetDensityInSzIzBasisHToL(this...) .densityInSzIzBasisH)

this.DensitySzIzL = densityInSzIzBasisH(:,:,);

end

function[A] = GetDensityH0BasisL(this)
A = this.GetDensityInSzIzBasisH();
A = this.Hamiltonian.SzIzToH0BasisH(A);
A = A(:,:,);
end % function GetDensityMat

function[A] = GetDensitySzIzBasisL(this)
A = this.GetDensityInSzIzBasisH();
A = this.Hamiltonian.H0ToSzIzBasisH(A);
A = A(:,:,);
end % function GetDensityMat

end % function GetDensityMat
function SetDensityInH0ToSzIzBasisL(...
this, densityInH0BasisL)

    B = densityInH0BasisL;
    n = size(B, 1);
    m = sqrt(real(n));
    densityInH0BasisH = reshape(B, [m, m]);

    densityInSzIzBasisH = this.Hamiltonian.H0ToSzIzBasisH(densityInH0BasisH);
    this.DensitySzIzL = densityInSzIzBasisH(:);
    %reshape(densityH, [4*4, 1]);
end % function SetDensityInH0ToSzIzBasisL

function ToNormalBasis(this)
    B = this.DensitySzIzL;
    this.DensitySzIzL = this.Hamiltonian.H0ToNormalBasisDensity(B);
end % function ToNormalBasis

function ToHBasis(this)
    B = this.DensitySzIzL;
    this.DensitySzIzL = this.Hamiltonian.NormalToH0BasisDensity(B);
end % function ToHBasis

function Print(~)
    disp('Density_class');
end % function Print

end % classdefDensity_class

% 2d-ELDOR (Two-dimensional electron-electron double resonance) Pulse EPR Matlab simulation
% Lin Li, Department of Physics Concordia, %2018-05-03

classdef FED_class< handle
properties
    Hamiltonian;
    Density;
end %properties

methods
    % constructor
    function this = FED_class(hamiltonian, density)
        this.Density = density;
        this.Hamiltonian = hamiltonian;
    end % FED_class constructor

    % Calculate it in the eigenvector %basis of H in Liouville space
    function DensityEvolve(this, t)
        % p0 = this.Density.EquilibriumH0L;
        % p = this.Density.GetDensityH0BasisL();
        % p = this.Density.DensitySzIzL;
        fEDOperator = this.CalFEDOperator(t);
        density = fEDOperator * (p) ;
        this.Density.DensitySzIzL=density;
    end % function DensityEvolve

end %classdefFED_class

Appendix E. Free evolution operator(name: FED_class):
Free Evolution Decay (FED)
Pulsetype2;

end % DensityEvolve

% Calculate FED superoperator in the
%eigenvector basis of H in Liouville space
function [fEDOperator] =...
CalFEDOperator(this, t)
fEDOperator =...
expm(t*this.Hamiltonian.LiouvilleH0);
end % CalFEDOperator

function Print(~)
disp('FED_class');
end % function Print

end %method
end %classdefFED_class

Appendix F. Pulse operator (name: Pulse_class):

% 2d-ELDOR (Two-dimensional electron-electron
%double resonance) Pulse EPR Matlab simulation
% Lin Li, Department of Physics Concordia,
%2018-05-03

classdef Pulse_class< handle

    properties
        Hamiltonian;
        Density;

    end % properties

    Pulsetime0; % pi/2 pulse( -5 ns), unit:
    Pulsetype1;

end % Pulse_class constructor

function this = Pulse_class(hamiltonian,...
    relaxation,density,pulsetime0, pulsetype1,...
    pulsetype2, withrelaxationpulse,withh0pulse)
this.Hamiltonian = hamiltonian;
this.Density = density;
this.Pulsetime0 = pulsetime0;
this.Pulsetype1 = pulsetype1;
this.Pulsetype2 = pulsetype2;
this.Relaxation=relaxation;
this.Withrelaxationpulse=withrelaxationpulse;
this.Withh0pulse=withh0pulse;
end % Pulse_class constructor
% Calculate pulse superoperator in Liouville space and |SzIz> basis
function [pulseOperator] = CalPulseOperator(this, pulsetype)

phase = pulsetype(1);
tipAngle = pulsetype(2);

% In |SzIz> basis
s_plus = zeros(2);
s_plus(1,2) = 1;
s_plus = kron(s_plus, eye(2));
s_minus = zeros(2);
s_minus(2,1) = 1;
s_minus = kron(s_minus, eye(2));

% t*H1: timepulse = (pulse time) x (%irradiating microwave pulse), in Hilbert space and |SzIz> basis
timeH0 = this.Pulsetime0*...
timeH = this.Hamiltonian.SzIzToH0BasisH...

% H0 << H1
%H0ToNormalBasis(pulseOperator);

%p = expm(-i*timeH1); % exp(-i*t*H1)
pulseOperator = kron(ctranspose(p), p); % for H0 << H1
pulseOperator = this.Hamiltonian...
H0ToNormalBasis(pulseOperator);
end % function CalPulseOperator

% t*H1: timepulse = (pulse time) x (%irradiating microwave pulse), in Hilbert space and |SzIz> basis
timeH1 = 0.5*tipAngle*(exp(-1i*phase)*s_plus...
+ exp(1i*phase)*s_minus);
timeH1 = kron(timeH1, eye(2));
timeH0 = this.Pulsetime0*...
this.Hamiltonian.HamiltonianSzIzH;
timeH = this.Hamiltonian.SzIzToH0BasisH...
(timeH);
timeH0 = this.Hamiltonian.SzIzToH0BasisH(timeH0);

if (tipAngle == this.Pulsetype1(2))
this.Density.DensitySzIzL = ...
this.PulseOperator1*this.Density.DensitySzIzL;
elseif (tipAngle == this.Pulsetype2(2))
this.Density.DensitySzIzL = ...
this.PulseOperator2*this.Density.DensitySzIzL;
else
error('Error: \n%d is not a valid pulse',...
pulsetype);
end %function DensityEvolve

% Calculate pulse superoperator in Liouville space and |SzIz> basis
kron(transpose(timeH), eye(4));
pulseOperator = expm(-1i*p);
if(this.Withrelaxationpulse)
pulseOperator = round(expm(-1i*p-...
R*tp)*10e10)/10e10;
end

p = kron(eye(4), timeH) -...
Appendix G. Relaxation matrix (name: Relaxation_class):

% 2d-ELDOR (Two-dimensional electron-electron double resonance) Pulse EPR Matlab simulation
% Lin Li, Department of Physics Concordia, 2018-05-03
% Modified by H R Salahi

classdef Relaxation_class< handle
    properties
        Relaxation = zeros(16, 16); % Relaxation operator in Liouville space and |SzIz> basis
        T2e; % Off-diagonal electron spin-spin %relaxation, unit: microsecond
        T2n; % Off-diagonal nuclear spin-spin %relaxation, unit: microsecond
        We ; % Diagonal electron spin-spin %relaxation, unit: microsecond
        Wn ; % Diagonal nuclear spin-spin %relaxation, unit: microsecond
        Wx ; % Diagonal electron nuclear spin %relaxation, unit: microsecond
        Wy ; % Diagonal electron nuclear spin %relaxation, unit: microsecond
        Whe; % Heisenberg exchange relaxation
    end %properties

    methods

        function this = Relaxation_class(t2e, t2n,...
            we, wn, wx, wy, whe)
            relaxation = zeros(16, 16); % Relaxation operator in Liouville space and W0 basis
            this.T2e = t2e;
            this.T2n = t2n;
            this.We = we;
            this.Wn = wn
            this.Wx = wx
            this.Wy = wy;
            this.Whe = whe;
            RT2e = -1.0 / t2e;
            RT2n = -1.0 / t2n;
            rt_ba = RT2n;
            rt_ca = RT2e;
            rt_da = RT2e;
            rt_ab = RT2n;
            rt_cb = RT2e;
            rt_db = RT2e;
            rt_ac = RT2e;
        end

end %classdef Relaxation_class
rt_bc = RT2e;
rt_dc = RT2n;
rt_ad = RT2e;
rt_bd = RT2e;
rt_cd = RT2n;
w_ab = wn;
w_ba = wn;
w_cd = wn;
w_da = w_da - whe;
w_db = w_db + whe;
w_dc = w_dc + whe;
rt_ba = RT2n;
rt_ca = RT2n;
rt_da = RT2n;
rt_db = RT2n;
w_ac = we;
w_ca = we;
w_bd = we;
w_db = we;
w_ac = w_ac + whe;
w_ca = w_ca + whe;
w_bd = w_bd + whe;
w_db = w_db + whe;
w_ad = w_ad - whe;
w_ba = w_ba + whe;
w_bc = w_bc - whe;
w_bd = w_bd + whe;
w_ca = w_ca + whe;
w_cb = w_cb - whe;
w_cd = w_cd + whe;
w_da = w_da - whe;
w_db = w_db + whe;
w_dc = w_dc + whe;
w_ac = w_ac + whe;
w_ca = w_ca + whe;
w_ad = w_ad - whe;

relaxation(1,1) = w_aa;
relaxation(2,2) = rt_ba;
relaxation(3,3) = rt_ca;
relaxation(4,4) = rt_da;
relaxation(5,5) = rt_ab;
relaxation(6,6) = w_bb;
relaxation(7,7) = rt_cb;
relaxation(8,8) = rt_db;
relaxation(9,9) = rt_ac;
relaxation(10,10) = rt_bc;
relaxation(11,11) = w_cc;
relaxation(12,12) = rt_dc;
relaxation(13,13) = rt_ad;
relaxation(14,14) = rt_bd;
relaxation(15,15) = rt_cd;
relaxation(16,16) = w_dd;

relaxation(1,6) = w_ab;
relaxation(1,11) = w_ac;
relaxation(1,16) = w_ad;
relaxation(6,1) = w_ba;
relaxation(6,6) = w_bb;
relaxation(6,11) = w_bc;
relaxation(6,16) = \text{w}_{bd};

relaxation(11,1) = \text{w}_{ca};

relaxation(11,6) = \text{w}_{cb};

%relaxation(11,11) = \text{w}_{cc};

relaxation(11,16) = \text{w}_{cd};

relaxation(16,1) = \text{w}_{da};

relaxation(16,6) = \text{w}_{db};

relaxation(16,11) = \text{w}_{dc};

%relaxation(16,16) = \text{w}_{dd};

this.Relaxation = relaxation;

end % Relaxation_class constructor this

function Print(this)

disp('Relaxation_class');
disp(this.Relaxation); % Relaxation operator

end % function Print

disp('Relaxation matrix in Liouville space:');
disp(this.Relaxation); % Relaxation operator

end % methods

end %classdefRelaxation_class

Appendix H. Pathway operator (name: Pathway_class):

classdef Pathway_class< handle

properties

Density;
P1;
P0;
P_1;

end %properties

methods

% constructor

function this = Pathway_class(density)

this.Density = density;

splus = zeros(2,2); % s+ of 1/2 spin
splus(1,2) = 1;
sminus = zeros(2,2); % s- of 1/2 spin
sminus(2,1) = 1;

this.P1 = kron(splus, ones(2));
this.P0 = kron(eye(2), ones(2));
this.P_1 = kron(sminus, ones(2));

end %classdef Pathway_class

function SelectPath(this, spinPathway)

% In SzIz basis

disp('2d-ELDOR (Two-dimensional electron-electron double resonance) Pulse EPR Matlab simulation % Lin Li, Department of Physics Concordia, 2018-05-03 % Coherence Pathway')

end % function SelectPath

end % methods

end %classdefPathway_class
density =... %2018-05-03
this.Density.GetDensityInSzIzBasisH(); %Modified by H R Salahi
classdef Signal_class< handle

switch spinPathway
case 1 % p=1, S+; set 1st,3rd and 4th
%quadrants to zero
 density = (this.P1).* density;
case 0 % p=0
 density = (this.P0).* density;
case -1 % p=-1
 density = (this.P_1).* density;
case -11 % p=-1 or 1
 density = (this.P1 + this.P_1).* density;
 otherwise
 error('Error. \nNo such pathway %d.',...
spinPathway)
 end %switch path

this.Density.SetDensityInSzIzBasisHToL...
(density);
end % SelectPath

end % methods
end % Pathway_class< handle

Appendix I. Calculation of the time domain signals according to the
pulse sequence and the pathway defined in (name: Signal_class):

function this = Signal_class(hamiltonian,...
density, pulse, fED, pathway, ... %2d-ELDOR (Two-dimensional electron-electron
double resonance) Pulse EPR Matlab simulation
% Lin Li, Department of Physics Concordia,
withGaussian, delta, signalFile)
end

this.Hamiltonian = hamiltonian;
this.Density = density;
this.Pulse = pulse;
this.FED = fED;
this.Pathway = pathway;

this.SignalType = signalType;

function Init(this, i, j, m, n, euler_angle)

mn = m*n;
ij = (i-1)*n+j;

if((mn< 10) ||(ij == 1) ||(ij == mn) ||...
((mn< 50) && (mod(ij, 5) == 0)) || ...
(mod(ij, 10) == 0))
    fprintf('Calculating: (theta, phi) = (%d/%d, %d/%d), %d of %d
', i, m, j, n, ij, mn);
end

this.Hamiltonian.Init(euler_angle);
this.Density.Init();
this.Pulse.Init();
%this.FED.Init();
end

function Output2Pulses(this, pulses,...
spinPathway)

thetas = this.Thetas;
phis = this.Phis;

m = length(thetas);
n = length(phis);

thetas = this.Thetas;
phis = this.Phis;

dlmwrite(this.SignalFile, signals);
function signals = Cal2Pulses(this, pulses,...
spinPathway, signals)

% pulsepar(2): (phase, tipAngle) pairs

% pulsepar(2): (phase, tipAngle) pairs

tv1 = this.Tv1;
tv2 = this.Tv2;
n1 = length(tv1);
n2 = length(tv2);

this.Density.GetDensityH0BasisL();
this.Density.ToNormalBasis();
this.Pulse.DensityEvolve(pulses{1});
this.Density.ToHBasis();
this.Pathway.SelectPath(spinPathway(1));
%After pulse
this.Density.ToNormalBasis();
densityLst1 = this.Density.DensitySzIzL;
for i = 1: n1
    this.Density.DensitySzIzL = densityLst1;
    t1 = tv1(i);
    this.FED.DensityEvolve(t1);
    this.Density.ToHBasis();
    this.Density.GetDensitySzIzBasisL();
    signal = this.Measure ();
    signal = imag(signal); % real-dispersion,
    %imag-absorption, abs-Complex Magnitude
    if(this.WithGaussian)%Gaussian inhomogeneous
        %broadening effect
        delta = this.Delta;
        g = exp(-2 * (pi * delta * (t1))^2);
        signal = g*signal;
        end
    signals(i,j) = signals(i,j) + signal;
this.FED.DensityEvolve(t1);
%fprintf(fileId,'%12.5e	', signal);
end
this.Pulse.DensityEvolve(pulses{2});
%fprintf(fileId,'
');
function Output3Pulses(this, pulses, t, spinPathway)
    signals = zeros(length(this.Tv1), length(this.Tv2));
    thetas = this.Thetas;
    phis = this.Phis;
    m = length(thetas);
    n = length(phis);
    for i = 1:m
        for j = 1:n
            euler_angle = zeros(1, 3);
            euler_angle(2) = thetas(i);
            euler_angle(3) = phis(j);
            this.Init(i, j, m, n, euler_angle);
            signals = this.Cal3Pulses(pulses, t, spinPathway, signals);
            end % for j = 1:n
        end % for i = 1:m
        signals = signals / (max(max(abs(signals)))); % scale to 1
        dlmwrite(this.SignalFile, signals);
    end %function Output2Pulses
end

function signals=Cal3Pulses(this, pulses, t, spinPathway, signals)
    tv1 = this.Tv1;
    tv2 = this.Tv2;
    n1 = length(tv1);
    n2 = length(tv2);
    thetas = this.Thetas;
    phis = this.Phis;
    m = length(thetas);
    n = length(phis);
    for i = 1:m
        for j = 1:n
            euler_angle = zeros(1, 3);
            euler_angle(2) = thetas(i);
            euler_angle(3) = phis(j);
            this.Init(i, j, m, n, euler_angle);
            signals = this.Cal3Pulses(pulses, t, spinPathway, signals);
            end % for j = 1:n
        end % for i = 1:m
        t1 = tv1(i);
        for i = 1:n1
            this.Density.ToNormalBasis();
            this.FED.DensityEvolve(t1);
            this.Pulse.DensityEvolve(pulses{1});
            this.Density.ToHBasis();
            this.Pathway.SelectPath(spinPathway(1));
            this.Density.ToNormalBasis();
            densityLst1 = this.Density.DensitySzIzL;
        end % for i = 1:n1
        this.FED.DensityEvolve(t);
        this.Pulse.DensityEvolve(pulses{2});
        this.Density.ToHBasis();
        this.Pathway.SelectPath(spinPathway(2));
        this.Density.ToNormalBasis();
        this.FED.DensityEvolve(t);
    end %function Output3Pulses
end
this.FED.DensityEvolve(t1);
densityLst2 = this.Density.DensitySzIzL;
for j = 1: n2
    this.Density.DensitySzIzL = densityLst2;
    t2 = tv2(j);  % SECSY
    this.FED.DensityEvolve(t2);
    this.Density.ToHBasis();
    this.Density.GetDensitySzIzBasisL();
    signal = this.Measure();
    signal = imag(signal);  % real-dispersion, imag
    % absorption, abs-Complex Magnitude
    if(this.WithGaussian)  % Gaussian inhomogeneous
        delta = this.Delta;
        g = exp(-2 * (pi * delta * (t2))^2);
        signal = g*signal;
        end
    signals(i,j) = signals(i,j) + signal;
end
end %function Output3Pulses

function [signal] = Measure (this)
    density = ...
    this.Density.GetDensityInSzIzBasisH();
end % function Measure

function Print(this)
    disp('Signal_class');
    this.Density.Print();
    this.Pulse.Print();
    this.FED.Print();
    disp('Sc+ matrix:');
    disp(this.S_plus);
    disp('Sc- matrix:');
    disp(this.S_minus);
end % function Print
end %classdefSignal_class< handle

end %properties

methods

Appendix J. Simulated image plot (name: Plot_class):

% 2d-ELDOR (Two-dimensional electron-electron
double resonance) Pulse EPR Matlab simulation
% Lin Li, Department of Physics Concordia,
% 2018-05-03
% Modified by H R salahi

classdef Plot_class< handle

function this = Plot_class(fs1, fs2,... plotExistData)

this.Init(fs1, fs2);
end % Plot_class constructor

function Init(this, fs1, fs2)

%fs1 = 64;       % Sampling frequency
dt1 = 1/fs1;    % Sample time, unit: Microsecond
n1 = 1*fs1;     % Length of signal
tv1 = (0:n1-1)*dt1; % (-n1/2:n1/2-1)*dt1;

%fs2 = 64;       % Sampling frequency
dt2 = 1/fs2; % Sample time, unit: Microsecond
n2 = 1*fs2; % Length of signal
tv2 = (0:n2-1)*dt2;

this.Fs1 = fs1;  % Sampling frequency
this.Dt1=dt1; % Sample time, unit: Microsecond
this.N1 =  n1;   % Length of signal
this.Tv1=tv1; % Time vector, unit: Microsecond
this.Df1 = df1;  % Frequency increment

this.Fs2 = fs2;  % Sampling frequency
this.Dt2=dt2; % Sample time, unit: Microsecond
this.N2 =  n2;   % Length of signal
this.Tv2=tv2; % Time vector, unit: Microsecond
this.Df2 = df2;  % Frequency increment

this.SignalFile= SignalFile;
this.Signal2d = Signal2d;
this.Signal2d_fft = Signal2d_fft;

this.PlotExistData = PlotExistData = false;

end %classdefPlot_class< handle
this.Fv1 = fv1; % Frequency vector, unit: MHz
tv1 = this.Tv1; % Time vector
fv1 = this.Fv1;

this.Fs2 = fs2;  % Sampling frequency
this.Dt2 = dt2;  % Sample time            this.
N2 = n2;    % Length of signal
this.Tv2 = tv2;  % Time vector, unit:
this.Df2 = df2;
this.Fv2 = fv2;

end % Init this

function SignalToFFT(this, signalFile)
this.SignalFile = signalFile;
signal2d = dlmread(signalFile); % Read in data
this.Signal2d = signal2d;

signal2d_fft = fft2(signal2d);
signal2d_fft = fftshift(signal2d_fft);
this.Signal2d_fft = signal2d_fft;

end% function SignalToFFT

function SignalToFFT_Init(this, signalFile)
[fs1, fs2] = size(this.Signal2d);
this.Init(fs1, fs2);
end

function Plot(this, titleName, withTitle,...
rotateDegree, withTranspose, plotAll)
if (~exist('plotAll','var'))
plotAll = false;
end

n1 = this.N1;   % Length of signal

tv2 = this.Tv2;  % Time vector, unit:
%Microsecond
fv2 = this.Fv2;  % Frequency vector, unit: MHz

[filepath,name,~] =...
fileparts(this.SignalFile);
if isempty(filepath)
fileName = name;
else
fileName = strcat(filepath, '\', name)
end

signal2d = this.Signal2d;
signal2d = signal2d /(max(max(abs(signal2d))));
signal2d = abs(signal2d);
if withTranspose
signal2d = transpose(signal2d);
end
signal2d_fft = this.RotationZ(...
signal2d_fft, rotateDegree);

if plotAll == true
row = 2;
col = 2;
y1 = signal2d(1:n1, col);
y2 = signal2d(row, 1:n2);

figure('Name','1D ESEEM (Time-domain)''...,'Visible','On');</
plot(tv1,y1)
title('1D ESEEM (Time-domain)')
xlabel('T1 (Microsecond)')
ylabel('Intensity')

figure('Name','1D ESEEM (Time-domain)''...,'Visible','On');</
plot(tv2,y2)
title('1D ESEEM (Time-domain)')
xlabel('T2 (Microsecond)')
ylabel('Intensity')

y1_fft = fft(y1,n1);
y1_fft = fftshift(y1_fft);
figure('Name','1D ESEEM (Frequency-domain)''...,'Visible','On');</
plot(fv1,abs(y1_fft))
title('1D ESEEM (Frequency-domain)')
xlabel('F1 (MHz)')
ylabel('Intensity')

y2_fft = fft(y2,n2);
y2_fft = fftshift(y2_fft);
figure('Name','1D ESEEM (Frequency-domain)''...,'Visible','On');</
plot(fv2,abs(y2_fft))
title('1D ESEEM (Frequency-domain)')
xlabel('F2 (MHz)')
ylabel('Intensity')

plotTimeDomain = true;
if plotTimeDomain == true
hFig = figure('Name','2D(Time-domain)''...,'Visible','On');</
mesh(tv1,tv2, signal2d)
xlabel('T1 (\textmu s)') % Microsecond
ylabel('T2 (\textmu s)') % Microsecond
zlabel('Intensity')
if withTitle == true
title(['{f', titleName, '}'])
end
view(52.5, 30);% view(3) sets the default%three-dimensional view, az = –37.5, el = 30.
%colormap gray
if(~this.PlotExistData)
saveas(hFig, strcat(fileName, '_td', '.pdf'))
saveas(hFig, strcat(fileName, '_td', '.fig'));
end
end % if plotTimeDomain == true

plotFrequencyDomain = true;
if plotFrequencyDomain == true
hFig = figure('Name','2D(Frequency-domain)'...,'Visible','On');</
mesh(fv1, fv2, signal2d_fft)
xlabel('F1 (MHz)')
ylabel('F2 (MHz)')
zlabel('Intensity')
if withTitle == true
Appendix K. Input and output of data (name: IO_class):

% 2d-ELDOR (Two-dimensional electron-electron double resonance) Pulse EPR Matlab simulation
% Lin Li, Department of Physics Concordia, %2018-05-03
% Modified by H R Salahi
classdef IO_class< handle
    % IO_CLASS Summary of this class goes here
end

function [outmatrix] = RotationZ (~,...
inmatrix, degree)
    s = sign(degree);
    degree = mod(abs(degree), 360);
    switch degree
        % Special cases
        case 0
            outmatrix = inmatrix;
        case 90
            outmatrix = rot90(inmatrix, s);
        case 180
            outmatrix = rot90(inmatrix, s*2);
        case 270
            outmatrix = rot90(inmatrix, s*3);
        otherwise
            degree = s*degree;
    end %switch
    nrows = size(inmatrix, 1);
    ncols = size(inmatrix, 2);
    [nrows, ncols] = size(inmatrix);
end %function RotationZ
end %method
end %classdefPlot_class
% Detailed explanation goes here

properties

FileName;

DataMap;

Parameters;

end % properties

methods

function this = IO_class()

this.DataMap = containers.Map;

this.Parameters = {''};

end % IO_class constructor

function result = InputInt(this, name,...
prompt, nmin, nmax)

str = strtrim(input(prompt,'s'));

result = str2double(str);

while isnan(result) || (fix(result) ~= ... 
result) || (result <nmin) || (result >nmax)

str = strtrim(input(prompt,'s'));

result = str2double(str);

end

this.RecordItem(name, str);

end % function [result] = InputInt

function result = InputYes(this, name, prompt)

str = strtrim(input(prompt,'s'));

while ~(strcmpi(str,'y') || strcmpi(str,...
'yes')) || strcmpi(str,'n') || strcmpi(str,...
'no'))

str = strtrim(input(prompt,'s'));

end

result = strcmpi(str, 'y') || strcmpi(str,...
'y','yes');

this.RecordItem(name, str);

end % function InputYes

function result = OutputFileName(this, name,...
nThetas, nPhis, withRelaxation, extension)

result = strcat(name, num2str(nThetas), '_... 
\textbackslash{}_{\text{r}}');

if withRelaxation

result = strcat(result, 'r');

else

result = strcat(result, 'nr');

end

dt = datestr(now,'yyyy_mm_dd_HH_MM_SS');

this.FileName = strcat(result, dt);

result = strcat(this.FileName, extension);

end

function result = PlotTitle(this, name,...
nThetas, nPhis, withRelaxation)

result = strcat(name, ' Thetas =... 
\textbackslash{}_{\text{r}}').'\textbackslash{}_{\text{r}}');

if withRelaxation

result = strcat(result, ' Relaxation = yes,');

else

result = strcat(result, ' Relaxation = no');

end
function result = PlotTitle2(this, name, note)
    result = strcat(name, note, ' ', datestr(now, ' mmmm dd, yyyy HH:MM:SS'));
    this.RecordItem('Plot Title', result);
end

function result = GetData(this, name)
    key = strtrim(name);
    if(isempty(key))
        error('Error: 
%s is not a valid name', name);
    end
    key = upper(key);
    % if HamiltonianMap contains thr key, get the 
    % values and return
    if (isKey(this.DataMap, key))
        value = this.DataMap(key);
        this.RecordItem(name, value);
    else
        error('Error. 
No values exist for %s.', name);
    end
    switch key
result = strcat(result, ' Relaxation = no,');
    end
result = strcat(result, datestr(now, ' mmmm dd, yyyy HH:MM:SS'));
this.RecordItem('Plot Title', result);
end

function ReadData(this, fileName)
    fid = fopen(fileName);
    allData = textscan(fid, '%s','Delimiter','
');
    allData = allData{1};
    n = size(allData, 1);
    commentChar = '%';
    startChar = ':';

    for i = 1 : n
        str = strtrim(allData{i,1});
        if((length(str) < 3) || strcmp(str(1), commentChar) || strcmp(str(1), startChar))
            continue;
        end
        p = k(i,1);
        if (length(str) == p)
            continue;
        end
        name = upper(strtrim(str(1:p-1)));
        value = str(p+1:end);
        this.DataMap(name) = value;
    end
    fclose(fid);
end

function result = GetData(this, name)
    key = strtrim(name);
    if(isempty(key))
        error('Error: 
%s is not a valid name', name);
    end
    key = upper(key);
    % if HamiltonianMap contains thr key, get the 
    % values and return
    if (isKey(this.DataMap, key))
        value = this.DataMap(key);
        this.RecordItem(name, value);
    else
        error('Error. 
No values exist for %s.', name);
    end
    switch key

Citation: Sushil K. Misra and Hamid Reza Salahi (2019) A Rigorous Calculation of Pulsed EPR SECSY and Echo-ELDOR Signals: Inclusion of Static Hamiltonian and Relaxation during Pulses. J Apl Theol 3(2): 9-48. doi: https://doi.org/10.24218/jatpr.2019.20.
J Apl Theol 3(2)

Citation: Sushil K. Misra and Hamid Reza Salahi (2019) A Rigorous Calculation of Pulsed EPR SECSY and Echo-ELDOR Signals: Inclusion of Static Hamiltonian and Relaxation during Pulses. J Apl Theol 3(2): 9-48. doi: https://doi.org/10.24218/jatpr.2019.20.

case 'TITELNAME'
    result = value;
endcase

case 'USERINPUT'
    result = this.ParseYes(name, value);
endcase

case 'WITHRELAXATION'
    result = this.ParseYes(name, value);
endcase

case 'FS1'
    result = this.ParseNum(name, value, 1);
endcase

case 'FS2'
    result = this.ParseNum(name, value, 1);
endcase

case 'PLOTEXISTDATA'
    result = this.ParseYes(name, value);
endcase

case 'B0'
    result = this.ParseNum(name, value, 1);
endcase

case 'W_N'
    result = this.ParseNum(name, value, 1);
endcase

case 'G_TENSOR'
    result = this.ParseNum(name, value, 3);
endcase

case 'HYPERFINE_TENSOR'
    result = this.ParseNum(name, value, 3);
endcase

case 'T2E'
    result = this.ParseNum(name, value, 1);
endcase

case 'T2N'
    result = this.ParseNum(name, value, 1);
endcase

case 'WE'
    result = this.ParseNum(name, value, 1);
endcase

case 'WN'
    result = this.ParseNum(name, value, 1);
endcase

case 'WX'
    result = this.ParseNum(name, value, 1);
endcase

case 'WY'
    result = this.ParseNum(name, value, 1);
endcase

case 'WHE'
    result = this.ParseNum(name, value, 1);
endcase

case 'PULSETIME'
    result = this.ParseNum(name, value, 1);
endcase

case 'PULSETYPE1'
    result = this.ParseNum(name, value, 2);
endcase

case 'PULSETYPE2'
    result = this.ParseNum(name, value, 1);
endcase

case 'SIGNALTYPE'
    result = this.ParseNum(name, value, 1);
endcase

case 'THETAMIN'
    result = this.ParseNum(name, value, 1);
endcase

case 'THETAMAX'
    result = this.ParseNum(name, value, 1);
endcase

case 'PHIMIN'
    result = this.ParseNum(name, value, 1);
endcase

case 'PHIMAX'
    result = this.ParseNum(name, value, 1);
endcase

case 'WITHGAUSSIAN'
    result = this.ParseYes(name, value);
endcase

case 'DELTA'
    result = this.ParseNum(name, value, 1);
endcase

case 'ROTATIONAXIS'
    result = this.ParseNum(name, value, 1);
endcase

case 'ROTATIONANGLE'
    result = this.ParseNum(name, value, 1);
endcase

case 'SPINPATHWAY2'
    result = this.ParseNum(name, value, 2);
endcase

case 'SPINPATHWAY3'
    result = this.ParseNum(name, value, 3);
endcase

case 'FEDTIME'
    result = this.ParseNum(name, value, 1);
endcase

case 'ROTATEDEGREE'
    result = this.ParseNum(name, value, 1);
endcase

case 'WITHTITLE'
    result = this.ParseNum(name, value, 1);
endcase
function result = ParseYes(~, name, value)
    str = strtrim(value);
    if ~(strcmpi(str, 'y') || strcmpi(str, 'yes') || strcmpi(str, 'n') || strcmpi(str, 'no'))
        error('Error: %s must have a value yes/no, (XXX) not %s', name, value);
    end
    result = strcmpi(str, 'y') || strcmpi(str, 'yes');
end

function result = ParseNum(~, name, value, n)
    result = str2num(value);
    if isempty(result)
        error('Error: %s must have a number, not %s', name, value);
    elseif size(result) ~= n
        error('Error: %s must have numbers of %s, not (XXX)%s', name, n, value);
    end
end

function RecordItem(this, name, value)
    item = {strcat(strtrim(name), ': ', value)};
    this.Parameters = vertcat(this.Parameters,...
    item);
end

function SaveParameters(this)
    fileName = this.FileName;
    [filepath,name,~] = fileparts(fileName);
    if isempty(filepath)
        fileName = name;
    else
        fileName = strcat(filepath, '\', name);
    end
    fileName = strcat(fileName, '_par.txt');
    fid=fopen(fileName,'wt');
    items = this.Parameters;
    [rows,~]=size(items);
    for i=1:rows
        fprintf(fid,'%s
', items{i});
    end
    fclose(fid);
end

end % methods
end % IO_class
Appendix L. Input file for parameters. Create a folder and a .txt file inside that %folder (name: Data/Single&Powder _Data):

% 2d-ELDOR (Two-dimensional electron-electron double resonance) Pulse EPR Matlab simulation
% Input data
% Lin Li, Department of Physics Concordia, 2018-05-03
% A comment starts with ‘%’
% format of data entry:
% name: data1, data2,...,datan

titleName: Welcome to 2D SECSY, Echo-ELDOR!

% yes/no
withTitle: yes

% Only plot the backuped time-domain data:
%yes/no
plotExistData: no

%userInput: yes/no
userInput: yes

%withRelaxation: yes/no
withRelaxation: yes

%Sampling frequency in 1 Microsecond
fs1: 200
fs2: 200
% unit: Gauss, (3300)
B0: 0.0d0

% w_n the nuclear Larmor frequency for the nucleus uint: MHz
w_n: 2*pi*14.5

g_tensor: 2.0026d0, 2.0035d0, 2.0033d0

% T2e Off-diagonal electron spin-spin relaxation, unit: microsecond
t2e: 0.9d0

% T2n Off-diagonal spin-nuclear relaxation, unit: microsecond
t2n: 22.0d0

% Diagonal Lattice induced electron-spin flip relaxation rates, unit: 1/microsecond
we: 1.67E-02

% Diagonal Lattice induced nuclear-spin flip relaxation rates, unit: 1/microsecond
wn: 7.14E-03

% Diagonal Cross relaxation, unit: 1/microsecond
wx: 6.17E-03
wy: 6.17E-03

% Heisenber spin exchange, unit: 1/microsecond
whe: 0.0d0

% pulse time for a pi/2 pulse, unit: Microsecond
pulsetime: 5.0E-03;
pulsetype: 0.0d0, pi/2.0

% (phase, tipAngle) pair
pulsetype2: 0.0d0, pi

signalType: 1
% the rotation axis for the goniometer: x = 1
% (zy-quadrant), y = 2 (zx-quadrant), z = 3 (xy
%-quadrant)
rotationAxis: 1;

% in degree
rotationAngle: 45;

% with Gaussian: yes/no
withGaussian: yes

% Gaussian inhomogeneous broadening effect,
% unit: MHz
Delta: 5.0

% unit: radian
thetaMin: 0;
thetaMax: pi/2;
phiMin: 0;
phiMax: pi;

% In the coherence pathway, coherence order p
% for 2 pulses
spinPathway2: 1, -1

% In the coherence pathway, coherence order p
% for 3 pulses
spinPathway3: 1, 0, -1

%FEDTime: 5, 20, 40, 60; unit: microsecond
FEDTime: 5

% rotateDegree: rotate the plot around z-axis,
% unit: degree, not radian
rotateDegree: 0

% yes/no
withTranspose: yes
plotAll: no

rndoff: 10e12;

References

1. Lee Sanghyuk, Patyal Baldev R. Freed Jack H. A two-dimensional Fourier transform electron-spin resonance (ESR) study of nuclear modulation and spin relaxation in irradiated malonic acid. J. Chem. Phys. 1993; 98(5):3665-3689.

2. Schweiger Arthur, Jeschke Gunnar. Principles of Pulse Electron Paramagnetic Resonance. Oxford University Press; 2001.

3. Deligiannakis Yiannis, Louloudi Maria, Hadjiliadis Nick. Electron spin echo envelope modulation of metal centers. Coord. Chem. Rev. 2000; 204(1):1-112.

4. Prisner Thomas, Rohrer Martin, MacMillan Fraser: Pulsed EPR spectroscopy: biological applications. Ann. Rev. Phys. Chem. 2001; 52(1):279-313.

5. Van Doorslaer Sabine, Vink Evi. The strength of EPR and ENDOR techniques in revealing structure–function relationships in metalloproteins. Phys. Chem. Chem. Phys. 2007; 9(33):4620-4638.

6. Misra Sushil K, Li Lin. A Rigorous Procedure for Calculation of Pulsed EPR Signals with Relaxation. J Apl Theol. 2018; 2(1):5-16.

7. Høegh HJ, Krzystyyniak M, Charnock GTP, Hore Pj, Kuprov Ilya. Spinach—a software library for simulation of spin dynamics in large spin systems. J. Magn. Reson. 2011; 208(2):179-194.

8. Pribitzer Stephan, Doll Andri, Jeschke Gunnar. SPIDYAN, a MATLAB library for simulating pulse EPR experiments with arbitrary waveform excitation. J. Magn. Reson. 2016; 263:45-54.

9. Gamliel Dan, Levanon Haim. Stochastic processes in magnetic resonance. World Scientific; 1995.

10. Jeener Jean. Superoperators in magnetic-resonance. Adv. Magn. Reson. 1982; 10:1-51.

11. Redfield Alfred G. On the theory of relaxation processes. IBM Journal of Research and Development. 1957; 1(1):19-31.

12. Dan Gamliel, Jack H. Freed. Theory of Two-Dimensional ESR with Nuclear Modulation. J. Magn. Reson. 1990; 89:60-93.

13. Abragam Anatole. The Principles of Nuclear Magnetism. Oxford University Press; 1961.

14. Freed Jack H. Theory of multiple resonance and ESR saturation in liquids and related media. Multiple Electron Resonance Spectroscopy. US: Springer; 1979. p. 73-142.

15. Freed Jack H. Generalized Cumulant Expansions and Spin-Relaxation Theory. J. Chem. Phys. 1968; 49(1):376-391.

16. Freed Jack H. Quantum Effects of Methyl-Group Rotations in Magnetic Resonance: ESR Splittings and Linewidths. J. Chem. Phys. 1965; 43(5):1710-1720.

17. Gepnerle C, Aebli G, Schweiger A, Ernst RR. Phase cycling in pulse EPR. J. Magn. Reson. 1990; 88(2):241-256.

18. Misra Sushil K, Salahi HR, Li Lin. Calculation of single crystal and polycrystalline pulsed EPR signals including relaxation by phonon modulation of hyperfine and g matrices by solving Liouville von Neumann equation. Magn. Reson. Solids. 2019 (to be published).