VERIZONA PUBLISHER
Publishing Your Ideas

www.verizonaonlinepublishing.com

Vol: 3, Issue: 2

Journal of Applied and Theoretical Physics Research

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

*Corresponding author: Sushil K. Misra, Physics Department, Concordia University, 1455 de Maisonneuve Boulevard West, Montreal, Quebec H3G 1M8, Canada; E mail: sushil.misra@concordia.ca

Article Type: Research, Submission Date: 22 August 2019, Accepted Date: 01 september 2019, Published Date: 23 September 2019.

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.

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 lap top 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 straight forward 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, ρ, taking into account the relaxation effects is expressed in Liouville space as follows [6, 9-13].

$$\frac{d}{dt}\rho(\Omega,t) = -i[\hat{H},\rho(t)] - \hat{\hat{\Gamma}}(\rho(t)-\rho_0)$$

where $\hat{H} = \hat{H}_0 + \hat{H}_1$ is the Hamiltonian operator; $\hat{H}_0$ and $\hat{H}_1$ are its time-independent and time-dependent parts, respectively; $\hat{\hat{\Gamma}}$ is the relaxation superoperator, and assumed to be time independent here. (Throughout the paper, the single and double carets " $\hat{}$ " and " $\hat{\hat{}}$ " will be used to denote the operator and the superoperator, respectively.) In Eq. $\rho_0 \propto S_z$ 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 denoted as $\chi(t) \equiv \rho(t) - \rho_0$. Finally Eq. can be expressed as follows:

$$\frac{d}{dt}\chi(t) = -i[\hat{H},\chi(t)] - \hat{\hat{\Gamma}}\chi(t) \tag{2.2}$$

The operator equation (2.2), can be expressed as a matrix equation in a given set of operators, where are the eigenvectors of $\hat{H}_0$, in terms of the coefficients, χ_{ij}, H_{ij} in the expansion of the operators χ and $\hat{H}$ in this basis, respectively.

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

Because the LVN 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:

$$\frac{d}{dt}\hat{\chi} = Col\{-i[\hat{H},\chi(t)] - \hat{\hat{\Gamma}}\chi(t)\} \tag{2.3}$$

Here $\hat{\chi} \equiv Col\{\chi(t)\}$ denotes the column of the matrix $\chi(t)$ formed by stacking the columns of χ_{ij} into a single column vector as a matrix.

The Liouvillian

$$\hat{\hat{L}} \equiv (I_n \otimes \hat{H} - \hat{H}^T \otimes I_n) \tag{2.4}$$

with the dimension , 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:

$$\hat{\hat{R}}\hat{\chi} \equiv Col\{\hat{\hat{\Gamma}}\chi(t)\} \tag{2.5}$$

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

$$\hat{\hat{R}}_{\alpha\times n+\beta\ \alpha'\times n+\beta'} = \hat{\hat{\Gamma}}_{\alpha\beta\alpha'\beta'}; \alpha,\beta,\alpha',\beta' = 1,2,\ldots n \tag{2.6}$$

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

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

$$\frac{d}{dt}\hat{\chi} = -\hat{\hat{L}}'\hat{\chi} \tag{2.7}$$

where the generalized Liouville superoperator, $\hat{\hat{L}}'$. includes also the relaxation superoperator:

$$\hat{\hat{L}}' = i\hat{\hat{L}} + \hat{\hat{R}} \tag{2.8}$$

The solution of Eq. (2.7) is

$$\hat{\chi}(t) = e^{-(t-t_0)\hat{\hat{L}}'}\hat{\chi}(t_0) \tag{2.9}$$

In Eq. (2.9), $\hat{\chi}(t_0)$ 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 $\hat{\hat{H}}_0$, $\hat{\hat{H}}_1$ and $\hat{\hat{\Gamma}}$ where $\hat{H}_0$ is the static Hamiltonian, that given by Eq. (A.2) in Appendix below, and the pulse operator, $\hat{H}_1(t)$, is expressed as

$$\hat{H}_1(t) = \hat{\varepsilon}(t) = B_1\gamma_e(S_{1x}\cos(\phi) + S_{1y}\sin(\phi) + S_{2x}\cos(\phi) + S_{2y}\sin(\phi)) \tag{2.10}$$

where γ_e is the gyromagnetic ration of the electron, and $\hat{\varepsilon}(t)$ is the irradiating microwave pulse with the intensity $B_1=\omega/\gamma_e$, and phase ϕ . The tip angle θ over time t during the pulse is. $\theta=\omega t=B_1\gamma_e t$

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

$$\frac{d\hat{\hat{\rho}}}{dt} = -\hat{\hat{P}}\hat{\hat{\rho}} \tag{2.11}$$

where the propagator, taking into account the static Hamiltonian and the pulse, is

$$\hat{\hat{P}} \equiv i\left[I_n \otimes \left(\hat{H}_0 + \hat{H}_1(t)\right) - \left(\hat{H}_0 + \hat{H}_1(t)\right)^T \otimes I_n\right]\hat{\hat{R}} \tag{2.12}$$

The solution of Eq. (2.11) is

$$\hat{\hat{\rho}}(t) = e^{-(t-t_0)\hat{\hat{P}}}\hat{\hat{\rho}}(t_0) \tag{2.13}$$

Here $\hat{\hat{\rho}}(t_0)$ 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, H_0 (free evolution) in the absence of pulses, and under the action of the pulse H_1 , including the static Hamiltonian and relaxation, is discussed in the Appendix.

The calculation of the final density matrix, ρ_f , corresponding to the coherence pathways S_{c-} 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 coherent pathway S_{c-} in accordance with that used by LPF [1] (for more details of coherence pathways, see [6]). In these experiments, the time, t_1 between the pulses, for time-domain signals, is stepped in the experiment and the time t_2 is measured from the top of the echo, as shown in Figures 1 and 2.

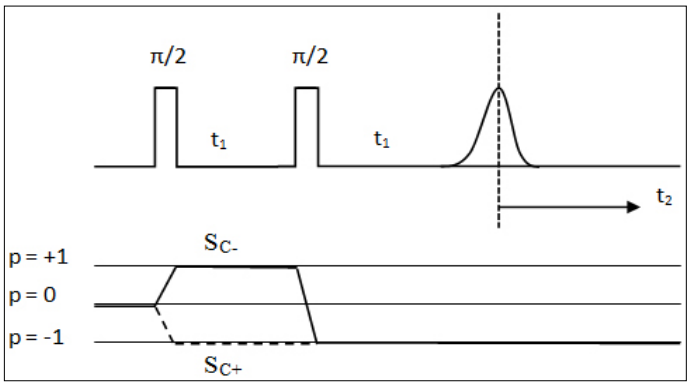


Figure 1: (Top) Pulse sequence for obtaining SECSY signal. The t_1 time between the two pulses and t_2 time from the echo are stepped. (Bottom) Coherence pathways used for calculating SECSY signal for an unpaired electron (S = ½) interacting with a single nucleus (I = ½). p is the coherence order, which represents transverse magnetization, corresponding to spins rotating in a plan perpendicular to the external field [17]

The 2D time-domain signal is calculated from ρ_f as follows:

$$S(t_1,t_2)=Tr(S_+\rho_f)=Tr\left(\left(S_x+iS_y\right)\rho_f\right)$$

The Fourier transform (FT) of the two-dimensional time domain signal $S(t_1,t_2)$, is the corresponding 2D-FT signal, $S(\omega_1,\omega_2)$.

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].

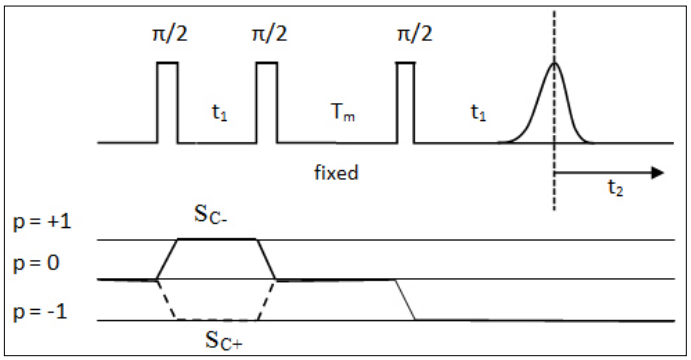


Figure 2: (Top) Pulse sequence for obtaining echo- ELDOR signal. The t_1 time between the first two pulses and t_2 time from the echo are stepped. Here T_m is the mixing time. (Bottom) Coherence pathways used for calculating echo- ELDOR signal for an unpaired electron (S = ½) interacting with a single nucleus (I = ½). p is the coherence order, which represents transverse magnetization, corresponding to spins rotating in a plan perpendicular to the external field [17]

The flow chart for executing the algorithm developed here is provided in Ref. [6].

Simulation of SECSY and echo-ELDOR signals

The technique developed here is used to illustrate for the cases of 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 $\tilde{A}$ and those of the $\tilde{g}$ 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 $\sim 5ns$ [1]; nuclear Zeeman frequency ω_n=14.5MHz; the spin-Hamiltonian parameters: $\tilde{g}=\left(g_{xx},g_{yy},g_{zz}\right)=(2.0026,2.0035,2.0033)$;
$\tilde{A}=\left(A_{xx},A_{yy},A_{zz}\right)=(-61.0MHz,-91.0MHz,-29.0MHz)$.

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 Δ=4MHz; electron spin–spin relaxation time, T_{2e}=0.900μs; nuclear spin–spin relaxation time, T_{2n}=22μs; inverse electron spin–spin relaxation time, W_e=0.0167μs^{-1}; inverse nuclear spin–spin relaxation time, W_n=0.00714μs^{-1}; inverse electron nuclear spin–spin relaxation time, W_x=0.00617μs^{-1}; inverse electron-nuclear spin–spin relaxation time W_x=W_y; inverse Heisenberg exchange relaxation time, ω_{HE}=0.0μs^{-1}.

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

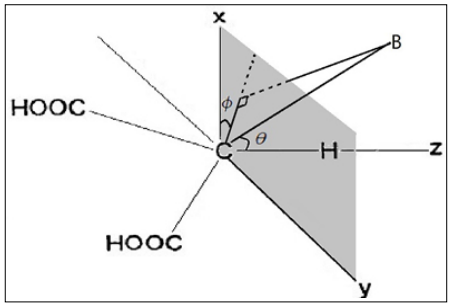


Figure 3: Relation of the principal axes (x,y,z) of the $\tilde{g}$ and $\tilde{A}$ (hyperfine) matrices, assumed coincident to the structure of the malonic acid radical $CH(COOH)_2$. Here, the z axis is along the C-H bond direction and the x- axis is perpendicular to the plane of the three carbon atoms [14]). The direction of the external static field B_0 is defined by the angles θ and φ, where θ is the angle between B_0 and the z axis, and φ is the angle between the x axis and the projection of B_0 on the xy plane

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: $(\theta,\phi)=\left(5^\circ,0^\circ\right),\left(30,0^\circ\right),\left(50^\circ,0^\circ\right)$ in the zx-quadrant, so that the corresponding Euler angles are $(\alpha,\beta,\gamma)=(0,-\theta,0)$ and (ii) in Figure 5, corresponding to the experiment of LPF [1] for two orientations $(\theta,\phi)=\left(5^\circ,90^\circ\right)$ and $\left(45^\circ,90^\circ\right)$ in the zy-quadrant, which correspond to $(\alpha,\beta,\gamma)=\left(0,\theta,90^\circ\right)$. [The relationship between (θ ,φ) and (α ,β ,γ) is described by Misra et al. [18]].

As for the echo-ELDOR signal, the time-domain signals, were calculated for the orientation $(\theta,\phi)=\left(30^\circ,0^\circ\right)$ in the zx-quadrant, so that $(\alpha,\beta,\gamma)=(0,-\theta,0)$ [1], with four mixing times T_m (the fixed time interval between the second and third $\pi/2$ pulses; see Figure 2): (a) 5 μS; (b) 20 μS; (c) 40 μS; (d) 60 μS. 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 ω_α and ω_β, are the same in all four cases shown in Figure 6. Their values are the same as those reported in LPF [1], i.e. $\omega_\alpha \approx 7.0 MHz$ and $\omega_\beta \approx 32.0 MHz$. 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 significantlythe peaks in the FT spectra; this fact is more evident inthe coherence cross peaks.

Forthermore, 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 ($\sim 5ns$) 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

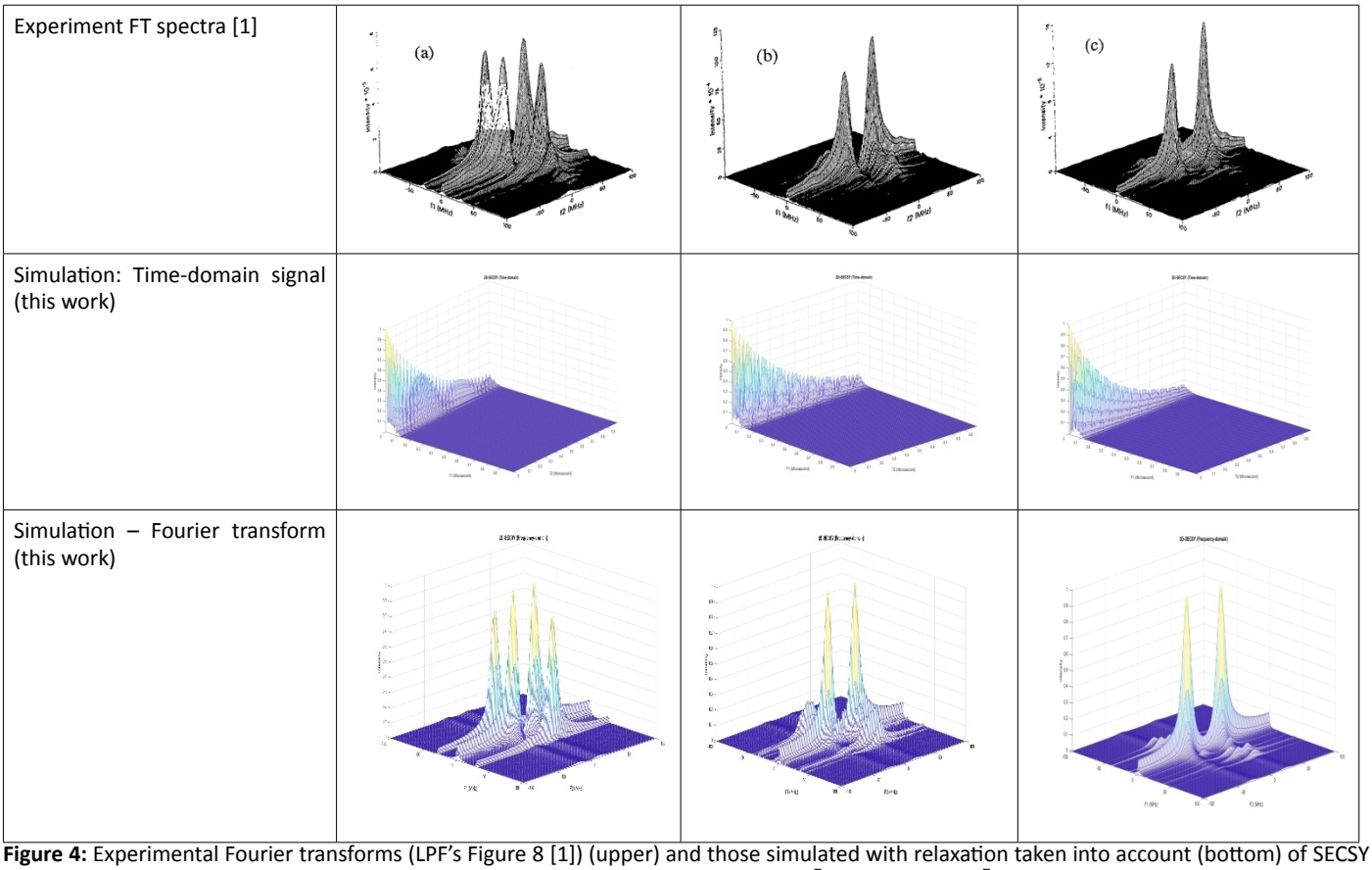


Figure 4: Experimental Fourier transforms (LPF's Figure 8 [1]) (upper) and those simulated with relaxation taken into account (bottom) of SECSY spectra at (θ,ϕ) orientations of (a) (5°, 0°), (b) (30°, 0°), (c) (50°, 0°) in the zx-quadrant $\left[(\alpha,\beta,\gamma)=(0,-\theta,0)\right]$ [1]. The corresponding simulated time-domain spectraare shown in the middle row. The relaxation constants $T_2(ns)$ used for the various orientations are: (a) 800, (b) 812, (c) 687. A Gaussian inhomogeneous broadening width Δ=4 MHz, Eq. (A.14), is used in the simulation of the 2D-FT signal for each orientation

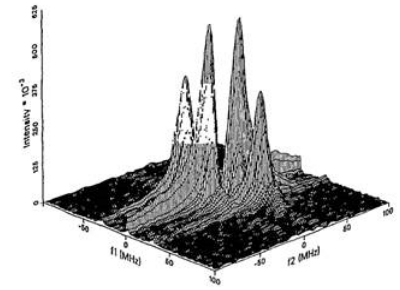
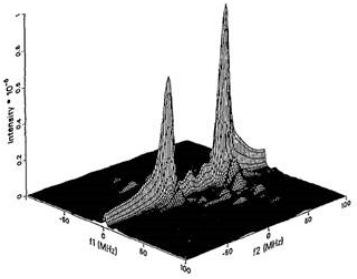
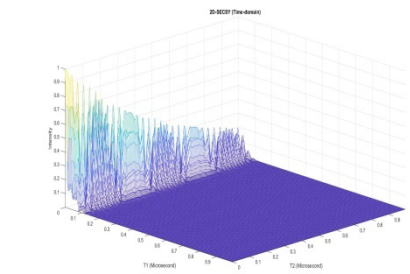
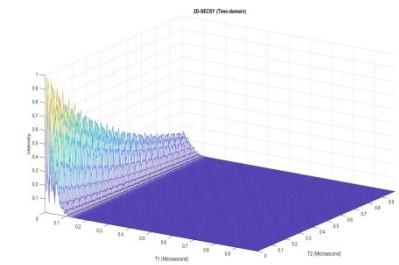
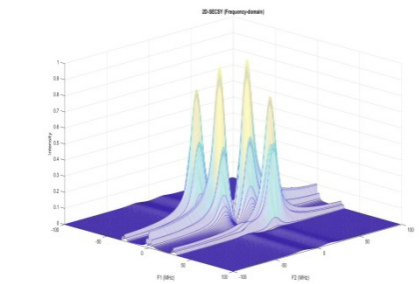
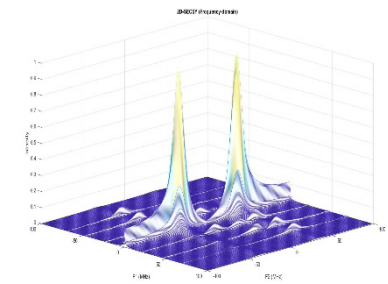
Experiment FT spectra [1]		
Simulation Time-domain signal (this work)		
Simulation – Fourier transform (this work)		

Figure 5: Experimental (LPF's Figure 9[1]) (upper) and that simulated with relaxation taken into account (bottom) of the SECSY spectra at (θ,ϕ) orientations of (a) (5°, 90°), (b) (45°, 90°) in the zy-quadrant $[(\alpha,\beta,\gamma)=(0,-\theta,0)]$, in the second and third column respectively [1]. The corresponding simulated time-domain spectraare shown in the middle row. The relaxation constants $T_2\,(ns)$ for the various orientations are (a) 1223, (b) 1068. A Gaussian inhomogeneous broadening width Δ=4 MHz, Eq. (A.14), is used in the simulation of 2D-FT signal for each orientation

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 pule Hamiltonian [1]

$$\hat{H}=\hat{H}_0+\hat{H}_1 \tag{A.1}$$

where

$$\hat{H}_0=CS_z-\omega_n I_z+AS_z I_z+\frac{1}{2}BS_z I_+ +\frac{1}{2}B^* S_z I_- \tag{A.2}$$

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

$$C=\frac{\beta_e B_0}{h}\left[\bar{g}+F\frac{1}{2}\left(3\cos^2\beta-1\right)+F^{(2)}\sin^2\beta\cos(2\gamma)\right] \tag{A.3}$$

$$A=-2\pi\left[\bar{a}+D\frac{1}{2}\left(3\cos^2\beta-1\right)+D^{(2)}\sin^2\beta\cos(2\gamma)\right] \tag{A.4}$$

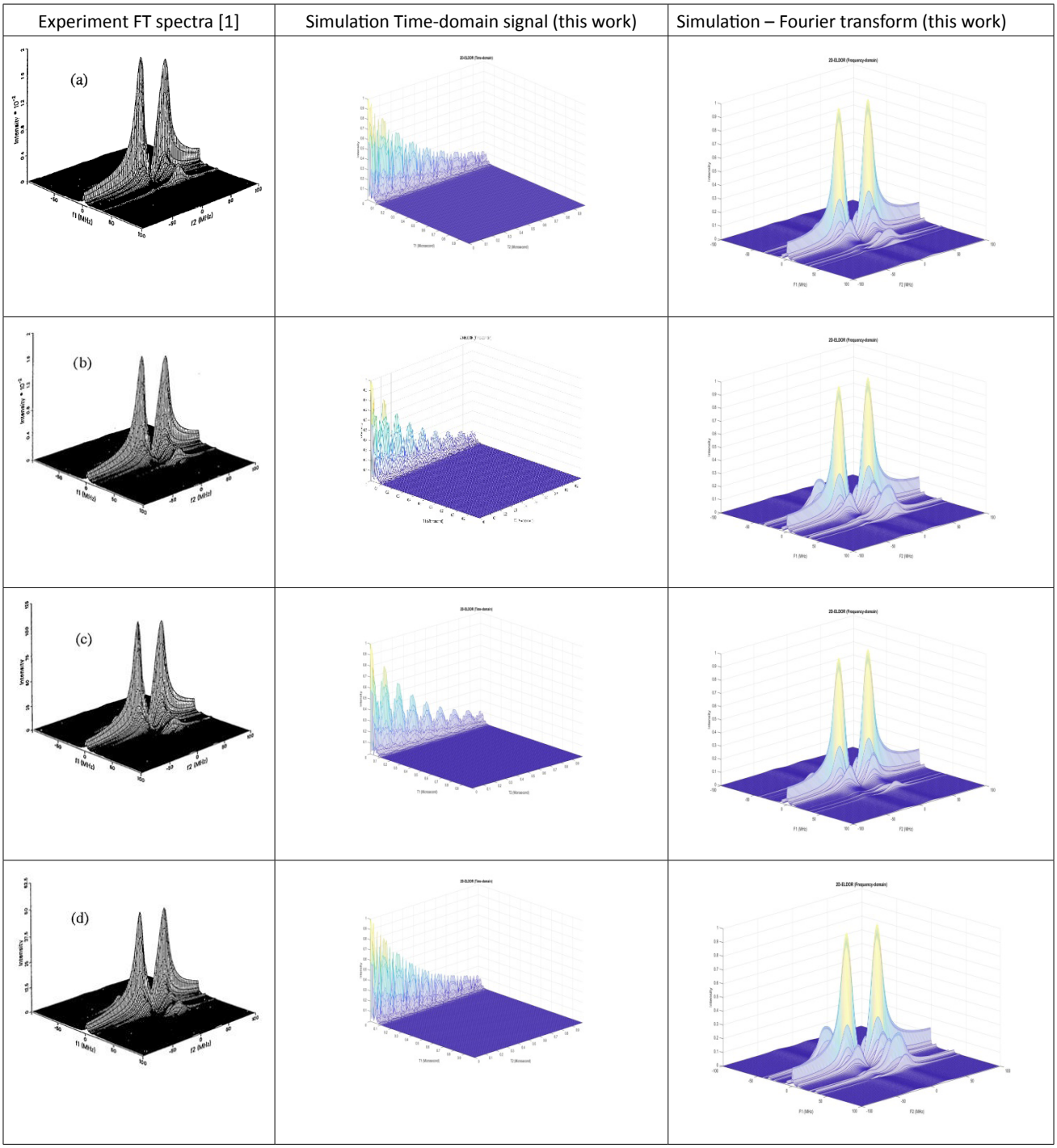


Figure 6: Experimental FT (LPF Figure 11[1]) (first column) and simulated time-domain with relaxation taken into account (second column) of the echo-ELDOR spectra at (θ,ϕ) orientations of (30°, 0°) in the zx-quadrant $\left[(\alpha,\beta,\gamma)=(0,-\theta,0)\right]$ [1], with the mixing times T_m: (a) 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 Δ=4 MHz, Eq. (A.14), is used in the simulation

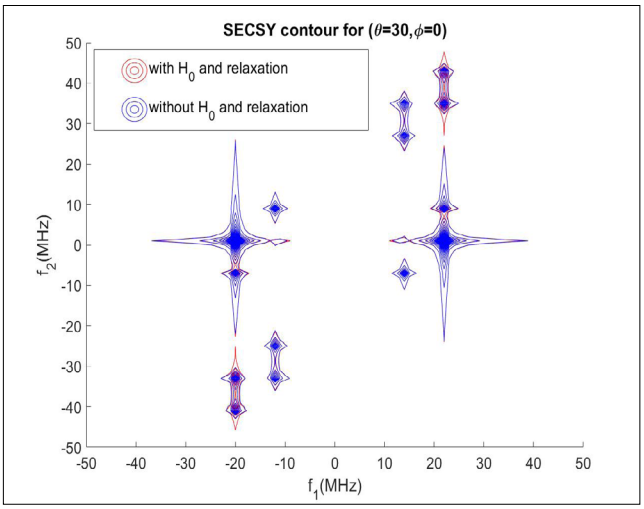


Figure 7: Overlap of SECSY contour plots in the Fourier domain for the two cases: i) With H_0 and relaxation included during the pulses (red) ii) Without both H_0 and relaxation during the pulses (blue). The two plots are found to be distinctly different from each other

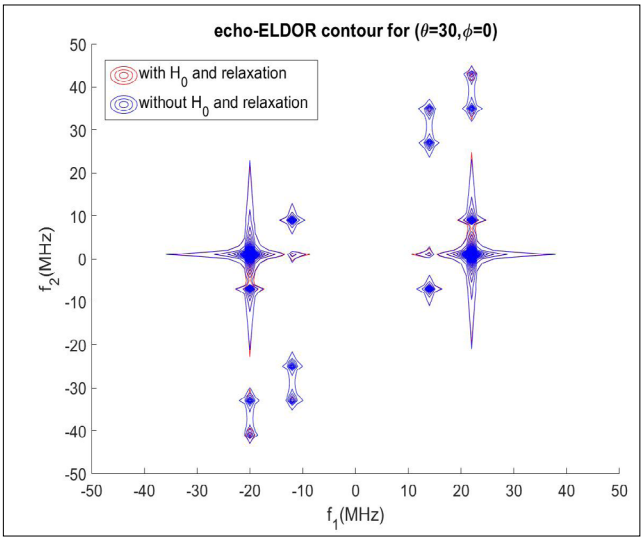


Figure 8: Overlap of echo-ELDOR contour plots in the Fourier domain for the two cases: i) With H_0 and relaxation included during the pulses (red) ii) Without both H_0 and relaxation included during the pulses (blue). The two plots are found to be distinctly different from each other

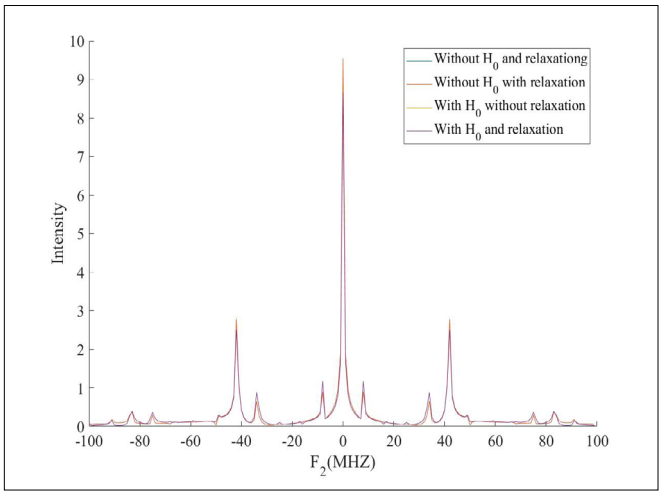


Figure 9: Comparison of SECSY 1D spectrum in the Fourier domain along f_2 for the slice along f_1=0 for four different cases: i) Without H_0 and relaxation ii) Without H_0 but with relaxation included iii) With H_0 without relaxation iv) With H_0 and with relaxation during the pulses

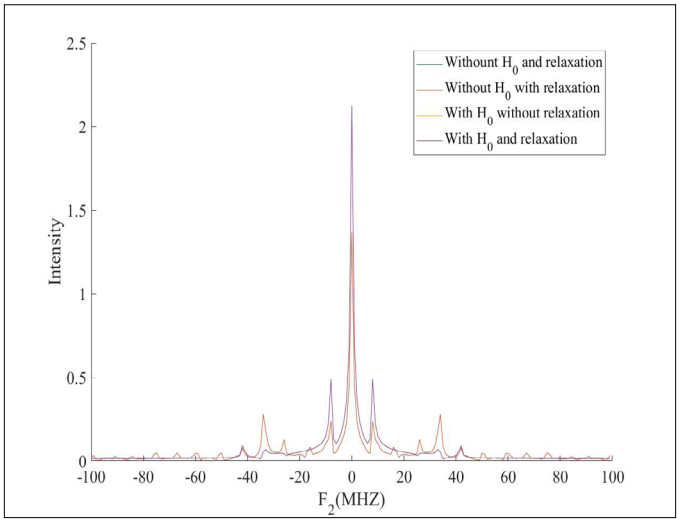


Figure 10: Comparison of echo-ELDOR 1D spectrum in the Fourier domain along f_2 for the slice along f_1=0 for four different cases: i) Without H_0 and relaxation ii) Without H_0 but with relaxation included iii) With H_0 without relaxation iv)With H_0 and with relaxation during the pulses

$$B=-4\pi\left\{D\frac{3}{4}\sin\beta\cos\beta-D^{(2)}\frac{1}{2}\sin\beta\left[\cos\beta\cos(2\gamma)-i\sin(2\gamma)\right]\right\} \quad \text{(A.5)}$$

Here $\Omega(0,\beta,\gamma)$ are the Euler angles which describe the orientations of the principal axes of the $\tilde{g}$ -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 2π in Eq. (A.4) and 4π in Eq. (A.5), there was the factor $\frac{\gamma_e}{\hbar}$ in both places in LPF [1].)

$$\bar{g}=\frac{1}{3}\left(g_{xx}+g_{yy}+g_{zz}\right) \quad \text{(A.6)}$$

$$\bar{a}=\frac{1}{3}\left(A_{xx}+A_{yy}+A_{zz}\right) \quad \text{(A.7)}$$

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

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

$$F^{(2)}=\frac{1}{2}\left(g_{xx}-g_{yy}\right); \quad \text{(A.10)}$$

$$D^{(2)}=\frac{1}{2}\left(A_{xx}-A_{yy}\right); \quad \text{(A.11)}$$

Here $\bar{g}$ and $\bar{a}$ 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]:

$$\frac{d}{dt}\rho_{\alpha\alpha'}(t)=-i\omega_{\alpha\alpha'}\rho_{\alpha\alpha'}-\sum_{\beta\beta'}\hat{\hat{\mathrm{R}}}_{\alpha\alpha'\beta\beta'}\left(\rho_{\beta\beta'}-\rho_{0\beta\beta'}\right) \quad \text{(A.12)}$$

Where

$$\omega_{\alpha\beta}=E_\alpha-E_\beta \quad \text{(A.13)}$$

Here E_α , E_β are the eigenvalues of the static Hamiltonian, $\hat{H}_0$, for the electron-nuclear coupled system (S = ½, I = ½).

In Eq. (A.12), $\hat{\hat{\mathrm{R}}}_{\alpha\alpha'\beta\beta'}$ are relaxation matrix elements, where α, α', β, β' 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 ω_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:

$$f_b(t_2)=f(t_2)\frac{1}{\sqrt{2\pi}\Delta}\int_{-\infty}^{\infty}exp\left(-\frac{\nu^2}{2\Delta^2}\right)e^{-i2\pi\nu t_2}d\nu=f(t_2)e^{-2(\pi\Delta t_2)^2} \quad \text{(A.14)}$$

where $f_b(t_2)$ is the Gaussian-broadened signal along t_2 and Δ 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 "polycrystalline".)

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 theappendices. 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) *nphis*: enter the number of phi values on the unit-sphere grid (enter 1 for single-crystal orientation simulation). (i) For the choice *nthetas*=*nphis*=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 *nphis*=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_1, 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 "…" at the end. If the code does not run because of the presence of "…" (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 Matlabsimulation
% Lin Li, Department of Physics Concordia,
%20 18-05-03

%function Single&Powder_Main()
```

```
%fclose('all'); % Close all open files

clear % Remove items from MATLAB workspace and
%reset MuPAD engine

%close all hidden % Deletes all figures

%including those with hidden handles.

clc%  Clears all input and output from the
%Command Window display


disp('Welcome to SECSY and Echo-ELDOR!');


% 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);

if editParameters

system('notepad data/Single&Powder_Data.txt');

    return;

end


%fs1 Sampling frequency

fs1 = myIO.GetData('fs1');

%fs2 Sampling frequency

fs2 = myIO.GetData('fs2');


%Only plot the backuped time-domain data

plotExistData = myIO.GetData('plotExistData');

plot_class=Plot_class(fs1, fs2,plotExistData);


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

rotateDegree = myIO.GetData('rotateDegree');

withTitle = myIO.GetData('withTitle');

withTranspose = myIO.GetData('withTranspose');

plotAll = myIO.GetData('plotAll');


if plotExistData

disp('Please select an exist file to plot.')

plotTitle = '';


    filter = 'figures/*.dat'

    title = 'Select a data file to open'

    [file,path] = uigetfile(filter,title);


if(file == 0)

error('Error: \n Not a valid file %s', file

    end


signalFile = strcat(path, file);


plot_class.SignalToFFT_Init(signalFile);

plot_class.Plot(plotTitle, withTitle,...

rotateDegree, withTranspose, plotAll);


    return;

end


myIO.RecordItem('Start', datestr(now));


%titleName = myIO.GetData('titleName');


prompt = 'Please enter number of pulses, 2(for

(XXX) SECSY) or 3(for Echo-ELDOR): ';

min = 2;

max = 3;

nPulses = myIO.InputInt('pulses', prompt,...
min, max);
```

```
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');

% prompt = 'With relaxation, Y/N: ';

% withRelaxation = myIO.InputYes('Relaxation',
%prompt);


thetaMin = myIO.GetData('thetaMin');

cosThetaMin = cos(thetaMin);

thetaMax = myIO.GetData('thetaMax');

cosThetaMax = cos(thetaMax);

thetas = zeros(nThetas, 1); % [min, max]

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


phiMin = myIO.GetData('phiMin');

phiMax = myIO.GetData('phiMax');

phis = zeros(nPhis, 1); % [min, max]

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');

rotationNote = '';

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);


prompt = 'Please enter the rotate angle

(XXX)theta (0 - 180): ';

    min = 0;

    max = 180;

%rotationAngle = myIO.GetData('rotateAngle');

Theta = myIO.InputInt('Theta', prompt,...

 min, max);


    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

if ((nThetas == 1) && (nPhis!= 1))



prompt = 'Please enter the rotate angle

(XXX)theta (0 - 180): ';

    min = 0;

    max = 180;

%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);



   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


    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(withGaussian)

    %         rotationNote =
%strcat(rotationNote, ', with Gaussian');

    %     else

    %         rotationNote =
%strcat(rotationNote, ', without Gaussian');

    %     end
```

```
end

prompt = 'With Gaussian, Y/N: ';

withGaussian = ...

myIO.InputYes('WithGaussian', prompt);



prompt = 'With Relaxation During the

(XXX)Pulses?, Y/N: ';

withrelaxationpulse = ...

myIO.InputYes('WithRelaxationPulse', prompt);


prompt = 'With H0 During the Pulses?, Y/N:';

withh0pulse =...

myIO.InputYes('WithRelaxationPulse', prompt);

% make output data file name

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 uint: 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'); %

we = myIO.GetData('we'); % Diagonal Lattice

%induced electron-spin flip ralaxation rates,

%unit: 1/microsecond

wn = myIO.GetData('wn'); % Diagonal Lattice

%induced nuclear-spin flip ralaxation rates,

%unit: 1/microsecond

wx = myIO.GetData('wx'); % Diagonal Cross

%relaxation, unit: 1/microsecond

wy = myIO.GetData('wy'); % Diagonal Diagonal

%Cross relaxation, unit: 1/microsecond

whe = myIO.GetData('whe'); % Heisenber spin

%exchange, unit: 1/microsecond

rndoff = myIO.GetData('rndoff'); % round off

relaxation = Relaxation_class(t2e, t2n, we,...

wn, wx, wy, whe);


hamiltonian =...

Hamiltonian_class(hamiltonianFactor,...

relaxation, withRelaxation,rndoff);


density = Density_class(hamiltonian);


%pulse time of a pi/2 pulse, unit: Microsecond

pulsetime = myIO.GetData('pulsetime');

%(phase, tipAngle) pair

pulsetype1 = myIO.GetData('pulsetype1');

% (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 p
%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);
    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\n',nPulses);
fprintf('(nThetas,nPhis) = (%d,
(XXX)%d)\n',nThetas,nPhis);

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, uint: MHz

G_tensor; % g_tensor(3)

Hyperfine_tensor; % hyperfine_tensor(3)

%hyperfine tensor, unit: Gauss


C;      %uint: MHz

A;      %uint: MHz

         B1;    %uint: MHz

         B2;    %uint: MHz


    end %properties


    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, uint: 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:

%,μ_B=9.27400968(20)×10^(-28)  J*Gauss^(-1)

planck = 6.62606957D-34; %Planck constant:

%h=6.62606957(29)×10^(-34) J*S

planck = planck/(2*pi); % reduced Planck

%constant J*S/rad
```

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



wn = this.Wn;

g_tensor = this.G_tensor;

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) +...

g_tensor(2) + g_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;
```

```
                % alpha = euler_angles(1);

                beta = euler_angles(2);

                gamma = euler_angles(3);



this.C = gAverage+0.5*f * (3*cos(beta)*...

cos(beta) - 1) + f2 * sin(beta)*sin(beta)*...

cos(2* gamma);



this.A =-2*pi*(aAverage + 0.5*d *(3.0*...

cos(beta)*cos(beta) - 1) + d2 *...

sin(beta)*sin(beta) * cos(2 * gamma));



this.B1 = -4*pi*((3.0/4.0)*d * sin(beta) *...

cos(beta)-0.5*d2*sin(beta) * (cos(beta) *...

cos(2 * gamma) -1i*sin(2*gamma)));



this.B2 =-4*pi* ((3.0/4.0)*d * sin(beta)*...

cos(beta)-0.5*d2 * sin(beta) * (cos(beta)...

* cos(2 * gamma) +1i*sin(2*gamma)));



% For calculation of orientation-dependent spin
%relaxation matrix

%            wa = (this.A/2- wn)^2 +
%(abs(this.B1/2))^2;

%            wa = sqrt(wa);

%            this.Wa = wa;

%

%            wb = (this.A/2 + wn)^2 +
%(abs(this.B1/2))^2;

%            wb = sqrt(wb);

%            this.Wb = wb;



%            this.C1 = 1 + (this.A/2 - wn)/wa;

%            this.C1 = sqrt(this.C1/2);
```

```
%             this.C2 = 1 - (this.A/2 - wn)/wa;

%             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 %classdefHamiltonianFactor_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
%Modified by H R Salahi


classdef Hamiltonian_class< handle

    properties

HamiltonianFactor;

        Relaxation;

WithRelaxation = true;


HamiltonianSzIzH ; % Hamiltonian %operator in
%Hilbert space and |SzIz> basis

EigenvalueH ; % Of Hamiltonian operator.

RightEigenvectorH; % Of Hamiltonian operator.
%Au = tu, any such column vector u is called a
%right eigenvector of A

RightEigenvectorL % Of Liouvillian operator

HamiltonianH0L;

LiouvilleH0; % Liouville superoperator in

%Liouville space  by eigenvector basis of H

Rndoff; %Round off

    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;

hamiltonianFactor.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(:,[3,4]) =

%rightEigenvectorH(:,[4,3]);

this.RightEigenvectorL = rightEigenvectorL;

liouvilleH=H0ToNormalBasis(this,liouville);

this.LiouvilleH0 = liouvilleH;
```

```
        end % Init()

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] = ...
H0ToNormalBasisDensity(this,operator)

A = ctranspose(this.RightEigenvectorL)*...
operator;
 end

function [A] = NormalToH0BasisDensity...
(this,operator)
A = this.RightEigenvectorL* operator;
        end

function [A] = ToHBasisDensity (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

    end % methods

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


EquilibriumH0H; % 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



 end %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.EquilibriumSzIzH = equilibriumH;

        end % Density_class constructor


        function Init(this)

% Equilibrium density operator in eigenvector

%basis of H

this.EquilibriumH0H = this.Hamiltonian...
.SzIzToH0BasisH(this.EquilibriumSzIzH);

this.EquilibriumH0L = this.EquilibriumH0H(:);


this.DensitySzIzL = this.EquilibriumSzIzH(:);

        end


function[A] = GetDensityInSzIzBasisH(this)


            B = this.DensitySzIzL;

            n = size(B, 1);

            m = sqrt(real(n));


            A = reshape(B, [m, m]);


        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(:);

this.DensitySzIzL=A;

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 GetDensityMat


function ToNormalBasis(this)

B = this.DensitySzIzL;

this.DensitySzIzL =...

this.Hamiltonian.H0ToNormalBasisDensity(B);

        end


        function ToHBasis(this)

            B = this.DensitySzIzL;

this.DensitySzIzL =this.Hamiltonian...

.NormalToH0BasisDensity(B);

        end




        function Print(~)

disp('Density_class');


        end % function Print


    end % methods


end %classdefDensity_class
```

Appendix E. Free evolution operator(name: FED_class):

```
% 2d-ELDOR (Two-dimensional electron-electron
%double resonance) Pulse EPR Matlab simulation

% Lin Li, Department of Physics Concordia,

%2018-05-03


%Free Evolution Decay (FED)

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 this


% 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 % 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;

Pulsetime0; % pi/2 pulse( -5 ns), unit:
%Microsecond
        Pulsetype1;
        Pulsetype2;

PulseOperator1; % pi/2 Pulse superoperator in
%Liouville space by eigenvector basis of H
PulseOperator2; % pi Pulse superoperator in
%Liouville space by eigenvector basis of H
        Relaxation;
Withrelaxationpulse;
        Withh0pulse;
    end % properties

    methods

function this = Pulse_class(hamiltonian,...
relaxation,density,pulsetime0, pulsetype1,...
pulsetype2, withrelaxationpulse,withh0pulse)
this.Hamiltonian = hamiltonian;
this.Density = density;

this.Pulsetime0 = pulsetime0;
% pi/2 pulse( -5 ns), unit: Microsecond
this.Pulsetype1 = pulsetype1;
this.Pulsetype2 = pulsetype2;
this.Relaxation=relaxation;
this.Withrelaxationpulse=withrelaxationpulse;
this.Withh0pulse=withh0pulse;
        end % Pulse_class constructor

        function Init(this)
this.PulseOperator1 =...
this.CalPulseOperator(this.Pulsetype1);
this.PulseOperator2 =...
this.CalPulseOperator(this.Pulsetype2);
        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
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...
(timeH1);


timeH0=this.Hamiltonian.SzIzToH0BasisH(timeH0);
tp=this.Pulsetime0;
            if(this.Withh0pulse)
timeH=timeH+timeH0;
            end
R=this.Relaxation.Relaxation;
p = kron(eye(4), timeH) -...
kron(transpose(timeH), eye(4));
pulseOperator = expm(-1i*p);
 if(this.Withrelaxationpulse)
pulseOperator = round(expm(-1i*p-...
R*tp)*10e10)/10e10;
            end


%p = expm(-1i*timeH1); % exp(-i*t*H1)
%pulseOperator = kron(ctranspose(p), p); % for
%H0 << H1
pulseOperator=this.Hamiltonian...
.H0ToNormalBasis(pulseOperator);
end % function CalPulseOperator


function DensityEvolve(this, pulsetype)
            %phase = pulsetype(1);
tipAngle = pulsetype(2);


            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


        end %function DensityEvolve


        function Print(this)
disp('Pulse_class');
```

```
disp('Pulse superoperator:');

disp('this.PulseOperator1'); %Pulsesuper

%operator in Liouville space and |SzIz> basis


        end % function Print


    end % methods

end %classdefDensity_class
```

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

%-spin relaxation, unit: microsecond

Wy ; % Diagonal electron nuclear spin

%-spin relaxation, unit: microsecond


Whe; % Heisenberg exchange relaxation


    end %properties


    methods


%Suppose, at initial state, the spin is in

%z-axis

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;
```

```
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_dc = wn;

w_ac = we;
w_ca = we;
w_bd = we;
w_db = we;

w_ad = wy;
w_da = wy;

w_bc = wx;
w_cb = wx;

w_ab = w_ab + whe;
w_ac = w_ac + 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_aa = -(w_ab + w_ac + w_ad);
w_bb = -(w_ba + w_bc + w_bd);
w_cc = -(w_ca + w_cb + w_cd);
w_dd = -(w_da + w_db + w_dc);

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(1,1) = w_aa;
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) = w_bd;

relaxation(11,1) = w_ca;
relaxation(11,6) = w_cb;
                %relaxation(11,11) = w_cc;
relaxation(11,16) = w_cd;


relaxation(16,1) = w_da;
relaxation(16,6) = w_db;
relaxation(16,11) = w_dc;
                %relaxation(16,16) = w_dd;


this.Relaxation = relaxation;


end %  Relaxation_class constructor this

 function Print(this)
disp('Relaxation_class');

disp('Relaxation matrix in Liouville space:');
disp(this.Relaxation); % Relaxation operator
% and |SzIz> basis

disp(this);
        end % function Print

    end % methods

end %classdefRelaxation_class
```

Appendix H. Pathway operator (name: Pathway_class):

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

% Coherence Pathway
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 % Pathway_class constructor

        function SelectPath(this, spinPathway)

% In SzIz basis
```

```
density =...
this.Density.GetDensityInSzIzBasisH();

            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):

```
% 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 Signal_class< handle
    properties

        Hamiltonian;
        Density;
        Pulse;
        FED;
        Pathway;

SignalType; % 1 - Sc+, 2 - Sc-
        Tv1;
        Tv2;
        Thetas;
        Phis;
SignalFile;

Delta; %Gaussian inhomogeneous broadening
%effect, unit: MHz
WithGaussian; % Gaussian inhomogeneous
%broadening effect

S_plus;  % Measurement matrix Sc+ = (Sx + iSy)
%in the original basis (|SzIz>)
S_minus; % Measurement matrix Sc- = (Sx - iSy)
%in the original basis (|SzIz>)

    end

    methods

function this = Signal_class(hamiltonian,...
density, pulse, fED, pathway, ...
signalType, tv1, tv2, thetas, phis,...
```

```
withGaussian, delta, signalFile)


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


this.SignalType = signalType;


this.Tv1 = tv1;
this.Tv2 = tv2;
this.Thetas = thetas;
this.Phis = phis; %
this.WithGaussian = withGaussian;
this.Delta = delta;
this.SignalFile = signalFile;


this.S_plus = zeros(4);
this.S_plus(1,3) = complex(1.0, 0.0);
this.S_plus(2,4) = complex(1.0, 0.0);


this.S_minus = zeros(4);
this.S_minus(3,1) = complex(1.0, 0.0);
this.S_minus(4,2) = complex(1.0, 0.0);
        end % Signal_class constructor


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,
(XXX) %d/%d), %d of %d\n', 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)


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.Cal2Pulses(pulses, ...
spinPathway, signals);
                end % for j = 1:n
            end % for i = 1:m


 signals = signals /(max(max(abs(signals))));
dlmwrite(this.SignalFile, signals);
```

```
        end

function signals = Cal2Pulses(this, pulses,...
spinPathway, signals)

   % 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.Pulse.DensityEvolve(pulses{2});
this.Density.ToHBasis();

this.Pathway.SelectPath(spinPathway(2));
this.Density.ToNormalBasis();

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
%broadening effect
     delta = this.Delta;
g = exp(-2 * (pi * delta * (t2))^2);
signal = g*signal;
   end

 signals(i,j) = signals(i,j) + signal;
 %fprintf(fileId,'%12.5e\t', signal);
                end
   %fprintf(fileId,'\n');
```

```
            end

            %fclose(fileId);

        end %function Output2Pulses

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 signals=Cal3Pulses(this, pulses,...
 t, spinPathway, signals)

% pulsepar(2): (phase, tipAngle) pairs
            tv1 = this.Tv1;
            tv2 = this.Tv2;

            n1 = length(tv1);
            n2 = length(tv2);
this.Density.ToNormalBasis();
this.Pulse.DensityEvolve(pulses{1});
this.Density.ToHBasis();
this.Pathway.SelectPath(spinPathway(1));
this.Density.ToNormalBasis();
densityLst1 = this.Density.DensitySzIzL;
%fileId = fopen(signalFile,'wt');
            for i = 1: n1

this.Density.DensitySzIzL = densityLst1;

                t1 = tv1(i);
this.FED.DensityEvolve(t1);

this.Pulse.DensityEvolve(pulses{2});
this.Density.ToHBasis();
this.Pathway.SelectPath(spinPathway(2));
this.Density.ToNormalBasis();
this.FED.DensityEvolve(t);

this.Pulse.DensityEvolve(pulses{3});
this.Density.ToHBasis();
this.Pathway.SelectPath(spinPathway(3));
this.Density.ToNormalBasis();
```

```
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
%broadening effect
   delta = this.Delta;
g = exp(-2 * (pi * delta * (t2))^2);
         signal = g*signal;
end
signals(i,j) = signals(i,j) + signal;
%fprintf(fileId,'%12.5e\t', signal);
  end
  %fprintf(fileId,'\n');
            end

            %fclose(fileId);

        end %function Output3Pulses

function [signal] = Measure (this)
density = ...
this.Density.GetDensityInSzIzBasisH();
measurementMat =...
this.Hamiltonian.SzIzToH0BasisH(this.S_plus);
  switch this.SignalType
                case 1

measurementMat = this.S_plus;
                case 2
measurementMat = this.S_plus;
                otherwise
disp('No such Signal Type, thus output default
(XXX) Sc+')
            end %switch SignalType

signal = trace( measurementMat * density);
        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 % methods
```

```
end %classdefSignal_class< handle
```

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

    properties
        Fs1;   % Sampling frequency
        Dt1;  % Sample time, unit: Microsecond
        N1;         % Length of signal
        Tv1;   % Time vector, unit: Microsecond

        Df1; % Frequency increment, unit: MHz
        Fv1; % Frequency vector, unit: MHz

        Fs2;   % Sampling frequency
        Dt2; % Sample time, unit: Microsecond
        N2;         % Length of signal
        Tv2;  % Time vector, unit: Microsecond

        Df2; % Frequency increment, unit: MHz
        Fv2; % Frequency vector, unit: MHz

SignalFile; % Name of signal data file
        Signal2d;   % Sampled data
        Signal2d_fft; % Fourier transform (FT)

PlotExistData = false;
    end %properties

    methods

        % constructor
function this = Plot_class(fs1, fs2,...
plotExistData)
this.PlotExistData = plotExistData;
this.Init(fs1, fs2);
        end %  Plot_class constructor this

        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;
%Time vector, unit: Microsecond
df1 = fs1/n1; % Frequency increment, unit: MHz
fv1 = (-n1/2:n1/2-1)*(df1);

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

df2 = fs2/n2;% Frequency increment, unit: MHz
fv2 = (-n2/2:n2/2-1)*(df2); % Frequency vector

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.Fv1 = fv1; % Frequency vector, unit: MHz

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)

this.SignalToFFT(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

tv1 = this.Tv1; % Time vector

fv1 = this.Fv1;


            n2 = this.N2;   % 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);


            signal2d_fft = this.Signal2d_fft;

signal2d_fft = signal2d_fft...

/(max(max(signal2d_fft)));

signal2d_fft = abs(signal2d_fft);


            if withTranspose

                signal2d = transpose(signal2d);

  signal2d_fft = transpose(signal2d_fft);

            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 (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 (Frequency-domain)'...
,'Visible','On');
                plot(fv2,abs(y2_fft))
title('1D ESEEM (Frequency-domain)')
xlabel('F2 (MHz)')
ylabel('Intensity')
            end % if plotAll == true

plotTimeDomain = true;
            if plotTimeDomain == true
hFig = figure('Name','2D(Time-domain)'...
,'Visible','On');
mesh(tv1,tv2, signal2d)
xlabel('T1 (\mus)') % Microsecond
ylabel('T2 (\mus)') % Microsecond
zlabel('Intensity')
                if withTitle == true
title(['{\bf', 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
```

```
title(['{\bf', titleName, '}'])
                end
set(gca,'XTick',-this.N1/2:50:this.N1/2)
set(gca,'YTick',-this.N2/2:50:this.N2/2)
view(45, 22.5);% view(3) sets the default
%three-dimensional view, az = -37.5, el = 30.

                if(~this.PlotExistData)
saveas(hFig, strcat(fileName, '_fd', '.pdf'));
saveas(hFig, strcat(fileName, '_fd', '.fig'));
% save fig file
                end
end % if plotFrequencyDomain == true
        end % function Plot

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);

                    % General rotations
                otherwise
                    degree = s*degree;

   [nrows, ncols] = size(inmatrix);
midx=ceil((ncols+1)/2);
midy=ceil((nrows+1)/2);

   a = degree*pi/180;
Mr = [cos(a) sin(a); -sin(a) cos(a)];

                        % rotate about center
[X, Y] = meshgrid(1:ncols,1:nrows);
XYt = [X(:)-midx Y(:)-midy]*Mr;
XYt = bsxfun(@plus,XYt,[midxmidy]);

xout = round(XYt(:,1));
yout = round(XYt(:,2));
xout(xout<1) = 1;
xout(xout>ncols) = ncols;
yout(yout<1) = 1;
yout(yout>nrows) = nrows;
outmatrix =...
inmatrix(sub2ind(size(inmatrix),yout,xout));
outmatrix = reshape(outmatrix,size(inmatrix));
            end %switch
end % function RotationZ
    end %method
end %classdefPlot_class
```

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
```

```
    % 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,...
 'yes');


this.RecordItem(name, str);


        end % InputYes


function result = OutputFileName(this, name,...
nThetas, nPhis, withRelaxation, extension)
            % 'signaldqcnThetas_nPhis_withRe
%laxation.txt';
result = strcat(name, num2str(nThetas), '_',...
num2str(nPhis), '_');
            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)
            % 'signaldqcnThetas_nPhis_withRe-
%laxation.txt';
result = strcat(name, ' Thetas =',...
num2str(nThetas), ', Phis =', ...
num2str(nPhis), ',');
            if withRelaxation
result = strcat(result, ' Relaxation = yes,');
            else
```

```
result = strcat(result, ' Relaxation = no,');
            end
result = strcat(result, datestr(now,...
' mmmm dd, yyyy HH:MM:SS'));

this.RecordItem('Plot Title', result);
        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 ReadData(this, fileName)

            fid = fopen(fileName);
allData = textscan(fid,'%s','Delimiter','\n');
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 %if

                k = strfind(str,startChar);
                if isempty(k)
                    continue;
                end

                p = k(1,1);
                if (length(str) == p)
                    continue;
                end


name = upper(strtrim(str(1:p-1)));
                value = str(p+1:end);

this.DataMap(name) = value;

            end %for
fclose(fid);

        end % ReadData

        function result = GetData(this, name)
            key = strtrim(name);
            if(isempty(key))
error('Error: \n%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. \nNo values exist for %s.',name)
            end

            switch key
```

```
                case 'TITLENAME'
                    result = value;
                case 'USERINPUT'
    result = this.ParseYes(name, value);
                case 'WITHRELAXATION'
   result = this.ParseYes(name, value);
                case 'FS1'
result = this.ParseNum(name, value, 1);
                case 'FS2'
     result = this.ParseNum(name, value, 1);
                case 'PLOTEXISTDATA'
 result = this.ParseYes(name, value);
                case 'B0'
  result = this.ParseNum(name, value, 1);
                case 'W_N'
   result = this.ParseNum(name, value, 1);
                case 'G_TENSOR'
   result = this.ParseNum(name, value, 3);
                case 'HYPERFINE_TENSOR'
  result = this.ParseNum(name, value, 3);
                case 'T2E'
   result = this.ParseNum(name, value, 1);
                case 'T2N'
   result = this.ParseNum(name, value, 1);
                case 'WE'
   result = this.ParseNum(name, value, 1);
                case 'WN'
  result = this.ParseNum(name, value, 1);
                case 'WX'
 result = this.ParseNum(name, value, 1);
                case 'WY'
 result = this.ParseNum(name, value, 1);
                case 'WHE'
  result = this.ParseNum(name, value, 1);
                case 'PULSETIME'
  result = this.ParseNum(name, value, 1);
                case 'PULSETYPE1'
 result = this.ParseNum(name, value, 2);
                case 'PULSETYPE2'
result = this.ParseNum(name, value, 1);
                case 'SIGNALTYPE'
result = this.ParseNum(name, value, 1);
                case 'THETAMIN'
result = this.ParseNum(name, value, 1);
                case 'THETAMAX'
result = this.ParseNum(name, value, 1);

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

                case 'WITHGAUSSIAN'
 result = this.ParseYes(name, value);
                case 'DELTA'
result = this.ParseNum(name, value, 1);
                case 'ROTATIONAXIS'
result = this.ParseNum(name, value, 1);
                case 'ROTATIONANGLE'
result = this.ParseNum(name, value, 1);

                case 'SPINPATHWAY2'
result = this.ParseNum(name, value, 2);
                case 'SPINPATHWAY3'
 result = this.ParseNum(name, value, 3);
                case 'FEDTIME'
result = this.ParseNum(name, value, 1);
                case 'ROTATEDEGREE'
result = this.ParseNum(name, value, 1);
                case 'WITHTITLE'
```

```
  result = this.ParseYes(name, value);
                case 'WITHTRANSPOSE'
  result = this.ParseYes(name, value);
                case 'PLOTALL'
  result = this.ParseYes(name, value);
                case 'RNDOFF'
  result = this.ParseNum(name, value, 1);

               otherwise
error('Error. \nNo values exist for %s.',...
name);
            end %switch path

        end

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


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


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

        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\n', 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

% uint: MHz
hyperfine_tensor: -61.0d0, -91.0d0, -29.0d0

%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
%ralaxation rates, unit: 1/microsecond
we: 1.67E-02
% Diagonal Lattice induced nuclear-spin flip
%ralaxation rates, unit: 1/microsecond
wn: 7.14E-03
% Diagonal Cross relaxation, unit:
%1/microsecond
wx: 6.17E-03
% wy = wx; Diagonal Diagonal Cross relaxation,
%unit: 1/microsecond
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;
% (phase, tipAngle) pair
pulsetype1: 0.0d0,  pi/2.0
pulsetype2: 0.0d0,  pi

% 1 - S+=(sx + iSy), 2 - S-=(Sx - iSy)
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;


%withGaussian: 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 (ESEEM) spectroscopy as a tool to investigate the coordination environment 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, Vinck 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. Hogben HJ, Krzystyniak 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. Gemperle 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).