Freie Universität Berlin

Bachelorarbeit Juli 2012

Large Scale Parallel Simulation of EPR Lineshape Spectra

Abstract: *Electron Paramagnetic Resonance* is a spectroscopy method to investigate systems of unpaired electron spins. *Spinach* is a Matlab library to simulate different kinds of spin system experiemnts, including EPR. This paper aims to give an overview of the theoretical concepts involved in spin dynamics, especially those necessary to comprehend Spinach's way of numerical computation. After understanding the parameters determining Spinach's resources usage we adopt parallel computing methods on Linux clusters to accelerate the calculation of EPR lineshape spectra.

Author: Johannes Hörmann

Supervisor: Hossam Elgabarty

Contents

1 Theory of $\frac{\pi}{2}$-pulsed EPR

1.1 Introduction

The mechanisms of *EPR (Electron Paramegnetic Resonance)*, also called *ESR (Electron Spin Resonance)*, work analogous to the mechanisms of *NMR (Nuclear Magnetic Resonance)*. In *diamagnetic* materials, all electrons are spin-paired, making the elctron magnetic dipole vanish, and enabling the system to be accessible by NMR. EPR experiments are suitable to investigate *paramagnetic* systems with unpaired electron spins, which exhibit a non-zero electron spin. The basic idea is to perturb a spin system's equilibrium by a small pulsed oscillating magnetic field: When we place a *powder sample* of spin sytems inside a static magnetic field $\vec{B}_0 = B_0\hat{z}$ and let it settle to equilibrium, due to Zeeman effect more electron spins are goint to align parallel to $\vec{B}_0$ than antiparallel, resulting in a net magnetization of the sample. When speaking of a powder sample we mean a sample in which many spin systems exist in randomly distributed spatial orientations. A magnetic pulse $\vec{B}_1(t) = B_1(t)\hat{x}$ linearly polarized in x-direction will tilt the electron spins and disturb the equilibrium in a way we are going to examine in the course of this article. The resulting time-dependent magnetization during the relaxation process may be recorded. This time-resolved signal is called the *free induction decay (FID)*.

1.2 Spin interactions

1.2.1 Spin system Hamiltonian

Like in any other quantum mechanical problem the starting point when treating an EPR experiment theoretically is the Schrödinger equation[1]

$$i\hbar\frac{\partial}{\partial t}|\psi\rangle = \mathcal{H}|\psi\rangle \tag{1}$$

For a start we examine the Hamiltonion of an unpaired spin electron system in a static magnetic field. It will exhibit several perturbation terms of different nature:

$$\mathcal{H} = [\mathcal{H}_{EZ} + \mathcal{H}_{ECS} + \mathcal{H}_{LS}] + [\mathcal{H}_{HF}] + [\mathcal{H}_{NZ} + \mathcal{H}_{NCS} + \text{weaker interactions}] \tag{2}$$

1. Electron Zeeman contribution $\mathcal{H}_{EZ} = -\vec{\mu}\cdot\vec{B}_0 = -\hbar B_0(\gamma_L\mathbf{L}_z + \gamma_S\mathbf{S}_z)$

   The static magnetic field $\vec{B}_0 = B_0\hat{z}$ acts a torque on the electron's magnetic dipole moment μ, linearly dependent on its angular momentum and spin. Thus the *Zeeman effect* lifts the spin degeneracy of energy levels, reducing the energy of spins aligned parallel to the magnetic field (-), and increasing the energy of spins aligned antiparallel (+) by the correction term[2]

   $$E^1_\pm = \pm\mu_B\gamma_J B_0 m_J \tag{3}$$

2. Electron chemical shift $\mathcal{H}_{ECS} = \gamma_S\hbar\mathbf{S}\cdot\sigma_S(t)\cdot\vec{B}_0$

   The moving electron clouds change the effective magnetic field $\vec{B}_{\text{eff}}$ "seen" by the every electron spin. This "shielding" behaviour is desctribed by the *chemical shift tensor* $\sigma_S(t)$:

   $$\vec{B}_{\text{eff}}(t) = -\sigma_S(t)\cdot\vec{B}_0 \tag{4}$$

3. Spin-orbit coupling $\mathcal{H}_{LS} = \lambda\mathbf{L}\cdot\mathbf{S}$

   The electron's motion around the nucleus creates a magnetic field, with which the electron's spin will interact. Together with the Zeeman interaction, those two Hamiltonian contributions are due to the influence of a magnetic field. Furthermore, the externam Zeeman field causes $\vec{L}$ to change, thus also influencing the spin-orbit interaction.

[1] Following spin dynamics conventions in the course of this article we set $\hbar = 1$

[2] for a thorought derivation of the energy correction term see [6, chap. 6.4 The Zeeman Effect, p. 277ff]

4. Hyperfine interaction $\mathcal{H}_{HF} = \frac{\gamma_I \gamma_S \hbar^2}{r^3} \left[\frac{3(\mathbf{I}\cdot\vec{r}(t))(\mathbf{S}\cdot\vec{r}(t))}{r^2} - \mathbf{I}\cdot\mathbf{S} \right]$

   The nucleus interacts with the orbiting electron due to the electron's induced magnetic field acting on the nuclear magnetic dipole moment.

5. Nuclear Zeeman contribution $\mathcal{H}_{NZ} = -\hbar B_0 \gamma_I \mathbf{I}_z$

   Just like the electron, the nucleus posseses intrinsic spin and thus a nuclear magnetic moment, enabling it to interact with an external magnetic field.

6. Nuclear chemical shift $\mathcal{H}_{NCS} = \gamma_I \hbar \mathbf{I} \cdot \sigma_I(t) \cdot \vec{B}_0$

   Of course, the shielding by the electron clouds also applies to the nuclei's spins $\mathbf{I}$.

$$\vec{B}_{\text{eff}}(t) = -\sigma_I(t) \cdot \vec{B}_0 \tag{5}$$

7. Other weaker interactions like coupling of the nuclear quadrupole moment to the electron's electromagnetic field and the magnetic coupling of electrons with each other or nuclei with each other.

1.2.2 g-tensor and A-tensor

In the spin Hamiltonian above different interactions have been grouped with square brackets into three packages. The latter package $[\mathcal{H}_{NZ} + \mathcal{H}_{NCS} + \text{weaker interactions}]$ simply marks interactions which we are allowed to neglect in the case of high field EPR. When the external magnetic field $\vec{B}_0$ becomes sufficiently strong, their contribution diminishes in comparison with the interactions depending on $\vec{B}_0$.
The former package $[\mathcal{H}_{EZ} + \mathcal{H}_{ECS} + \mathcal{H}_{LS}]$ marks major interactions linear in spin. In EPR the overall behaviour of those interactions is summarized in the *g-tensor*:

$$\mathcal{H}_{\text{linear}} = \mu_B \, \mathbf{S} \cdot g \cdot \vec{B}_0 \tag{6}$$

The second package only including the hyperfine interaction characterizes the term bilinear in spin: the coupling between one spin and another. Though not evident from the sketch above, but those interactions are anisotropic in general and thus summarized by the A-tensor in the case of EPR:

$$\mathcal{H}_{\text{bilinear}} = \mathbf{S} \cdot A \cdot \mathbf{I} \tag{7}$$

One might wonder, why the spin-orbit interaction does not contribute to the bilinear part. In [8, chap 11.2, p. 505ff] one finds a dedicated explanation why spin-orbit coupling results in a changed effective magnetic field and thus rather contributes to the g-tensor, very much like the chemical shift. Another possible self-interaction $\mathcal{H}_{\text{quadratic}} = \mathbf{S} \cdot A \cdot \mathbf{S}$ we are not going to encounter in the case of high field EPR.

1.3 Spin formalism

Our aim is to understand the mechanisms employed in Spinach's simulation, and we are especially curious about the scaling of Spinach's memory and CPU time consumption. Hence we will examine the mathematical formalism directly underlying numerical computation more thoroughly.

1.3.1 Quantum states and measurments done on them

Any allowed spin state $|\psi\rangle$ can be written as a linear superposition of an orthogonal basis set of a Hilbert space spanned by all allowed *azimuthal quantum number* states $|m\rangle$:

$$|\psi\rangle = \sum_m a_m |m\rangle \tag{8}$$

where the amplitudes are complex $a_m = |a_m|e^{i\phi_m}$ with phase ϕ_m and magnitude $|a_m|$. The $|m\rangle$ can be represented by a proper scaled basis of choice, but the m lable offers a general independent representation.
All measurements to be done on a spin system yield eigenvalues of a linear operator associated with the particular measurement. The corresponding observed physical quantity is called *observable*. Measuring the spin component of a system in one of the basis states along the z-axis $\mathbf{S_z}$ thus yields

$$\mathbf{S_z}|m\rangle = m|m\rangle \tag{9}$$

The orthogonal basis can be normalized by requiring the inner product of basis vectors to be

$$\langle m|m'\rangle = \delta_{mm'} \tag{10}$$

If a spin system exists in the eigenstate $|m\rangle$ of $\mathbf{S_z}$, then the measurement of $\mathbf{S_z}$ will yield

$$\langle m|\mathbf{S_z}|m\rangle = m \tag{11}$$

The measurement on a general superposition will yield

$$\langle\psi|\mathbf{S_z}|\psi\rangle = \sum_{m,m'} a_m^* a_{m'} \langle m'|\mathbf{S_z}|m\rangle \tag{12}$$

$$= \sum_{m,m'} a_m^* a_{m'} m\langle m'|m\rangle \tag{13}$$

$$= \sum_{m} |a_m|^2 m \tag{14}$$

due to the orthormality of the basis set.

1.3.2 Evolution operator and time propagation

In the following we shall make use of matrix exponentials to express some quantum mechanical operators. The defintion of the exponential

$$e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!} = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + ... \tag{15}$$

can be easily applied to square matrices, eg.:

$$e^{i\phi A} = I + i\phi A - \frac{(\phi A)^2}{2!} - i\frac{(\phi A)^3}{3!} + \frac{(\phi A)^4}{4!} + i\frac{(\phi A)^5}{5!} - ... \tag{16}$$

such that exponentials are defined for matrices as well.
In case of a stationary Hamiltonian, the Schrödinger equation

$$i\frac{\partial}{\partial t}|\psi(t)\rangle = \mathcal{H}|\psi(t)\rangle \tag{17}$$

has the solution

$$|\Psi(t)\rangle = U(t)|\psi(0)\rangle \tag{18}$$

where

$$U(t) = e^{-i\mathcal{H}t} \tag{19}$$

is called the *evolution operator*.
In case of a time-dependent, but piecewise-constant Hamiltonian the solution has the form

$$|\Psi(t)\rangle = \left[\prod_k e^{-i\mathcal{H}_k \Delta t_k}\right] |\Psi(0)\rangle \tag{20}$$

This is the basis of numerical time propagation.

1.3.3 Density operator and spin ensembles

Due to equation (8) the expectation value of an observable A can be expressed as

$$\langle A\rangle = \langle\psi|A|\psi\rangle = \sum_{m,n} c_m^* c_n \langle m|A|n\rangle \tag{21}$$

Now, when the expectation value of a certain observable is required, we are always interested in the product $c_m^* c_n$ rather than the distinct c_n, thus we can think of a matrix representation of those probabilities and define an operator P with

$$\langle n|P|m\rangle = c_n c_m^* \tag{22}$$

Equation (21) becomes

$$\langle A\rangle = \sum_{m,n} \langle n|P|m\rangle\langle m|A|n\rangle \tag{23}$$

Since $|m\rangle$ form an orthonormal basis set, the results of A and P acting on a basis vector can be expanded in the basis:

$$P|m\rangle = \sum_l a_l|l\rangle = \sum_l \langle l|P|m\rangle|l\rangle \tag{24}$$

$$A|n\rangle = \sum_m b_m|m\rangle = \sum_m \langle m|A|n\rangle|m\rangle \tag{25}$$

since the a_l and b_a may be found as

$$\langle l|P|m\rangle = \sum_{l'} a_{l'}\langle l'|l\rangle = a_l \tag{26}$$

$$\langle m|A|n\rangle = \sum_{m'} b_{m'}\langle m'|m\rangle = a_m \tag{27}$$

$$\tag{28}$$

allowing the following expression to be rewritten

$$\Rightarrow PA|n\rangle = \sum_m P|m\rangle\langle m|A|n\rangle \tag{29}$$

$$= \sum_{l,m} |l\rangle\langle l|P|m\rangle\langle m|A|n\rangle \tag{30}$$

$$\Rightarrow \langle n|PA|n\rangle = \sum_{l,m} \langle n|l\rangle\langle l|P|m\rangle\langle m|A|n\rangle \tag{31}$$

$$= \sum_{l,m} \delta_{ln}\langle l|P|m\rangle\langle m|A|n\rangle \tag{32}$$

$$= \sum_m \langle n|P|m\rangle\langle m|A|n\rangle \tag{33}$$

Comparing with equation (23) we find

$$\langle A\rangle = \sum_n \langle n|PA|n\rangle = Tr(PA) = Tr(AP) \tag{34}$$

where Tr is the trace – the total sum of the matrix' diagonal elements. P is an Hermitian operator since

$$\langle n|P|m\rangle = c_n c_m^* = (c_m c_n^*)^* = \langle m|P|n\rangle^* \tag{35}$$

and we might as well write

$$\langle A\rangle = Tr(P^\dagger A) = Tr(A^\dagger P) \tag{36}$$

In EPR we pulse a powder sample. Theoretically this means a measurement on an ensemble of many spin systems with many (most generally different) spin states $|\psi\rangle$ instead of determining the state of a single system. The observable averaged about the whole statistical ensemble is written as

$$\begin{aligned}
\overline{\langle A\rangle} &= \overline{\langle\psi|A|\psi\rangle} && (37)\\
&= \sum_\psi p_\psi \langle\psi|A|\psi\rangle && (38)\\
&= \sum_\psi p_\psi \left(\sum_{m,n} c_m^* c_n \langle m|A|n\rangle\right) && (39)\\
&= \sum_{m,n} \left(\sum_\psi p_\psi c_m^* c_n\right) \langle m|A|n\rangle && (40)\\
&= \sum_{m,n} \overline{c_m^* c_n} \langle m|A|n\rangle && (41)
\end{aligned}$$

where p_ψ is an appropriate statistical averaging weight chosen according to the occupancy of $|\psi\rangle$. On this basis we introduce the *density matrix operator*

$$\langle n|\rho|m\rangle = \overline{c_m^* c_n} = \overline{\langle n|P|m\rangle} \tag{42}$$

whereby equation (41) can be expressed in analogy to equation (34) as

$$\overline{\langle A\rangle} = \sum_{n,m} \langle n|\rho|m\rangle\langle m|A|n\rangle = \sum_n \langle n|\rho A|n\rangle = Tr(\rho^\dagger A) = Tr(A^\dagger \rho) \tag{43}$$

Another beautiful expression for the density matrix can be derived by noticing that

$$\begin{aligned}
\langle n|\psi\rangle\langle\psi|m\rangle &= \langle n| \left(\sum_{n'} c_{n'}|n'\rangle\right) \left(\sum_{m'} c_{m'}^* \langle m'|\right) |m\rangle && (44)\\
&= \sum_{n',m'} c_{n'} c_{m'}^* \langle n|n'\rangle\langle m'|m\rangle = c_n^* c_m = \langle n|P|m\rangle && (45)\\
\Rightarrow |\psi\rangle\langle\psi| &= P \quad ; \quad \overline{|\psi\rangle\langle\psi|} = \rho && (46)
\end{aligned}$$

1.3.4 Liouville equation

Under comparison with the general Schrödinger equation and its complex conjugate below

$$\begin{aligned}
\langle\psi|\frac{\partial}{\partial t}|\psi\rangle &= -i\langle\psi|\mathcal{H}|\psi\rangle && (47)\\
\left(\frac{\partial}{\partial t}\langle\psi|\right)|\psi\rangle &= \sum_{m,n} \frac{\partial c_m^*}{\partial t} c_n \langle m|n\rangle = \sum_m \frac{\partial c_m^*}{\partial t} c_m && (48)\\
&= \left(\sum_m c_m^* \frac{\partial c_m}{\partial t}\right)^* = \left(\langle\psi|\frac{\partial}{\partial t}|\psi\rangle\right)^* = i\langle\psi|\mathcal{H}|\psi\rangle && (49)
\end{aligned}$$

the density matrix' equation of motion can be derived by differentiating equation (46) with respect to time:

$$\frac{\partial}{\partial t}\rho = \left(\frac{\partial}{\partial t}|\psi\rangle\right)\langle\psi| + |\psi\rangle\left(\frac{\partial}{\partial t}\langle\psi|\right) \tag{50}$$

$$= -i\mathcal{H}|\psi\rangle\langle\psi| + i|\psi\rangle\langle\psi|\mathcal{H} \tag{51}$$

$$= -i(\mathcal{H}\rho - \rho\mathcal{H}) = -i[\mathcal{H}, \rho] \tag{52}$$

We arrived at the *Liouville equation*. Its general solution follows directly from the Schrödinger equation's solution (18):

$$\rho(t) = |\psi(t)\rangle\langle\psi(t)| = e^{-i\mathcal{H}t}|\psi(0)\rangle\langle\psi(0)|e^{i\mathcal{H}t} = e^{-i\mathcal{H}t}\rho(0)e^{i\mathcal{H}t} \tag{53}$$

1.3.5 Superoperators, Liouville space

The Hamiltonian $\mathcal{H}$ is an operation defined on the space of vectors, similarly the commutation operator $\mathcal{L} = [\mathcal{H}, \cdot]$ is an operation defined on the space of operators. Thus it is called the *Liouvillian superoperator*. The spaces one usually comes to deal with when treating spin dynamics are

- the space where all states live. States are represented as a vector (8) of dimension N, and with the scalar product $\langle\phi|\psi\rangle$ they fulfill every requirement to span an N-dimensional *Hilbert space*. This space is isomorph to $\mathbb{C}^N$.
- the space where all operators which act on spin states live. Operators such as $\mathcal{H}$ are represented by $N \times N$-matrices, thus the space spanned by those operators is N^2-dimensional and isomorph to $\mathbb{C}^{N^2}$. There exist many possible scalar products, so again we deal with a Hilbert space. The special operator space, where we choose $Tr(A^\dagger B)$ as the scalar product is known as the *Liouville space* in spin dynamics.
- the space where all *superoperators* which act on operators live. Superoperators such as $\mathcal{L} = [\mathcal{H}, \cdot]$ are represented by $N^2 \times N^2$ matrices, thus the space spanned by those superoperators is N^4-dimensional and isomorph to $\mathbb{C}^{N^4}$. Again, we chose the scalar product $Tr(A^\dagger B)$ for this Hilbert space and call it *superoperator space*.

Now, if we represent states as vectors and operators as matrices, how come we do not have to represent superoperators as more complex objects like "three-dimensional matrices"? The mathematical trick is to stretch the original operator matrix ρ column-wise into a vertical vector:

$$\rho = \left(\vec{\rho}_1, \vec{\rho}_2, \ldots, \vec{\rho}_N\right) \rightarrow |\rho\rangle = \begin{pmatrix} \vec{\rho}_1 \\ \vec{\rho}_2 \\ \cdots \\ \vec{\rho}_N \end{pmatrix} \tag{54}$$

Of course, every single column vector of ρ has dimension N, so $|\rho\rangle$ has dimension N^2. The trace of two matrices A, B stretches into a vector scalar product:

$$Tr(A^\dagger B) = \sum_{n,k} A^*_{nk} B_{nk} = \sum_{n'} A^*_{n'} B_{n'} = \langle A|B\rangle \tag{55}$$

This allows the calculation of a matrix scalar product in $O(N^2)$ runtime.
With the operator stretched into a vector, the superoperator can accordingly be stretched into an $N^2 \times N^2$ matrix representation. The matrix version of the Liouvillian is the sum of two Kronecker products

$$\mathcal{L} = E \otimes \mathcal{H}^T - \mathcal{H} \otimes E \tag{56}$$

where E is the $N \times N$ unity matrix.
For visualization, a spin-$\frac{1}{2}$ system with the two-dimensional Hilbert space basis consisting of $|-\frac{1}{2}\rangle$ and $|\frac{1}{2}\rangle$ yields a four-dimensional Liouville space, and as its orthonormal basis we might choose $\frac{1}{2}E, J_x, J_y$ and J_z, or $E, I_z, -\frac{1}{\sqrt{2}}J_+$ and $\frac{1}{\sqrt{2}}J_-$, which are all 2×2 matrices. Similarly a spin-1 system with the three-dimensional Hilbert space spanned by $|-1\rangle, |0\rangle$ and $|1\rangle$ requires a nine-dimensional Liouville basis, and in general a spin system of N possible states yields an N^2 dimensional Liouville space (and an N^4 dimensional superoperator space). The next section treats the question, how to choose an apropriate basis out of many orthonormal possibilities.

1.3.6 Choice of basis set and irreducible spherical tensors

When we are looking for a suitable basis of the Liouville space to conduct some spin dynamics simulation, the perfect choice would be *eigenoperators* S_k of the Liouvillian, since they are invariant under all interactions of the system:

$$\mathcal{L}S_k = l_k S_k \tag{57}$$

To find those eigenoperators in their vector form, one could diagonalize the Liouvillians matrix representation. Unfortunately the computation costs would be enourmous due to the N^4 dimensions of the superoperator space. A wise choice would be a set of eigenoperators invariant under commutation with J_z, since this operator characterizes the dominating Zeeman interaction in EPR. Or even better, a set of operators invariant under rotations, since we have to examine the spin system from many different angles to average the powder spectrum.
As we know, the angular momentum operators J_x, J_y and J_z generate rotations. It is possible to define a family of operators T_{lm} called *irreducible spherical tensors (IST)* of rank l and order m, which fulfill the commutation relations

$$[J_\pm, T_{lm}] = \sqrt{l(l+1) - m(m \pm 1)}\, T_{l(m\pm1)} \tag{58}$$

$$[J_z, T_{lm}] = m T_{lm} \tag{59}$$

In the space of operators they are the analogon to the spherical harmonics in the space of wavefunctions, for every l there exist $2l+1$ independent IST with $m = -l, -l+1, \ldots, l-1, l$ and any rotation transforms T_{lm} into a linear combination of $T_{lm'}$ with same rank l:

$$R(\alpha, \beta, \gamma)\ T_{lm} = \sum_{m'=-l}^{l} \mathfrak{D}^{(l)}_{m',m}(\alpha, \beta, \gamma)\ T_{lm'} \tag{60}$$

The matrix elements $\mathfrak{D}^{(l)}_{m,m'}(\alpha.\beta, \gamma)$ are called *Wigner functions* and make up the *Wigner rotation matrix* of rank l corresponding to a certain rotation $R(\alpha, \beta, \gamma)$. They can be evaluated with the spherical harmonics

$$\mathfrak{D}^{(l)}_{m,m'}(\alpha.\beta, \gamma) = \langle Y_{lm} | R(\alpha, \beta, \gamma) | Y_{lm'} \rangle \tag{61}$$

Just as the spherical harmonics Y_{lm} form an orthonormal basis on the unit sphere of the wave functions' Hilbert space, do all IST T_{lm} form an orthonormal basis on the Liouville space of operators, and thus any operator, e.g. the density matrix ρ of dimension N^2, may be expanded in a linear combination of IST:

$$\rho = \sum_{l=0}^{N-1} \sum_{m=-l}^{l} a_{lm} T_{lm} \tag{62}$$

Obviously, a spin-$\frac{1}{2}$ density matrix of dimension $N^2 = 4$ can be fully represented by the single zero-order IST and the three first-order IST, while for a $N^2 = 9$ Liouville space all IST of second rank are required as well.

The lowest rank IST is the identity operator, the first rank IST are linear superpositions of J_x, J_y and J_z. All higher rank IST $T_{LM}(J_1, J_2)$ can be formed as products of lower rank IST following the rule for combining angular momenta:

$$T_{LM}(J_1, J_2) = \sum_{m_1,m_2} \langle l_1 m_1 l_2 m_2 | LM \rangle \; T_{l_1 m_1}(J_1) \; T_{l_2 m_2}(J_2) \tag{63}$$

where the *Clebsch-Gordan coefficients* $\langle l_1 m_1 l_2 m_2 | LM \rangle$ are zero except for $m_1 + m_2 = M$. K ranges from $|k_1 - k_2|$ to $k_1 + k_2$. The new IST again obey the rotational features above. Taking $k_1 = k_2 = 1$ we can produce the nine tensors below for $J_1 = J_2 = J$

$$\begin{array}{llll} & & T_{2-2} & = \frac{1}{2} J_-^2 \\ & T_{1-1} = \frac{1}{\sqrt{2}} J_- & T_{2-1} & = \frac{1}{2}(J_z J_- + J_- J_z) \\ T_{00} = E & T_{10} = J_z & T_{20} & = \frac{1}{\sqrt{6}}(3J_z^2 - 2E) \\ & T_{11} = -\frac{1}{\sqrt{2}} J_+ & T_{21} & = -\frac{1}{2}(J_z J_+ + J_+ J_z) \\ & & T_{22} & = \frac{1}{2} J_+^2 \end{array} \tag{64}$$

while for $J_1 \neq J_2$ we can produce 16 independent tensors, of which the second rank tensors read

$$\begin{aligned} T_{2-2} &= J_{1-} J_{2-} \\ T_{2-1} &= (J_{1z} J_{2-} + J_{1-} J_{2z}) \\ T_{20} &= \sqrt{\tfrac{2}{3}}(3 J_{1z} J_{2z} - J_1 \cdot J_2) \\ T_{21} &= -(J_{1z} J_{2+} + J_{1+} J_{2z}) \\ T_{22} &= -J_{1+} J_{2+} \end{aligned} \tag{65}$$

From there on it would be possible to construct IST for an arbitrary number of spins, but looking at the most general interaction included in an EPR Hamiltonian, the maximum number of interacting spins amounts to two, e.g. $\vec{L}$ and $\vec{S}$ coupled by the interaction tensor A

$$\vec{L} \cdot A \cdot \vec{S} = \sum_{k,n} a_{kn} L_k S_n \quad \text{with} \quad k, n = x, y, z \tag{66}$$

of which linear and quadratic interactions form special cases. Having a linear superposition of $L_k S_n$ terms, the interaction must be expandeble in a basis consisting of rank two IST at maximum. Practically, first rank terms' contribution diminishes and hence is ignored:

$$\vec{L} \cdot A \cdot \vec{S} = \sum_{l=0}^{2} \sum_{m=-l}^{l} \alpha_{lm} T_{lm}(L, S) \approx \alpha_{00} T_{00} + \sum_{m=-2}^{2} \alpha_{2m} T_{2m}(L, S) \tag{67}$$

Therefore the whole anisotropic Hamiltonian consisting of scalar-spin and spin-spin interactions of N spins may be expanded in second rank IST representation (ignoring quadratic self-interaction):

$$\mathcal{H} = \mathcal{H}_{\text{iso}} + \sum_{L} \sum_{m=-2}^{2} \alpha_{L,m} \; T_{2m}(L) + \sum_{L,S \neq L} \sum_{m=-2}^{2} \beta_{L,S,m} \; T_{2m}(L, S) \tag{68}$$

The special feature of this expansion is its behaviour under rotations $R(\alpha,\beta,\gamma)$ – the Hamiltonian inherits the IST's rotational transformation rule (60):

$$R\mathcal{H} = \mathcal{H}_{\text{iso}} + \sum_L \sum_{m=-2}^{2} \alpha_{L,m}\ RT_{2m}(L) + \sum_{L,S\neq L} \sum_{m=-2}^{2} \beta_{L,S,m}\ RT_{2m}(L,S) \tag{69}$$

$$= \mathcal{H}_{\text{iso}} + \sum_L \sum_{m=-2}^{2} \alpha_{L,m} \sum_{m'=-2}^{m'} \mathfrak{D}^{(2)}_{mm'} T_{2m'}(L) + \sum_{L,S\neq L} \sum_{m=-2}^{2} \beta_{L,S,m} \sum_{m'=-2}^{2} \mathfrak{D}^{(2)}_{mm'} T_{2m'}(L,S) \tag{70}$$

$$= \mathcal{H}_{\text{iso}} + \sum_{m=-2}^{2} \sum_{m'=-2}^{2} \mathfrak{D}^{(2)}_{mm'} \left(\sum_L \alpha_{L,m} T_{2m'}(L) + \sum_{L,S\neq L} \beta_{L,S,m} T_{2m'}(L,S) \right) \tag{71}$$

$$= \mathcal{H}_{\text{iso}} + \sum_{m,m'=-2}^{2} \mathfrak{D}^{(2)}_{mm'} Q_{mm'} \quad \text{with} \quad Q_{mm'} = \sum_L \alpha_{L,m} T_{2m'}(L) + \sum_{L,S\neq L} \beta_{L,S,m} T_{2m'}(L,S) \tag{72}$$

The 25 elements $Q_{mm'}$ of the 5×5 matrix Q are called *rotational basis operators*. They store the anisotropic part of any spin system's Hamiltonian in such a way that any rotational orientation of the system yields a linear combination of this rotational basis with precomputable Wigner function coefficients. Notice that each $Q_{mm'}$ has dimensions $N \times N$.

1.4 Understanding the experiment

1.4.1 Rotating reference frame

A magnetic field B_0 applied along the z-axis causes a magnetic moment to precess around the z-axis at the Larmor frequency ω_0. With this classical approach as a starting point it is possible to explain a fair amount of EPR phenomena, but since we want to get used to spin formalism, we will explain a spin system's behaviour from the quantum mechanical perspective right from the beginning. Since the electron's magnetic moment is proportional to its angular momentum $\vec{\mu} = \gamma\mathbf{J}$ with the *Landé g-factor* γ, the interaction energy $E = -\vec{\mu}\cdot\vec{B}$ and thus the Hamiltonian, the evolution operator and the Schrödinger equation's solution can be expressed as

$$\mathcal{H} = -\gamma B_0 \mathbf{J_z}, \quad U(t) = e^{i\gamma B_0 t \mathbf{J_z}}, \quad |\psi(t)\rangle = e^{i\gamma B_0 t \mathbf{J_z}} |\psi(0)\rangle = e^{i\omega_0 t \mathbf{J_z}} |\psi(0)\rangle \quad \text{with} \quad \omega_0 = \gamma B_0 \tag{73}$$

Analogous to the classical approach, the time dependent solution must be a rotation of the initial state by angle $\phi = -\omega_0 t$, and we can identify

$$\mathbf{R_z}(\phi) = e^{-i\phi \mathbf{J_z}} \tag{74}$$

as an rotation operator around the z-axis. $\phi > 0$ results in an *active* rotation of the state in "positive", anticlockwise direction, whereas $\phi < 0$ results in a rotation in "negative", clockwise direction. Likewise we can speak of $\phi > 0$ causing a *passive* rotation of the reference frame in negative, clockwise direction, whereas $\phi < 0$ rotates the reference frame in positive, anticlockwise direction.
If we want to determine an observable $\mathbf{A}$ of the rotated state $|\psi(t)\rangle$, we find

$$\langle\psi(\phi)|\mathbf{A}|\psi(\phi)\rangle = \langle\psi(0)|e^{-i\phi\mathbf{J_z}}\ \mathbf{A}\ e^{i\phi\mathbf{J_z}}|\psi(0)\rangle \tag{75}$$

$$= \langle\psi(0)|\mathbf{A}'(\phi)|\psi(0)\rangle \quad \text{with} \quad \mathbf{A}'(\phi) = e^{-i\phi\mathbf{J_z}}\ \mathbf{A}\ e^{i\phi\mathbf{J_z}} \tag{76}$$

it being the same as applying an rotated operator $\mathbf{A}'$ on the unrotated state $|\psi(0)\rangle$. Say $|\psi(\phi)$ is rotated in positive sense, then the operator $\mathbf{A}'$ must be rotated in the negative sense. The

two perspectives must correspond to two different reference frames. In the first case we observe a rotated system from the unrotated lab frame, whereas in the second case the axis of our reference frame are aligned with the spin system, such that it appears unrotated in our frame. Hence we can convert back and forth between resting frame and rotating frame:

$$|\psi\rangle = e^{-i\omega_1 t\mathbf{J_z}}|\psi'\rangle \tag{77}$$
$$|\psi'\rangle = e^{i\omega_1 t\mathbf{J_z}}|\psi\rangle \tag{78}$$

And if we know the Hamiltonian $\mathcal{H}$ for $|\psi\rangle$ in one frame, we can find the rotating Hamiltonian $\mathcal{H}'$ in the frame of $|\psi'\rangle$ by plugging into the Schrödinger's equation:

$$\frac{\partial}{\partial t}|\psi\rangle = \frac{\partial}{\partial t}\left(e^{-i\omega_1 t\mathbf{J_z}}|\psi'\rangle\right) = -i\omega_1 t\mathbf{J_z}\ e^{-i\omega_1 t\mathbf{J_z}}|\psi'\rangle + e^{-i\omega_1 t\mathbf{J_z}}\frac{\partial}{\partial t}|\psi'\rangle \tag{79}$$
$$i\frac{\partial}{\partial t}|\psi\rangle = \mathcal{H}|\psi\rangle \tag{80}$$
$$\Rightarrow i\left(-i\omega_1\mathbf{J_z}\ e^{-i\omega_1 t\mathbf{J_z}}|\psi'\rangle + e^{-i\omega_1 t\mathbf{J_z}}\frac{\partial}{\partial t}|\psi'\rangle\right) = \mathcal{H}\left(e^{-i\omega_1 t\mathbf{J_z}}|\psi'\rangle\right) \quad |\cdot e^{i\omega_1 t\mathbf{J_z}},\ e^{i\omega_1 t\mathbf{J_z}}\mathbf{J_z}e^{-i\omega_1 t\mathbf{J_z}} = \mathbf{J_z} \tag{81}$$
$$\Leftrightarrow \omega_1\mathbf{J_z}|\psi'\rangle + i\frac{\partial}{\partial t}|\psi'\rangle = \left(e^{i\omega_1 t\mathbf{J_z}}\mathcal{H}e^{-i\omega_1 t\mathbf{J_z}}\right)|\psi'\rangle \tag{82}$$
$$\Leftrightarrow i\frac{\partial}{\partial t}|\psi'\rangle = \left(e^{i\omega_1 t\mathbf{J_z}}\mathcal{H}e^{-i\omega_1 t\mathbf{J_z}} - \omega_1\mathbf{J_z}\right)|\psi'\rangle \tag{83}$$
$$= \mathcal{H}'|\psi'\rangle \tag{84}$$
$$\text{with} \quad \mathcal{H}' = e^{i\omega_1 t\mathbf{J_z}}\mathcal{H}e^{-i\omega_1 t\mathbf{J_z}} - \omega_1\mathbf{J_z} \tag{85}$$

1.4.2 Resonant frequency field

Suppose we observe the equilibrium system due to Zeeman interaction $\mathcal{H} = -\gamma B_0 J_z$ from a frame rotating with ω_1. According to equation (85)

$$\mathcal{H}' = e^{i\omega_1 t\mathbf{J_z}}\mathcal{H}e^{-i\omega_1 t\mathbf{J_z}} - \omega_1\mathbf{J_z} = -\gamma(B_0 + \frac{\omega_1}{\gamma})\mathbf{J_z} \tag{86}$$

The rotating frame introduces another term acting like an additional magnetic field. By choosing the angular velocity to equal the Zeeman effect's Larmor frequency $\omega_1 = -\gamma B_0$ we can make the net magnetic field vanish in the rotating frame. This is easy to imagine: The rotating frame just follows the system's Larmor precession, letting the system appear stationary.
Now it is easy to imagine what happens in the rotating frame, if we pulse the system by a transversal circularly polarized magnetic field $\vec{B}_1 = B_1(\hat{x}\cos\omega_2 t + \hat{y}\sin\omega_2 t)$ and choose the frequency of B_1 to approximately equal the system's Larmor frequency around the static field $\omega_2 \approx \omega_1 = -\gamma B_0$. In this case of "resonance" the static magnetic field in the rotating frame disappears and the circularly polarized fiel appears to be static in x-direction. The static Hamiltonian[3]

$$\mathcal{H} = -\gamma\vec{B}_{\text{tot}}\cdot\mathbf{J} = -\gamma\left(B_0\mathbf{J_z} + B_1(\cos\omega_2 t\ \mathbf{J_x} + \sin\omega_2 t\ \mathbf{J_y})\right) \tag{87}$$
$$= -\gamma\left(B_0\mathbf{J_z} + B_1 e^{-i\omega_2 t\mathbf{J_z}}\mathbf{J_x}e^{i\omega_2 t\mathbf{J_z}}\right) \tag{88}$$

[3]for a prove of $e^{-i\omega_2 t\mathbf{J_z}}\mathbf{J_x}e^{i\omega_2 t\mathbf{J_z}} = \cos\omega_2 t\ \mathbf{J_x} + \sin\omega_2 t\ \mathbf{J_y}$ see [8, chap 2.6 Exponential Operators, p. 27f]

transforms to the rotating Hamiltonian

$$\mathcal{H}' = e^{i\omega_1 t \mathbf{J_z}} \mathcal{H} e^{-i\omega_1 t \mathbf{J_z}} \tag{89}$$

$$= -\gamma \left((B_0 + \frac{\omega_1}{\gamma}) \mathbf{J_z} + B_1 \mathbf{J_x} \right) \quad \text{with} \quad \omega_2 = \omega_1 \tag{90}$$

$$= -\gamma B_1 \mathbf{J_x} \quad \text{with} \quad \omega_1 = -\gamma B_0 \tag{91}$$

In case of using a linearly polarized instead of a circularly polarized pulse, it may be argued that the rotating Hamiltonian still looks the same, since a linearly polarized field may be decomposed into two circularly polarized components, rotating in opposite directions. In the rotating frame , one of those components just behaves like the B_1 field above, the other one oscillates so rapidly at $2\omega_2$, that the spins experience only the fluctuating field's average value of zero[4]. From the classical point of view, in the rotating frame the spins now precess around $B_1\hat{x}'$ as they do around $B_0\hat{z}$ in the resting frame. From the quantum mechanical point of view, spins are now inclined to undergoe quantized transitions from states aligned around B_0 into states aligned around B_1.
If the pulse is applied for a duration Δt such that $\omega_1 \Delta t = \frac{\pi}{2}$, the spin system aligned along the z-axis will be reorientated along the y'-axis of the rotating frame. After the pulse, the system will evolve in time and relax into its equilibrium under the spin Hamiltonian's interactions.

1.4.3 FID detection

When the spins are precessing around B_1 in the rotating frame, and undergoing a more complex motion resulting from a superposition of precession around the static B_0 and the rotating B_1, what physical quantity are we going to measure? As has been noticed, the system's magnetic moment is proportional to its spin orientation $\vec{\mu} \propto \mathbf{S}$, hence we want to record the system's total magnetization. For this purpose a detection coil is installed orthogonal to the static magnetic field B_0, which records the change of the spin system's oscillating magnetization via Faraday induction.
For $\frac{\pi}{2}$-pulse EPR, right after the pulse the spins lie in the x-y-plane of the lab frame, rotating around the z-axis, and together with them the magnetization $\vec{M}$. Say, the detection coil is aligned along the x-axis, then it will record the change of magnetization in x-direction M_x, and the coil's signal will oscillate with the spin system's Larmor frequency

$$S(t) = S_0 \cos \omega_1 t = S_0 \frac{e^{-i\omega_1 t} + e^{i\omega_1 t}}{2} \tag{92}$$

The signal is modulated with a complex oscillating signal $e^{i\omega t}$

$$S'(t) = S(t) e^{i\omega t} = \frac{1}{2} S_0 \left(e^{-i(\omega_1 - \omega)t} + e^{i(\omega_1 + \omega)t} \right) \tag{93}$$

and the sum frequency term is filtered out to yield

$$S'(t) = \frac{1}{2} S_0 e^{-i(\omega_1 - \omega)t} \tag{94}$$

Neglecting any relaxation, the evolution of a density matrix titled into the x-y-plane by a $\frac{\pi}{2}$-pulse may be written as

$$\rho \propto \mathbf{J_y} \cos \omega_1 t + \mathbf{J_x} \sin \omega_1 t \tag{95}$$

and if we express the density matrix in a rotating frame of velocity ω

$$\rho \propto \mathbf{J_y} \cos(\omega_1 - \omega) t + \mathbf{J_x} \sin(\omega_1 - \omega) t \tag{96}$$

[4] this is argued in [7, chap. 4.2.2 The resonant frequency field, p.111f]

rank J	orientations N
47	770
53	974
59	1202
65	1454
77	2030
95	3074
101	3470
107	3890
113	4334
119	4802
125	5294
131	5810

Figure 1: the number of orientation for selected Lebedev grids of rank J

If we are able to evaluate $\mathbf{J}_+ = \mathbf{J_x} + i\mathbf{J_y}$ as a "complex observable" in the rotating frame, it would yield[5]

$$Tr(\mathbf{J}_+\rho') \propto i\ Tr({\mathbf{J_y}}^2)\cos(\omega_1-\omega)t + Tr({\mathbf{J_x}}^2)\sin(\omega_1-\omega)t \tag{97}$$

$$\propto i(\cos(\omega_1-\omega)t - i\sin(\omega_1-\omega)t) \quad \text{since} \quad Tr({\mathbf{J_y}}^2) = Tr({\mathbf{J_x}}^2) \tag{98}$$

$$N\gamma\ Tr(\mathbf{J}_+\rho') = iM_0e^{-i(\omega_1-\omega)t} \propto \frac{1}{2}S_0e^{-i(\omega_1-\omega)t} = S'(t) \tag{99}$$

Hence we see that the detection coil's modulated signal is just proportional to the magnetization we would observe in a reference frame rotating with the modulation frequency ω, and same applies for a relaxing manetization $M(t)$ instead of M_0 modulating the signal. The S' signal oscillates at frequency $\Delta\omega = \omega_1 - \omega$, and the conclusion to draw is that the rotating frame does not only constitute a handy tool to visualize the effect of a magnetic pulse, but proves equally useful in the detection process as well. Thus any simulation might well be performed without ever leaving the rotating frame, yielding the FID in form of the complex observable $\mathbf{J}_+ = \mathbf{J_x} + i\mathbf{J_y}$.

1.4.4 Real and imaginary spectrum

insert

1.4.5 Averaging over Lebedev grids

Since we consider powder samples, we have to average the spectra of many differently orientated spin ensembles. Assuming each spin ensemble to be randomly orientated, we need a method to discretize the unit sphere in a way such that the resulting discrete orientations represent the infinite amount of continuous possible orientations most accurately and then perform numerical integration about the unit sphere. *Gaussian spherical quadrature* is a powerful integration technique, and the most efficient sampling scheme for the sphere was developed by the Russian mathematician Lebedev. Lebedev grids provide a method of weighted discretization, depending on the resolution they are labeled with their rank J, and Spinach includes precomputed angles and weights for Lebedev grids up to rank 131. In table (1) one finds the number of orientations for Lebedev grids of certain rank.

[5] where the last line just reflects the relation between spin and magnetization in a sample of N spin systems, see [7, chap. 4.3.1 Free precession and Faraday detection, p. 125]

With the precomputed weights w_i and orientations θ_i, ϕ_i, the evaluation of the integral I of function f on the unit sphere reduces to a sum

$$I[f] = \sum_{i=1}^{N} w_i f(\theta_i, \phi_i) \tag{100}$$

1.5 Essence

The previous theoretical piece yields the following essential facts:

- the spin interaction properties of a system consisting of n spins including one electron and $n-1$ other nuclei can be described fully by one 3×3 g-tensor for the interaction linear in electron spin and a set of $n-1$ 3×3 A-tensors for the interactions between electron spin and each nucleus spin.
- the computation acts on the density matrix ρ in Liouville space
- the density matrix ρ may be propagated step-wise in time by applying the Liouvillian superoperator $\mathcal{L}$ repeatedly for short time steps in the manner of equation (20)
- a spin system of N states spans an N^2-dimensional Liouville space and an N^4-dimensional superoperator space
- the memory consumption of the simulation will be determined by the single isotropic part and the 25 anisotropic parts stemming from the $Q_{mm'}$ of the Liouvillian, which together scale with $26 \times N^4$.
- a spin simulation may be conducted entirely in the rotational frame
- the FID can be obtained by evaluating $\mathbf{J}_+$ on the propagated density matrix ρ
- the simulation has to be averaged over many orientations of a powder sample

A spin system consisting of an unpaired electron ($S = \frac{1}{2}$) and a nitrogen core ($S = 1$) will yield $N = 2 \cdot 3 = 6$ possible states, a density matrix of $N^2 = 36$ elements, and a Liouvillian of $26 \cdot N^4 = 26 \cdot 1296 = 33,696$ elements. A Matlab complex double representation takes 16 bytes, yielding an estimate of $16 \cdot 26 \cdot N^4\text{B} \approx 530\text{kB}$ memory usage at maximum. If we add 4 protons ($S = \frac{1}{2}$) to the system, we have $N = 6 \cdot 2^4 = 96$ states, 9216 density matrix elements, $\sim \cdot 2.2 \cdot 10^9$ Liouvillian elements and a maximum memory usage of 33GB. In fact, the Matlab simulations take by far less memory, since all of the involved matrices are sparse. We will have a thorough memory benchmarking lateron.

2 Parallelization of Spinach

The Matlab library *Spinach*[6] supplies efficient methods for large-scale spin dynamics simulations. It consists of the *kernel* with implementations of general spin dynamics simulation techniques and the *user-land* with a collection of different experiements to perform. Basically, the user prepares the description of a spin system, which is then translated by the kernel into basis set, Liouvillian superoperator, etc. The user-land decides how to deal with those objects, whether to apply a pre-established experiment, or whether to perform the kernel's simulation procedures manually. Though Spinach is able to simulate numerous kinds of experiments, in this work we are going to restrict ourselves to the standard $\frac{\pi}{2}$-pulsed EPR experiments, for which the user-land readily provides the method `pulse_acquire`.

[6] written by Theoretical Spin Dynamics Group, University of Southampton, www.spindynamics.org, for an architectural overview see `docs/developer_notes/architecture_overview`

2.1 Spinach modification

The implementation of a Spinach $\frac{\pi}{2}$-pulsed EPR simulation based on the theoretical concepts derived above is introduced in appendix (4.3) and may be referred to for better understanding of the following. The key modifications done to parallelize the code may be found in appendix (4.4).

2.1.1 Implementation

The simplemost form of parallelization applicable to Spinach will be to subdivide the amount of orientations into smaller packages and distribute them upon parallel tasks, depending on the number of nodes n and the number of cores per node p available. If we assume the restriction that MATLAB may only run parallel locally on one node, we may distribute a number N of orientations as follows:

1. Subdivide the orientations into n packages of N/n orientations each. If n is not divisor of N, then every package gets $\lfloor N/n \rfloor$ orientations and the remaining orientations N mod n are distributed equally over the first N mod n packages.
2. Evoke one MATLAB process on each node, hand them their orientation package and let them open a MATLAB pool with p workers.
3. (a) Execute a `parfor`-loop for each orientation in the package of $\lfloor N/n \rfloor$ or $\lfloor N/n \rfloor + 1$ orientations, in the following referred to as the "parallel" case.
   (b) Just as above, subdivide the orientation package again into p smaller packages for each core and distribute remaining orientations equally, then run a `parfor`-loop p times and let every core evaluate their package of orientations at once, in the following referred to as the "mixed" case, since it combines serial and parallel execution.

The two methods (a) and (b) of paralleization are both implemented. For case (a) we expect more communication between client and workers, but lower peak memory usage. For case (b) we expect to be able to reduce communication between client and workers and thereby safe time. Yet, since all orientations are propagated at once, we expect much higher peak memory usage. We are going to investigate this hypothesis lateron.
We will split the tasks performed by `pulse_acquire(...)` into three different functions, which are scheduled as distinct jobs and run subsequently, each one processing the they input data prepared by the previous:

- `function jlh_master_pulse_acquire(...)` — runs first, prepares the Liouvillian, and evokes several instances of...
- `function fid = jlh_outsourced_pulse_acquire(...)` — which run on each single node and do the actual propagation, producing every orientation package's FID.
- `function jlh_sum_results(..)` — finally runs after all `jlh_outsourced_pulse_acquire(...)` have finished, collects their results, finalizes them and outputs the total FID.

Furthermore we will make use of the helper functions

- `function job_scheduler = jlh_which_job_scheduler()` — determines the job scheduler equipped on the current platform and returns `'TORQUE'` or `'SLURM'`.
- `function [job_identifier status stdout] = jlh_submit_job(... )` — submits a job executing command `cmd` via the platform's job scheduler.
- `function [ pmem, walltime, queue ] = jlh_estimate_resources(...)` — a stub to be adapted to a specific cluster configuration. Shall be expanded to estimate the maximum CPU time and memory usage of a Spinach job depending on its parameters.

2.2 Benchmarking

There are three major technical factors, which limit the scalabitlity of our parallelization:

1. execution time – a simulation exceeding several days of runtime is not considered economical, in addition clusters usually limit the maximum execution time.
2. memory – depending on the spin system's dimensions the Liouvillian grows exponentialy. When several orientations are propagated simultaneously, many copies of the Liouvillian may reside in a node's memory simultaneously, easily exceeding the cluster's hardware limitations.
3. communication between threads – our parallelization relies on transferring the prepared Liouvillian in form of a local file, which might easily reach gigabyte dimensions. What is more, MATLAB internally limits the amount of data transferred to every `parfor`-iteration.

Reducing the amount of simultaneously used memory by serialization means increasing the runtime and vice versa. More cores involved mean more memory occupied at the same time. In the following we want to examine how far we may exhaust those limitations and which of them will prove most restrictive. The following benchmarking has been performed on *Soroban*[7] and evaluates only `jlh_outsourced_pulse_acquire(...)` instances, thus not accounting for the resources required to build the Liouvillian.

2.2.1 Single orientation

To start with, we examine the scaling of Spinach for evaluating single orientations of 2- to 20-spin systems without involving any parallelization. We use the complete basis ESR-2 and the reduced basis ESR-1 and measure evaluation time, peak virtual memory consumption[8] and file size of the common input file storing the Liouvillian in dependence on the spin systems complexity. The semi-logarithmic graphs and their exponential fits in figure (2a), (2b) and (2c) clearly reflect exponential scaling in all three cases, except for deviations in very small systems. In those regimes technical side-effects may influence the scaling.
For the complete basis EPR-2 memory usage clearly is the limiting factor. While one orientation may be computed within minutes, the memory costs blow up into gigabytes and exceed Soroban's limit for a 12-spin system. The incomplete basis EPR-1 reaches such memory dimensions only for spin systems larger than 20. Consequently, we were able to benchmark spin systems up to size 11 for ESR-2 and up to size 20 for ESR-1.
The Liouvillian's scaling determines the scaling of memory usage for a single orientation linearly, as may be found in graph (2d). Both ESR-1 and ESR-2 case show the Liouvillian in file form expanding into a simulation of 20- to 22-fold size on a node's virtual memory.

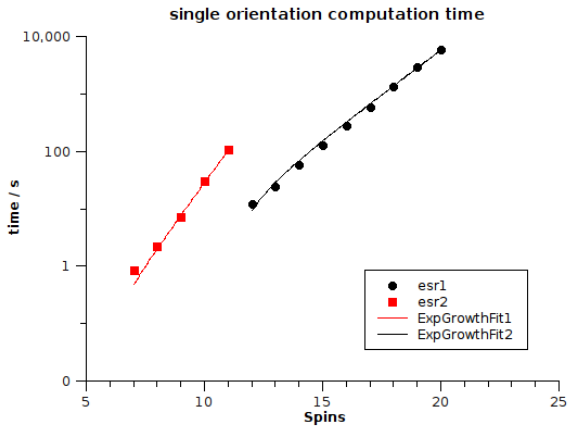
2.2.2 Single core computaion

If only one core is available, the parfor-loop reduces to a serial for-loop in "parallel" mode, while the "serial" mode will propagate all orientations at one shot. Clearly, MATLAB's vector calculus optimization will somehow serialize the matrix calculations internally. In this section we will find out the resource usage of explicit serial calculation and of MATLAB matrix operations.

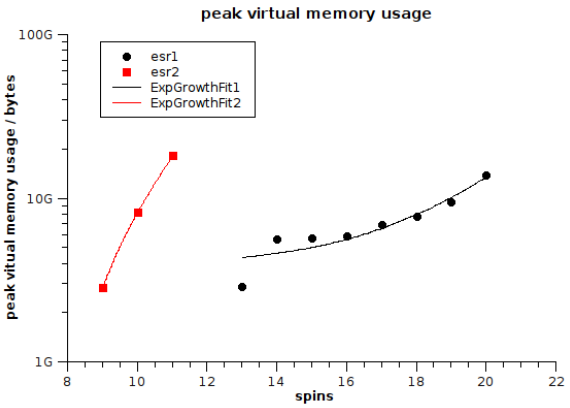
2.2.3 Small scale spin systems

The previous section leads to the conclusion that we are dealing with two different scaling regimes: For spin systems with less than seven spins, the "mixed" mode proves faster, whereas it does not bear any advantages, neither in runtime, nor in memory usage, for larger systems.

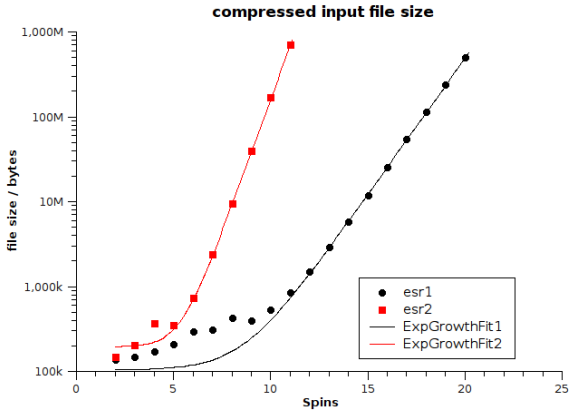
[7] see appendix (4.1.2) for detailed technical information
[8] virtual memory combines active RAM and inactive memory such as swap files



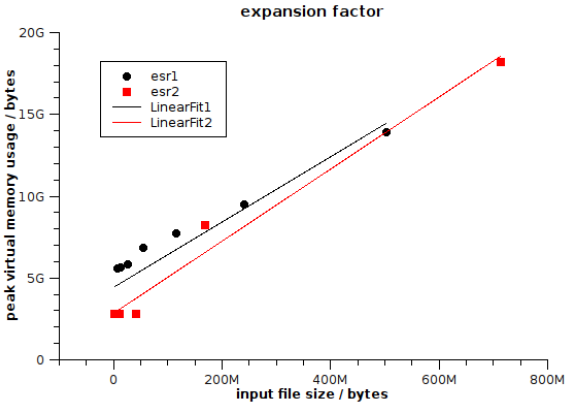
(a) execution time in dependence on spin system complexity



(b) peak virtual memory usage in dependence on spin system complexity

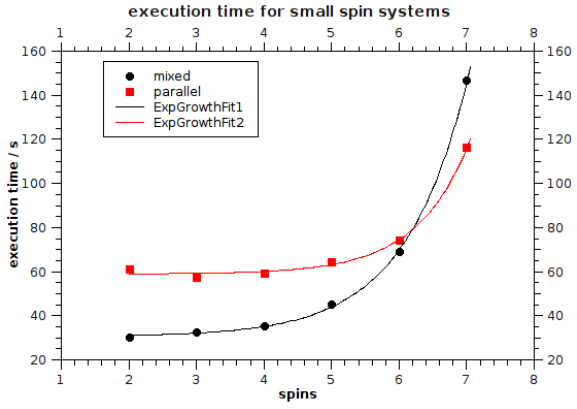


(c) Liouvillian file size in dependence on spin system complexity

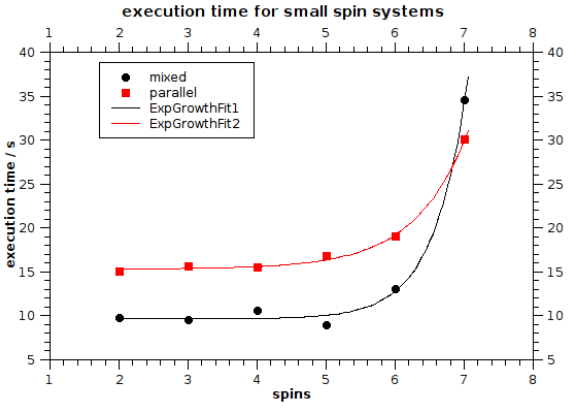


(d) virtual memory usage in dependence on Liouvillian file size

Figure 2: Spinach execution statistics for one single orientation

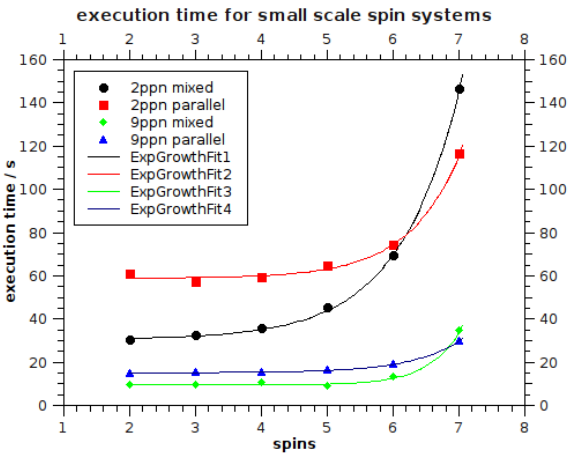


(a) two-core parallelization

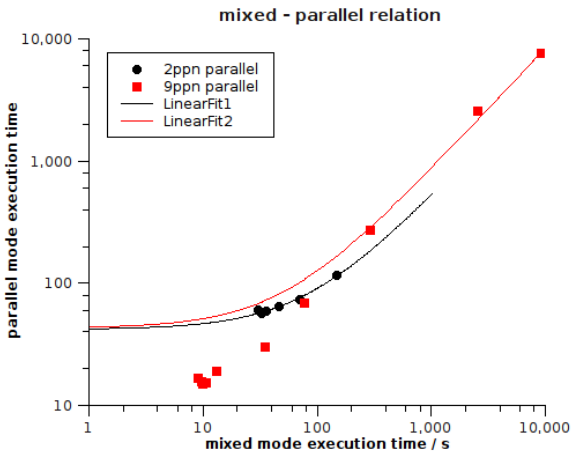


(b) nine-core parallelization

Figure 3: execution time in dependence on spin system complexity for small scale spin systems



(a) two-core and nine-core parallelization runtime



(b) runtime relation between mixed mode and parallel mode

Figure 4

2.2.4 Large scale spin systems

From now on we examine the scaling behaviour for spin systems with more than six or seven spins in parallel mode. Technical side-effects diminish in comparison to the exponential scaling and by benchmarking simulations for several spin-core configurations we try to build a scaling model for large spin systems on Soroban and extrapolate it to even more complex spin systems to be able to predict whether and on what cluster configuration a certain large scale EPR simulation may run.

2.2.5 Scaling model

Since the dimensions of the Liouvillian increase exponentially with the scale of the spin system x, we expect likewise exponential growth in runtime r and memory requirements m of form

$$r_y(x), m_y(x) \propto a + b \cdot e^{\alpha x} \tag{101}$$

Furthermore, we assume that CPU time is distributed equally among the cores y, such that

$$r_x(y) \propto c + \frac{d}{y} \tag{102}$$

whereas more cores y will occupy more memory m simultaneously, hence

$$m_x(y) \propto c + d \cdot y \tag{103}$$

Based on this background we will try to fit our benchmark results to the model

$$r(x,y) = (a_r + b_r \cdot e^{\alpha_r x}) \left(c_r + \frac{d_r}{y}\right) \tag{104}$$

$$m(x,y) = (a_m + b_m \cdot e^{\alpha_m x}) (c_m + d_m \cdot y) \tag{105}$$

3 Conclusion and outlook

4 Appendix

4.1 Clusters

Most supercomputers nowadays are so called *clusters*: Many *nodes* with several *cores (CPUs)* each are connected via fast network. One single node's *memory (RAM)* may be shared by all its cores, but every CPU usually has its own range of RAM, to which access is fastest. Access on another node's RAM via network is slowest. The advantage of computer clusters is obvious: hardware does not differ significantly from standard desktop computers, keeping purchase and maintainance costs reasonably low. The architecture of a cluster requires efficient resource management. For this purpose, several *job scheduler* platforms exist. Everyone, who wishes to run a parrallel job on the cluster, has to *submit* the job via the job scheduler. The job scheduler then asseses the job's potential resource usage upon the user's estimation and *enqueues* the job with a certain priority for execution. Hence the user shall estimate well: Too generous estimations of memory usage and runtime will cause the job to be enqueued for days, too tight estimations though will cause the job to be terminated by the scheduler when it exceeds its claimed resources by far.
Both clusters available for testing run on Linux, but they use different job schedulers. Here follows a short introduction of their features.

4.1.1 Sheldon

Sheldon is a linux computer cluster of the Freie Universität Berlin physics department and posseses nodes of different specifications, reaching from 68 eight core nodes with 2 to 4 GB RAM per core (n042-n109) over 32 twelve core nodes with 8GB RAM per core (n010-n041) up to two 32 core nodes with 2GB RAM per core (n110-n111)[9]. A process running exclusively on one node can thus make use of 96GB RAM at maximum on the twelve core nodes. *TORQUE Resource Manager*[10] serves as Sheldon's job scheduler, and here is a summary of the most important commands:

- `qsub` *jobfile-name* – submits a job for execution and returns a five digit job identifier
- `showstart` *job-id* – yields the expected start time of a certain job's execution
- `qdel` *job-id* – removes a certain job from the queue or aborts its execution
- `qstat` *job-id* – yields the status of all current jobs or a certain job already submitted
- `qf` – displays the resource usage on all nodes of the whole cluster

A jobfile to evoke a matlab function will look like this:

```
#!/bin/bash
#PBS -N testjob
#PBS -q batch
#PBS -l nodes=1:ppn=2
#PBS -l pmem=2gb
#PBS -l walltime=01:00:00
#PBS -m bea -M user@zedat.fu-berlin.de
#PBS -e testjob_torque_error
#PBS -o testjob_torque_output
#PBS -W depend=afterok:job-id1:job-id2:...

cd $PBS_O_WORKDIR
matlab -nodesktop -nosplash -r "generic_matlab_function(...); quit;" &> testjob_matlab.log
```

[9] see https://wiki.physik.fu-berlin.de/it/doku.php?id=services:cluster:start for detailed specifications
[10] maintained by *Adaptive Computing*, http://www.adaptivecomputing.com/products/open-source/torque/

All `#PBS`-lines are options to be read by TORQUE, they determine from top to bottom the job's name `testjob`, the queue, the number of requested nodes and processors per node (ppn), the requested RAM per core, the maximum execution time, an email address to send status messages to, a filename for the error logfile, a filename for the standard output logfile, and a dependency list for jobs to be finished successfully before executing this job. Sheldon possesses two queues `batch` and `highmem`, which enqueue for execution on the eight core nodes or on the twelve core nodes respectively. `cd $PBS_O_WORKDIR` changes to the directory from which `qsub` was called, usually somewhere in the user's home directory. After `testjob` is finished, the TORQUE error and output can be found in this directory as `testjob_torque_error.e` and `testjob_torque_output.o`, while Matlab's error and output will be located in `testjob_matlab.log`.
For the time of writing Sheldon had a trial version of the MATLAB Distributed Computing Server available. Thus it was possible to conduct many-node parallel simulations there.

4.1.2 Soroban

Soroban is a linux computer cluster of the Freie Universität Berlin ZEDAT consisting of 112 nodes with twelve cores and 2GB RAM per core each[11]. A process running exclusively on one node can thus make use of 24GB RAM at maximum. SLURM[12] is Soroban's job scheduler, and here is a summary of its most important commands:

- `sbatch` *jobfile-name* – submits a job for execution and returns a five digit job identifier
- `scancel` *job-id* – removes a certain job from the queue or aborts its execution
- `squeue -j` *job-id* – yields the status of all current jobs or a certain job already submitted
- `sinfo` or `sview`– display info on the resource usage on all nodes of the whole cluster, latter offers a GUI.

A jobfile to evoke a matlab function will look like this:

```
#!/bin/bash
#SBATCH --job-name=testjob
#SBATCH --partition=main
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=2
#SBATCH --mem-per-cpu=2GB
#SBATCH --mail-type=ALL
#SBATCH --mail-user=user@zedat.fu-berlin.de
#SBATCH --error=testjob_slurm_error.e
#SBATCH --output=testjob_slurm_output.o
#SBATCH --dependency=afterok:jod-id-1:job-id-2:...

cd $SUBMITDIR
module load matlab/R2011b
matlab -nodesktop -nosplash -r "generic_matlab_function(...); quit;" &> testjob_matlab.log
```

The meaning of all paramters goes in analogy with the TORQUE parameters described above. For Soroban it is not necessary to specify a certain queue, since the nodes do not differ much in hardware, but there still exist the queues (or "partitions") `main`, `test` and `gpu`.
On Soroban there are five MATLAB client licenses and two Parallel Computing Toolbox licenses available at the time of writing. Further restrictions apply: one MATLAB process can only open a MATLAB pool within one node with a maximum of INSERT workers. For our parallelization this means at most two multiple-core MATLAB processes with at most eight INSERT cores each, resulting a maximum of 16 parallel threads.

[11] see http://www.zedat.fu-berlin.de/Compute/Soroban for detailed specifications
[12] maintained by Lawrence Livermore National Laboratory 7000 East Avenue, Livermore, CA 94550, USA, https://computing.llnl.gov/linux/slurm/slurm.html

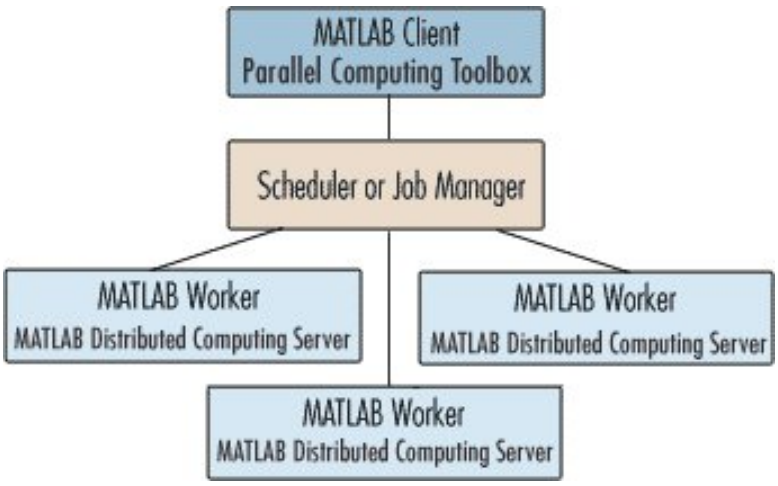


Figure 5: The paralleization model of MATLAB

source: http://www.mathworks.com/support/product/DM/installation/images/interaction_new1.gif

4.2 MATLAB on Clusters

Code parallelization is highly platform dependeble, as argued in the previous section. What is more, the MATLAB environment imposes further constraints upon parallelization. Here we outline MATLAB concepts and their limitations.

4.2.1 MATLAB licences

MATLAB has a rather dedicated licence model, and for a parallelized computation several licenses are necessary:

- A *MATLAB client*, the standard MATLAB distribution to be checked out for the user's work station when starting MATLAB
- A *Parallel Computing Toolbox*, likewise to be checked out for the client to enable parallel code features
- A *Distributed Computing Server* on the cluster to open *MATLAB workers* when requested by a client. The number of workers per node and per cluster is limited depending on the actual server licence.

The very idea of parallelization in Matlab visualized in figure (5) then works as follows: A user wants to run parallel code in their MATLAB client. With the Parallel Computing Toolbox' command `matlabpool open` *poolsize* they may send a request to the cluster's Distributed Computing Server to allocate a number *poolsize* of workers – remote MATLAB processes without GUI. Granted that enough worker licences are available, the connection between client and cluster will be established and the user is then free to run their code. Afterwards he is to release the worker licences by calling `matlabpool close`.
This approach does not prove very handy in the case of our Spinach simulations for several reasons. Firstly, if the connection between client and cluster is lost, it is problematic to reconnect and receive any results from the "workers without leader". Secondly, in general work stations do not necessarily have the Parallel Computing Toolbox equipped. Thirdly, a smooth communication between Distributed Computing Server and job scheduler is required, which is rather fancy to configure. Hence we will "work around" the client-cluster problem by evoking a "silent" MATLAB client on one cluster node with minimal resource usage, which serves as a *master process* by acting as the "user within the cluster", calling more MATLAB processes, opening the MATLAB pool and delegating tasks to the allocated workers automatically.

4.2.2 Matlab - Linux interaction

MATLAB offers one important command to evoke shell commands from within a MATLAB client:

```
[status,stdout] = unix('any_command');
```

With its help it will be possible for us to create a function `jlh_submit_job(...)` submitting jobs on the cluster dynamically during MATLAB execution by calling `qsub` or `sbatch`. Furthermore, we can extract the returned job identifier, track the job throughout its execution and create subsequent dependent jobs, all from within MATLAB. `jlh_submit_job(...)` can perform cluster-specific job submission by creating job-files such as mentioned above. Thereby we keep cluster-dependency away from Spinach.
One should notice that each MATLAB job executed directly via a jobfile checks out another MATLAB client and Parallel Computing Toolbox licence, while jobs indirectly scheduled via the `matlabpool` command stay inside one Parallel Computing Toolbox, but check out numerous worker licences. The aim will be to make maximum use of the available combination of client and worker licences.

4.2.3 Parallel code elements

The *Parallel Computing Toolbox* offers several parallelization concepts, of which only the `parfor`-loop will be important for us:

```
parfor i=int_a:int_b
  ...code...
end
```

The parfor-loop just works like a normal for-loop, with the difference that single loop iterations are distributed to all workers of the current MATLAB pool. The loop iterations are then only executed on the workers, but not on the calling MATLAB client process. Consequently, the parfor-loop may not be applied to replace for-loops where the computation results depend on the ordered sequence of execution. Since we are only averaging over many orientations of an EPR powder sample, the parfor-loop may well be used in our case.
Variables used inside a parfor-loop have to be initialized explicitely in the preceding code segment, in order to inform MATLAB about which workspace information to transfer to the workers. Furthermore, arrays indexed inside the parfor-body by the iteration variable `i` will be identified as "sliced" by MATLAB, and only the relevant parts of the array will be transferred to the worker. Those facts will turn out relevant lateron.

4.3 A Spinach EPR Simulation

Using the theoretical basis introduced above, the procedure of an EPR simulation comes down to the following key steps:

1. Spinach reads the g- and A-tensors from an input file and constructs the isotropic and anisotropic part of the Liouvillian in the rotational basis ...
2. ... propagates the density matrix in one step through the $\frac{\pi}{2}$-pulse by applying a rotation operator ...
3. ... then propagates the density matrix step by step through the relaxation process by applying the Liouville equation (52) repeatedly for short time steps, ...
4. ... and determines the transversal magnetization depending on $\mathbf{J}_+$, a superposition of spin in x- and y-direction, for each step.

In the following the preparation of input data and computation flow are summarized for a simple system consisting of an electron spin $\frac{1}{2}$, a nitrogen spin 1 and a proton spin $\frac{1}{2}$.

4.3.1 Typical input file

This Matlab file prepares a typical spin system and conducts a $\frac{\pi}{2}$-pulsed EPR experiment on it. The data structure `sys` contains information about the spin system and the experimental setup, `inter` represents the linear and bilinear interactions of the spins, `bas` specifies the basis set to be used and `parameters` specifies the enquired simulation results.

```
function jlh_3spins()

% Set the simulation parameters
sys.magnet=3.356;
sys.regime='powder';
bas.mode='ESR-1';
sys.tols.grid_rank=101;
```

Specifies $B_0 = 3.356T$ and tells Spinach to average the spectrum over uniformly distributed orientations in a "powder". The mode "ESR-1" generates a complete state space for all electrons, but reduces the state space for all nuclei in a certain way to be efficient enough and still yield reasonable EPR results. The mode "ESR-2" will create a full state space for all electrons and anisptropically coupled nuclei, that is all nuclei in the case of our system. This would result in a 12-dimensional state space, such that the density matrix stretched into a column vector will count 144 elements, and the superoperators will be represented by 144×144 matrices. The grid rank 101 chooses a certain Lebedev grid of 3470 orientations to average about.

```
% Interactions
sys.isotopes={'E','14N', '1H'};
inter.zeeman.matrix=cell(3,1);
inter.zeeman.matrix{1,1}=[ 2.0056023 -0.0013775 -0.0004019; -0.0008185 2.0038816 0.0002087;
    -0.0002747 0.0001729 2.0062982];
inter.coupling.matrix=cell(3,3);
inter.coupling.matrix{1,2}=1e6*[29.4479  3.9997 -0.4979;   3.9997  76.1445 -14.8367;  -0.4979
    -14.8367  38.4285];  %92.815191
inter.coupling.matrix{1,3}=1e6*[-1.2922  2.0819 -1.9943;  2.0819  -1.407 -1.7023; -1.9943
    -1.7023 -1.5011];  %5.3217912
```

Advises Spinach to prepare a spin system of an unpaired electron, a nitrogen nucleus and a proton spin[13]. The Zeeman matrix represents the 3×3 g-Tensor, while the coupling matrices represent the 3×3 A-tensors accounting for the hyperfine interactions between electron spin and all other spins.

```
% Set the sequence parameters
parameters.offset=0;
parameters.sweep=1e9;
parameters.npoints=512;
parameters.zerofill=1024;
parameters.spins='E';
parameters.axis_units='Gauss';
parameters.derivative=0;
```

Sets parameters for the experiment to simulate: `npoints` determines the number of time steps in the simulation, `sweep` chooses the spectral window's width, and thus the duration of one time step, `zerofill` sets the FID zero-filling and `spins` selects the spins to be pulsed and detected – the electron spin in our case. `derivative` can be set to evaluate for the derivative of the FID instead.

```
% Run Spinach
spin_system=create(sys,inter);
spin_system=basis(spin_system,bas);
fid=pulse_acquire(spin_system,parameters);
```

[13] for a full list of available standard isotopes have a look at `kernel/spin.m`

The first kernel functions to be called are `create(sys,inter)` and `basis(spin_system,bas)`. Former returns the data structure `spin_system`, which describes can be handed to the kernel lateron to reference to our spin system, e.g. to fetch the Liouvillian in question. A simplified sketch of `spin_system`'s internal structure may be found in figure (insert). The values assigned to its different attributes by `create` and verb*basis* for our specific sample spin system are added in bold font. Attributes directly determined by the parameters from this input file are marked red, attributes derived indirectly from those parameters are marked green.
`pulse_acquire(spin_system, parameters)` finally conducts the π-pulsed EPR and returns the time-resolved FID.

```
27 % Apodization
28 fid=apodization(fid,'crisp-1d');
29
30 % Perform Fourier transform
31 spectrum=fftshift(fft(fid,parameters.zerofill));
32
33 % ...
34
35 end
```

The "crisp" apodization modulates the FID by a declining cosine window function in the interval $[0, \frac{\pi}{2}]$

$$\mathrm{FID}'(t) = \mathrm{FID}(t) \cdot \cos^8\left(\frac{\pi}{2} \cdot \frac{t}{L_{\mathrm{FID}}}\right) \tag{106}$$

Line 31 finally performs Fast Fourier Transform to generate the frequency domain spectrum.

4.3.2 pulse_acquire

```
1  function fid=pulse_acquire(spin_system,parameters,L,rho)
2
3  %...
4
5  % Compute the digitization parameters.
6  timestep=1/parameters.sweep;
7
8  % Generate the basic operators
9  Lp=operator(spin_system,'L+',parameters.spins);
10 Ly=(Lp-Lp')/2i;
```

The function `operator` prepares the raising superoperator J_+ and then constructs

$$J_y = \frac{1}{2i}(J_+ - J_-) = \frac{1}{2i}(J_+ - J_+^\dagger) \tag{107}$$

The apostrophe in the Matlab code marks the complex conjugate transposition $J_+^\dagger$ of J_+. The relation above can be easily found by realizing that the raising and the lowering operator form a Hermitian conjugate pair $J_- = J_+^\dagger$

```
11 % Set the secularity assumptions
12 spin_system=secularity(spin_system,'nmr');
13
14 % Start from thermal equilibrium
15 rho=equilibrium(spin_system);
```

The `secularity` function decides about the importance of interactions. In high field EPR and NMR, only the electrons' states are accounted for fully, whereas the nuclei's spins are only evaluated in z-direction. Density matrix `rho` is initialized with the termal equilibrium state of the spin system.

```
16 [Iso,Q]=h_superop(spin_system);
```

The isotropic part of the Liouvillian superoperator is stored in the 144×144 matrix `Iso`, while `Q` is the set of 25 $Q_{mm'}$ rotational basis operators stored in a 5×5 matrix of 144×144 matrices representing the Liouvillian's anisotropic part. The dimensions of those two matrices determine Spinach's memory consumption.

```
% Get the spherical averaging grid
grid=load([spin_system.sys.root_dir spin_system.sys.slash 'exp' ...
           spin_system.sys.slash 'grids' ...
           spin_system.sys.slash 'lebedev_rank_' num2str(spin_system.tols.grid_rank) '.dat'],
               'ASCII');
grid_size=size(grid,1); phi=pi*grid(:,1)/180; theta=pi*grid(:,2)/180; weight=grid(:,3);
```

The Lebedev grid is read from a file holding all precomputed orientations on the unit sphere. The number of points for a Lebedev grid of certain rank can be found in table (1).

```
% Get the orientation array
L_aniso=orientation(Q,[phi theta zeros(size(theta))]);
L=blkdiag(L_aniso{:})+kron(speye(grid_size),Iso);
L=clean_up(spin_system,L,spin_system.tols.liouv_zero);
```

`orientation` rotates the anisotropic part of the Liouvillian in the rotational basis by the specified Euler angles and creates a cell array of operators, one for each orientation. The Liouvillian block matrix `L` has a diagonal element for every Lebedev orientation n:

$$L = L_{\text{aniso}} + I \otimes L_{\text{iso}} = \begin{pmatrix} L_{\text{aniso},1} & & & \\ & L_{\text{aniso},2} & & \\ & & \ldots & \\ & & & L_{\text{aniso},n} \end{pmatrix} + \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & \ldots & \\ & & & 1 \end{pmatrix} \otimes L_{\text{iso}} \qquad (108)$$

```
% Get the initial and the detection states
rho=kron(ones(grid_size,1),rho);
coil=kron(weight,state(spin_system,'L+',parameters.spins));
```

The density matrix is duplicated n times in a column vector to propagate one for each orientation, and similarly the observable to be detected is duplicated and the Lebedev weights are applied in a column vector `coil`. In the experimental setup the detection coil measures independently the magnetization in x- and y-direction, which correspond to the horizontal spin state $J_+ = J_x + iJ_y$, `'L+'` in Spinach notation.

```
% Apply the pulse
rho=step(spin_system,kron(speye(grid_size),Ly),rho,pi/2);

fid=evolution(spin_system,L,coil,rho,timestep,parameters.npoints-1,'observable');

end
```

Generally, `step(spin_system,L,rho,dt)` propagates the density matrix `rho` under the influence of a certain Liouvillian `L` by a time step `dt`. Because `step` makes uses of the evolution superoperator (19) internally, it can conduct a 90° rotation around the y-axis by replacing the time step by an angle $\frac{\pi}{2}$ and the Liouvillian by the spin superoperator J_y, letting it construct the rotation operator

$$R_y(\tfrac{\pi}{2}) = e^{-i\frac{\pi}{2}J_y} \qquad (109)$$

in analogy to equation (74). This just rotates all spins into the x-y-plane, just as a $\frac{\pi}{2}$-pulse will do. Sequently, `evolution` can be regarded as a sequence of `npoints` `step`-functions propagating the density matrix through the whole time interval by steps of duration `timestep` by applying the Liouvillian `L`. In addition, the observable `coil` is evaluated for every single step, recording the time resolved FID with magnetization in x-direction as its real part and magnetization in y-direction as its imaginary part.

4.4 Spinach modification code

4.4.1 Master process

The enhanced call

```
function jlh_master_pulse_acquire(spin_system,parameters,L,rho)
```

compared with the original call

```
function fid=pulse_acquire(spin_system,parameters,L,rho)
```

does not return any results. Hence it should be the last call in the input file[14]. In addition to the standard parameters it expects `parameters.nodes` and `parameters.ppn` to decide how many nodes and cores to distribute the orientations onto. `jlh_master_pulse_acquire` constructs the rotational basis set and the Liouvillian in the rotational basis by standard Spinach means, writes a file `input_file_name`

```
save(input_file_name, 'spin_system','Iso', 'Q', 'Ly', 'rho', 'phi', 'theta', 'weight',...);
```

where the stored information includes

- `Iso` and `Q` — the isotropic and anisotropic Liouvillian,
- `Ly` — the rotation operator to tilt the spins around the y-axis,
- `rho` — the density matrix in equilibrium,
- `phi`, `theta` and `weight` — the orientations and weights of the chosen Lebedev grid,
- ... and further technical parameters,

then uses `jlh_submit_job(...)` repeatetly to create n instances of `jlh_outsourced_pulse_acquire(...` which all read their data from the same input file `input_file_name`, but write there output to instance-specific output files. Since different orientations apparently take differing time intervals to be treated, the orientation packages are assembled such that every package contains only every n'th orientation, starting with offsets from $1, 2, \ldots$ to n. Thereby a better spatial distribution of the orientations among the packages is guaranteed.
Eventually, one instance of `jlh_sum_results(...)` is queued to run after the successful execution of all `jlh_outsourced_pulse_acquire(...)` instances, collecting and adding up all parial FIDs.

4.4.2 Child processes

The essence of the simulation is packed in

```
function fid=jlh_outsourced_pulse_acquire(orientation_start, orientation_step, orientation_end
    , input_file_name, output_file_name)
    orientations = orientation_start:orientation_step:orientation_end;
    orientation_count = length(orientations);
```

which is able to calculate the accumulated FID over an arbitrary range of orientations specified by its parameters `orientation_start`, `orientation_step` and `orientation_end`.
When loading the Liouvillian and other data from the .mat file prepared by `jlh_master_pulse_acquire` we have to initialize the variables explicitly to make them available inside a parfor-body

```
tmp = load(input_file_name);
spin_system = tmp.spin_system;
Iso = tmp.Iso;
Q = tmp.Q;
Ly = tmp.Ly;
rho = tmp.rho;
```

[14] discussed in appendix (4.3.1)

```
phi = tmp.phi;
theta = tmp.theta;
weight = tmp.weight;
```

The function offers “parallel”, “mixed” and “serial” mode. In parallel mode, the orientations are treated according to case (a) of above’s enumeration:

```
% Get the detection state
coil = state(spin_system,'L+',parameters.spins);

parfor i=1:orientation_count
    n = orientations(i);

    % Get the current orientation
    L=Iso+orientation(Q,[phi(n) theta(n) 0]);

    % Apply the pulse
    rho_pulsed=step(spin_system,Ly,rho,pi/2);

    % Run the simulation
    fid=fid+weight(n)*evolution(spin_system,L,coil,rho_pulsed,1/parameters.sweep,parameters.
        npoints-1,'observable');
end
```

For every orientation in the package the parfor-body is executed ones, causing MATLAB to distribute the parfor-iterations among the available cores and applying the evolution operator once to every single orientation.
In mixed mode, the orientations are treated according to case (b):

```
phi_set = cell(1,parameters.ppn);
theta_set = cell(1,parameters.ppn);
weight_set = cell(1,parameters.ppn);

for i=1:parameters.ppn
    set = orientations(i:parameters.ppn:orientation_count);
    phi_set{i} = phi(set);
    theta_set{i} = theta(set);
    weight_set{i} = weight(set);
end

parfor i=1:parameters.ppn
    L_aniso=orientation(Q,[phi_set{i} theta_set{i} zeros(size(theta_set{i}))]);
    L=blkdiag(L_aniso{:})+kron(speye(length(theta_set{i})),Iso);
    L=clean_up(spin_system,L,spin_system.tols.liouv_zero);

    % Get the initial and the detection states
    current_rho=kron(ones(length(theta_set{i}),1),rho);
    current_coil=kron(weight_set{i},state(spin_system,'L+',parameters.spins));

    % Apply the pulse
    current_rho=step(spin_system,kron(speye(length(theta_set{i})),Ly),current_rho,pi/2);

    fid=fid+evolution(spin_system,L,current_coil,current_rho,1/parameters.sweep,parameters.
        npoints-1,'observable');
end
```

First of all, again smaller sets of orientations are “sliced” into the cell arrays `phi_set`, `theta_set` and `weight_set`. Since the packages can differ in length, we use cell arrays to allow for differing dimensions in each column. Then, very much like the standard serial computation[15], each orientation package is propagated at once in one parfor-iteration. Since we have exactly p iterations, every available core will evaluate one parfor-body and hence propagate one subset of orientations.
The serial mode treats all orientations at once locally.
Eventually the FID is written to an instance-specific output file.

[15] discussed in appendix (4.3.2)

4.4.3 Finalization process

Reads every single output file listed in `output_file_name`, adds up the partial FIDs and performs all finalization tasks like apodization and Fourier transform originally done in the end of the Spinach input file[16].

```
function jlh_sum_results(input_file_name)
    load(input_file_name,'spin_system','parameters','output_file_name','spectrum_file_name', '
        debug_file_name');

    %preallocating fid array
    fid=zeros(parameters.npoints,1);

    for i=1:parameters.nodes
        %checking, whether each process wrote ouput file correctly
        if exist(output_file_name{i}, 'file')
            tmp = load( output_file_name{i}, 'fid');
            fid = fid + tmp.fid;
        else
           %reporting errror otherwise
           report(spin_system, [sprintf('jlh_sum_results: Error: jlh_outsourced_pulse_acquire
               instance %d did not write output file "', i) output_file_name{i} '"!']);
        end
    end

    % Apodization
    fid=apodization(fid,'crisp-1d');

    % Perform Fourier transform
    spectrum=fftshift(fft(fid,parameters.zerofill));

    %Compute the derivative if necessary
    if isfield(parameters,'derivative')
        spectrum=fft(ifft(spectrum).*fftdiff(parameters.derivative,length(spectrum),1)');
    end

    ax=axis_1d(spin_system,parameters);
    data = cat(2, transpose(ax), real(spectrum));
    save(spectrum_file_name,'data', '-ASCII');
end
```

4.5 Figures

[16] discussed in appendix (4.3.1)

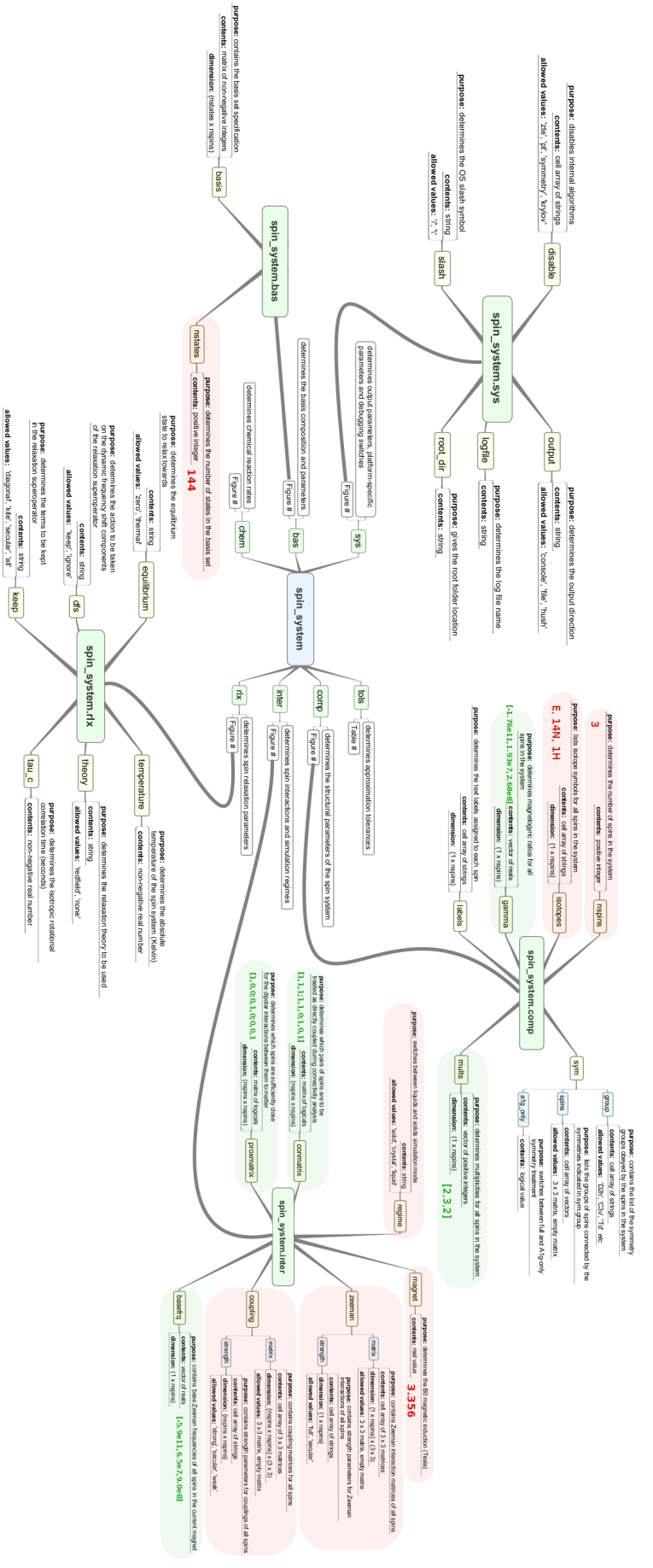


Figure 6: The Spinach spin-system data structure filled with attributes of an electron-nitrogen-proton spin system.

edited extract from http://spindynamics.org/documents/spinach_1.2.1217.zip

References

[1] Dr. Ilya Kuprov, *Spin Dynamics, Lecture 1 - 17*. University of Oxford, 2011.

[2] H.J. Hogben, M. Krzystyniak, G.T.P. Charnock, P.J. Hore, Ilya Kuprov, *Spinach - A software library for simulation of spin dynamics in large spin systems*. Journal of Magnetic Resonance, 208 (2011) 179-194.

[3] Theoretical Spin Dynamics Group, www.spindynamics.org, *Spinach package - input preparation manual*. University of Southampton, 2011.

[4] Corinne Cerf, *NMR Spectroscopy: From Quantum Mechanics to Protein Spectra*. Concepts in Magnetic Resonancem Vol. 9(1) 17-41, (1997)

[5] Baltzar Stevensson, Mattias Edén, *Efficient orientational averaging by the extension of Lebedev grids via regularized octahedral symmetry expansion*. Journal of Magnetic Resonance, 191 (2006) 162-176.

[6] David J. Griffiths, *Introduction to Quantum Mechanics*. Pearson, 2nd Edition, 2005.

[7] Paul T. Callaghan, *Translational Dynamics & Magnetic Resonance. Principles of Pulsed Gradient Spin Echo NMR*. Oxford University Press, 2011.

[8] Charles P. Slichter *Principles of Magnetic Resonance* Springer, 3rd Enlarged and Updated Edition, Corrected 3rd Printing, 1996.