SpinDoctor User Guide

Updated April 19, 2021

The user is advised to download the latest version of the User Guide from `https://github.com/jingrebeccali/SpinDoctor`.

Contents

1 Introduction

The MATLAB Toolbox SpinDoctor [1] is a simulation pipeline that

1. allows the user to define a geometrical configuration;
2. solves the Bloch-Torrey equation in that geometrical configuration;
3. fits the apparent diffusion coefficient from the simulated signal.

It includes two other methods for calculating the apparent diffusion coefficient:

1. The first is a homogenized apparent diffusion coefficient mathematical model, which was obtained recently using homogenization techniques on the Bloch-Torrey equation. In the homogenized model, the apparent diffusion coefficient of a geometrical configuration can be computed after solving a diffusion equation subject to a time-dependent Neumann boundary condition, under the assumption of negligible water exchange between compartments.
2. The second module computes the short time approximation formula for the apparent diffusion coefficient. The short time approximation implemented in SpinDoctor includes a recent generalization of this formula to account for finite pulse duration in the pulsed gradient spin echo.

Both of these two apparent diffusion coefficient calculations are sensitive to the diffusion-encoding gradient direction, unlike many previous works where the anisotropy is neglected in analytical model development.

In nutshell, SpinDoctor

1. solves the Bloch-Torrey equation in three dimensions to obtain the diffusion magnetic resonance imaging signal;
2. robustly fits the diffusion magnetic resonance imaging signal to obtain the apparent diffusion coefficient;
3. solves the homogenized apparent diffusion coefficient model in three dimensions to obtain the apparent diffusion coefficient;
4. computes the short-time approximation of the apparent diffusion coefficient;
5. computes useful geometrical quantities such as the compartment volumes and surface areas;
6. allows permeable membranes for the Bloch-Torrey equation (the homogenized apparent diffusion coefficient model assumes negligible permeability);
7. displays the gradient-direction dependent signal or apparent diffusion coefficient in three dimensions.

SpinDoctor provides the following built-in functionalities:

1. the placement of non-overlapping spherical cells (with an optional nucleus) of different radii close to each other;
2. the placement of non-overlapping cylindrical cells (with an optional myelin layer) of different radii close to each other in a canonical configuration where they are parallel to the z-axis;
3. the inclusion of an extra-cellular space that is enclosed either
   (a) in a tight wrapping around the cells; or
   (b) in a rectangular box.
4. the deformation of the canonical configuration by bending and twisting.

Built-in diffusion-encoding pulse sequences include

1. the Pulsed Gradient Spin Echo;
2. the Oscillating Gradient Spin Echo (cos- and sin- type gradients);
3. the Double Pulsed Gradient Spin Echo.

There is also support for custom diffusion-encoding pulse sequences.

SpinDoctor uses the following methods:

1. it generates a good quality surface triangulation of the user specified geometrical configuration by calling built-in MATLAB computational geometry functions;
2. it creates a good quality tetrahedra finite elements mesh from the above surface triangulation by calling Tetgen [2], an external package (executable files are included in the Toolbox package);
3. it constructs finite element matrices for linear finite elements on tetrahedra (P1 finite elements) using routines from [3];
4. it adds additional degrees of freedom on the compartment interfaces to allow permeability conditions for the Bloch-Torrey equation using the formalism in [4];
5. it solves the semi-discretized finite elements method equations by calling built-in MATLAB routines for solving ordinary differential equations.

Abbreviations

The following abbreviations are used throughout the text and code:

- ADC, Apparent Diffusion Coefficient;
- BT, Bloch-Torrey;
- BTPDE, Bloch-Torrey PDE;
- DMRI, Diffusion MRI;
- ECS, Extra-Cellular Space;
- FEM, Finite Element Method;
- FPK, Finite Pulse Karger model;
- HADC, homogenized ADC;
- HARDI, High Angular Resolution Diffusion Imaging;
- MF, Matrix Formalism;
- MRI, Magnetic Resonance Imaging;
- ODE, Ordinary Differential Equation;
- OGSE, Oscillating Gradient Spin Echo;
- PDE, Partial Differential Equation;
- PGSE, Pulsed-Gradient Spin Echo;
- STA, Short Time Approximation.

2 Installation

The SpinDoctor Toolbox has been tested with MATLAB R2018a-R2021a. The user is advised to download the latest version of the User Guide from `https://github.com/jingrebeccali/SpinDoctor`. Examples of drivers that run some typical simulations also can be found there.

The SpinDoctor Toolbox has been developed in the MATLAB R2020b and require no additional MATLAB toolboxes. SpinDoctor has support for parallel computations, if the MATLAB Parallel Computing Toolbox is available.

3 Mathematical background

Consider a domain $\Omega = \bigcup_{i=1}^{N_{\text{cmpt}}} \Omega_i$ of N_{cmpt} compartments $\{\Omega_i\}_{1\leq i\leq N_{\text{cmpt}}}$. Let $\Gamma_{ij} = \Omega_i \cap \Omega_j$ denote the interface between two compartments for $1 \leq i, j \leq N_{\text{cmpt}}$, $i \neq j$. Note that for compartments that do not touch, we have $\Gamma_{ij} = \emptyset$. Similarly, let $\Gamma_{ii} = \emptyset$ for the ease of notation. Finally, let $\partial\Omega$ denote the outer boundary of the domain.

3.1 Bloch-Torrey PDE

In diffusion MRI, a time-varying magnetic field gradient is applied to the tissue to encode water diffusion. Denoting the effective time profile of the diffusion-encoding magnetic field gradient by $f : [0, T_{\text{echo}}] \to [-1, 1]$, and letting the vector $\boldsymbol{g}$ contain the amplitude and direction information of the magnetic field gradient, the complex transverse water proton magnetization in the rotating frame satisfies the Bloch-Torrey PDE in each compartment Ω_i, $1 \leq i \leq N_{\text{cmpt}}$:

$$\frac{\partial}{\partial t} M_i(\boldsymbol{x}, t) = -\left(-\nabla \cdot \mathbf{D}_i \nabla + \frac{1}{\tau_i} + \underline{\mathrm{i}}\gamma f(t)\boldsymbol{g} \cdot \boldsymbol{x}\right) M_i(\boldsymbol{x}, t), \quad (\boldsymbol{x}, t) \in \Omega_i \times [0, T_{\text{echo}}], \tag{1}$$

where $\gamma = 2.67513 \times 10^8\,\text{rad}\,\text{s}^{-1}\text{T}^{-1}$ is the gyromagnetic ratio of the water proton and $\underline{\mathrm{i}}$ is the imaginary unit. For each compartment Ω_i, $1 \leq i \leq N_{\text{cmpt}}$, $\mathbf{D}_i$ is the intrinsic diffusion tensor, τ_i is the T_2-relaxation time and M_i is the magnetization. The diffusion tensors are assumed to be symmetric positive definite, and the relaxation times positive. We also allow for $\tau_i = \infty$, in which case the corresponding term is removed from the equation.

The magnetization is a function of position $\boldsymbol{x}$ and time t, and depends on the diffusion gradient vector $\boldsymbol{g}$ and the time profile f. The initial spin density is given by

$$M_i(\boldsymbol{x}, 0) = \rho_i \in \mathbb{C}, \quad 1 \leq i \leq N_{\text{cmpt}}. \tag{2}$$

The outer boundary conditions for the BTPDE are given by

$$\mathbf{D}_i \nabla M_i(\boldsymbol{x}, t) \cdot \boldsymbol{n}_i(\boldsymbol{x}) = -\kappa_i M_i(\boldsymbol{x}, t), \quad \boldsymbol{x} \in \Omega_i \cap \partial\Omega, \tag{3}$$

where $\boldsymbol{n}_i$ is the unit outward pointing normal vector of compartment Ω_i and κ_i is an outer boundary relaxation coefficient for Ω_i. Note that we may have $\Omega_i \cap \partial\Omega = \emptyset$, as all the compartments do not necessarily touch the outer boundary.

The BTPDE also needs to be supplemented by interface conditions. We recall that the interface between Ω_i and Ω_j is Γ_{ij}. The two interface conditions on Γ_{ij} are the flux continuity and a condition that incorporates a permeability coefficient κ_{ij} across Γ_{ij}:

$$\mathbf{D}_i \nabla M_i(\boldsymbol{x}, t) \cdot \boldsymbol{n}_i(\boldsymbol{x}) = -\mathbf{D}_j \nabla M_j(\boldsymbol{x}, t) \cdot \boldsymbol{n}_j(\boldsymbol{x}), \qquad \boldsymbol{x} \in \Gamma_{ij}, \tag{4}$$

$$\mathbf{D}_i \nabla M_i(\boldsymbol{x}, t) \cdot \boldsymbol{n}_i(\boldsymbol{x}) = \kappa_{ij}\left(c_{ij} M_j(\boldsymbol{x}, t) - c_{ji} M_i(\boldsymbol{x}, t)\right), \quad \boldsymbol{x} \in \Gamma_{ij}, \tag{5}$$

Here, the permeability coefficient characterizes the membrane: $\kappa_{ij} = \kappa_{ji}$. The two weights c_{ij} and c_{ji} account for the spin density equilibrium between the two compartments. These may both be set to 1, in which case a uniform spin density across compartments is favored in the absence of a gradient. For different initial spin densities ρ_i and ρ_j, we also allow for non-symmetrical weights, for example[5]

$c_{ij} = \frac{2\rho_i}{\rho_i+\rho_j}$ and $c_{ji} = \frac{2\rho_j}{\rho_i+\rho_j}$. This ensures that the non-uniform intitial spin density is preserved if the gradient $\boldsymbol{g}$ is zero, as c_{ij} and c_{ji} are proportional to ρ_i and ρ_j respectively. The normalization coefficient $2/(\rho_i + \rho_j)$ ensures that $c_{ij} = c_{ji} = 1$ if $\rho_i = \rho_j$.

For the ease of notation, inside volume integrals, we will use the quantities $\mathbf{D}$, τ, and M (defined almost everywhere):

$$\forall i \in \{1, \ldots, N_{\text{cmpt}}\} : \begin{cases} \mathbf{D}(\boldsymbol{x}) = \mathbf{D}_i, & \boldsymbol{x} \in \Omega_i \\ \tau(\boldsymbol{x}) = \tau_i, & \boldsymbol{x} \in \Omega_i \\ \rho(\boldsymbol{x}) = \rho_i, & \boldsymbol{x} \in \Omega_i \\ M(\boldsymbol{x}, t) = M_i(\boldsymbol{x}, t), & (\boldsymbol{x}, t) \in \Omega_i \times [0, T_{\text{echo}}]. \end{cases} \tag{6}$$

In equation (1), the minus sign is factored out of right hand side as, with the prescribed boundary conditions, the *Bloch-Torrey operator*,

$$-\nabla \cdot \mathbf{D}\nabla + \frac{1}{\tau} + \mathrm{i}\gamma f(t)\boldsymbol{g} \cdot \boldsymbol{x},$$

has eigenvalues with nonnegative real parts for all t, f, $\boldsymbol{g}$, and positive τ and $\mathbf{D}$.

3.2 Diffusion MRI model

The diffusion MRI signal is measured at echo time $t = T_{\text{echo}}$. This signal is computed as the spatial integral of $M(\cdot, T_{\text{echo}})$:

$$S = \int_\Omega M(\boldsymbol{x}, T_{\text{echo}}) \, \mathrm{d}\Omega(\boldsymbol{x}), \tag{7}$$

and represents the output of the SpindDoctor diffusion MRI model. The model input is the gradient sequence, encoded by the amplitude and direction $\boldsymbol{g}$ and effective time profile f. Some commonly used time profiles (diffusion-encoding sequences) are:

1. The pulsed-gradient spin echo (PGSE) [6] sequence, with two rectangular pulses of duration δ, separated by a time interval $\Delta - \delta$, for which the profile $f(t)$ is

$$f(t) = \begin{cases} 1, & t_1 \le t \le t_1 + \delta, \\ -1, & t_1 + \Delta < t \le t_1 + \Delta + \delta, \\ 0, & \text{otherwise}, \end{cases} \tag{8}$$

2. The oscillating gradient spin echo (OGSE) sequence [7, 8] was introduced to reach short diffusion times. An OGSE sequence usually consists of two oscillating pulses of duration δ, each containing n periods, hence the frequency is $\omega = n\frac{2\pi}{\delta}$, separated by a time interval $\Delta - \delta$. For a cosine OGSE, the profile $f(t)$ is

$$f(t) = \begin{cases} \cos\left(n\frac{2\pi}{\delta}(t - t_1)\right), & t_1 < t \le t_1 + \delta, \\ -\cos\left(n\frac{2\pi}{\delta}(t - \Delta - t_1)\right), & t_1 + \Delta < t \le t_1 + \Delta + \delta, \\ 0, & \text{otherwise}, \end{cases} \tag{9}$$

where t_1 is the starting time of the first gradient pulse and $t_1 + \Delta \ge T_{\text{echo}}/2$. By default, SpinDoctor assumes $t_1 = 0$ and $T_{\text{echo}} = \Delta + \delta$, meaning that ρ denotes the spin density just before the gradient sequence is applied and S is measured immediately after the end of the pulses. The user can however set up custom sequences taking into account the times before and after the pulses, or model the pulse transitions.

In a dMRI experiment, the pulsed sequence (time profile f) is usually fixed, while $\boldsymbol{g}$ is varied in amplitude (and possibly also in direction). S is usually plotted against a quantity called the b-value. The b-value depends on $\boldsymbol{g}$ and f and is defined as

$$b(\boldsymbol{g}, f) = \gamma^2 \|\boldsymbol{g}\|^2 \int_0^{T_{\text{echo}}} \mathrm{d}t \left(\int_0^t f(s) \, \mathrm{d}s \right)^2.$$

For PGSE, the b-value is [6]:

$$b(\boldsymbol{g}, \delta, \Delta) = \gamma^2 \|\boldsymbol{g}\|^2 \delta^2 \left(\Delta - \delta/3\right). \tag{10}$$

For the cosine OGSE with *integer* number of periods n in each of the two durations δ, the corresponding b-value is [9]:

$$b(\boldsymbol{g}, \delta) = \gamma^2 \|\boldsymbol{g}\|^2 \frac{\delta^3}{4n^2\pi^2} = \gamma^2 \|\boldsymbol{g}\|^2 \frac{\delta}{\omega^2}. \tag{11}$$

The reason for these definitions is that in a homogeneous medium, the signal attenuation is $\mathrm{e}^{-\boldsymbol{u}_{\boldsymbol{g}}^{\mathsf{T}} \mathbf{D}_0 \boldsymbol{u}_{\boldsymbol{g}} b}$, where $\mathbf{D}$ is the intrinsic diffusion tensor and $\boldsymbol{u}_{\boldsymbol{g}} = \boldsymbol{g}/\|\boldsymbol{g}\|$.

3.3 Fitting the ADC from the dMRI signal

An important quantity that can be derived from the dMRI signal is the "Apparent Diffusion Coefficient" (ADC), which gives an indication of the root mean squared distance travelled by water molecules in the gradient direction $\boldsymbol{g}/\|\boldsymbol{g}\|$, averaged over all starting positions:

$$D_{\mathrm{ADC}} = -\frac{\partial}{\partial b} \log \frac{S(b)}{S(0)}\bigg|_{b=0}. \tag{12}$$

We numerically compute ADC by a polynomial fit of

$$\log \frac{S(b)}{S_{\mathrm{initial}}} \approx c_0 + c_1 b + \cdots + c_n b^n,$$

increasing n from 1 onwards until we get the value of c_1 to be stable within a numerical tolerance, where $S_{\mathrm{initial}} = \int_\Omega \rho(\boldsymbol{x}) \, \mathrm{d}\Omega(\boldsymbol{x})$. The first coefficient is given by $c_0 = \log \frac{S(0)}{S_{\mathrm{initial}}}$, which is equal to zero if $\tau_i = \infty$ for all i.

The ADC may be interpreted as a correction to the intrinsic diffusion coefficient, taking into account the deviation from the free diffusion arising from interior interfaces, non-isotropic diffusion, fluid movements etc. For suffiently small b-values, the signal attenuation is given by

$$\mathrm{e}^{-D_{\mathrm{ADC}} b}.$$

Another quantity of interest is a slight generalization of the ADC—an effective diffusion tensor $\mathbf{D}_{\mathrm{eff}}$. The six coefficients of this symmetric positive tensor is fitted to best approximate the following signal attenuation, for all gradients $\boldsymbol{g} \in \mathbb{R}^3$:

$$\mathrm{e}^{-\frac{\boldsymbol{g}^{\mathsf{T}} \mathbf{D}_{\mathrm{eff}} \boldsymbol{g}}{\boldsymbol{g}^{\mathsf{T}} \boldsymbol{g}} b}.$$

The resulting ADC in direction $\boldsymbol{g}$ is then given by

$$D_{\mathrm{ADC}}(\boldsymbol{g}) = \frac{\boldsymbol{g}^{\mathsf{T}} \mathbf{D}_{\mathrm{eff}} \boldsymbol{g}}{\boldsymbol{g}^{\mathsf{T}} \boldsymbol{g}} = \boldsymbol{u}_{\boldsymbol{g}}^{\mathsf{T}} \mathbf{D}_{\mathrm{eff}} \boldsymbol{u}_{\boldsymbol{g}}. \tag{13}$$

3.4 HADC model

In a previous work [10], a PDE model for the time-dependent ADC was obtained starting from the Bloch-Torrey equation, using homogenization techniques. In the case of negligible water exchange between compartments (low permeability), there is no coupling between the compartments, at least to the quadratic order in $\boldsymbol{g}$, which is the ADC term. The ADC in compartment Ω is given by

$$D_{\mathrm{HADC}}(\boldsymbol{g}, f) = \boldsymbol{u}_{\boldsymbol{g}}^{\mathsf{T}} \mathbf{D} \boldsymbol{u}_{\boldsymbol{g}} - \frac{\int_0^{T_{\mathrm{echo}}} F(t) h(t) \, \mathrm{d}t}{\int_0^{T_{\mathrm{echo}}} F(t)^2 \, \mathrm{d}t}, \tag{14}$$

where $F(t) = \int_0^t f(s) \, \mathrm{d}s$, and

$$h(t) = \frac{1}{|\Omega|} \int_{\partial\Omega} \omega(\boldsymbol{x}, t) \, \boldsymbol{u}_{\boldsymbol{g}} \cdot \boldsymbol{n}(\boldsymbol{x}) \, \mathrm{d}\Gamma(\boldsymbol{x}) \tag{15}$$

is a quantity related to the directional gradient of a function ω that is the solution of the homogeneous diffusion equation with Neumann boundary condition and zero initial condition:

$$\frac{\partial}{\partial t}\omega(\boldsymbol{x},t) = \nabla\cdot\left(\mathbf{D}\nabla\omega(\boldsymbol{x},t)\right), \quad (\boldsymbol{x},t)\in\Omega\times[0,T_{\text{echo}}], \tag{16}$$

$$\omega(\boldsymbol{x},0) = 0, \qquad \boldsymbol{x}\in\Omega, \tag{17}$$

$$\mathbf{D}\nabla\omega(\boldsymbol{x},t)\cdot\boldsymbol{n} = F(t)\mathbf{D}\boldsymbol{u_g}\cdot\boldsymbol{n}, \qquad \boldsymbol{x}\in\partial\Omega, \tag{18}$$

$\boldsymbol{n}$ being the outward normal and $\boldsymbol{u_g}$ the unit gradient direction. The above set of equations, (14)-(18), comprise the homogenized model that we call the HADC model.

3.5 Short diffusion time approximation of the ADC

A well-known formula for the ADC in the short diffusion time regime is the following short time approximation (STA) [11, 12]:

$$D_{\text{STA}} = \left(1 - \frac{4\sqrt{D_0}}{3\sqrt{\pi}}\sqrt{\Delta}\frac{A}{d\,V}\right)D_0,$$

where $\dfrac{A}{V}$ is the surface to volume ratio, d is the spatial dimension and $D_0 = \boldsymbol{u_g}^{\mathsf{T}}\mathbf{D}\boldsymbol{u_g}$ is the intrinsic diffusion coefficient in the gradient direction. In the above formula, the pulse duration δ is assumed to be very small compared to Δ. A recent correction to the above formula [10], taking into account the finite pulse duration δ and the gradient direction $\boldsymbol{u_g}$, is the following:

$$D_{\text{STA}} = \left(1 - \frac{4\sqrt{D_0}}{3\sqrt{\pi}}C_{\delta,\Delta}\frac{A_{\boldsymbol{u_g}}}{V}\right)D_0, \tag{19}$$

where

$$A_{\boldsymbol{u_g}} = \int_{\partial\Omega}\left(\boldsymbol{u_g}\cdot\boldsymbol{n}(\boldsymbol{x})\right)^2\,\mathrm{d}\Gamma(\boldsymbol{x}) = \boldsymbol{u_g}^{\mathsf{T}}\mathbf{N}\boldsymbol{u_g}$$

with $\mathbf{N} = \int_{\partial\Omega}\boldsymbol{n}(\boldsymbol{x})\boldsymbol{n}^{\mathsf{T}}(\boldsymbol{x})\,\mathrm{d}\Gamma(\boldsymbol{x})$ and

$$C_{\delta,\Delta} = \frac{4}{35}\frac{(\Delta+\delta)^{7/2} + (\Delta-\delta)^{7/2} - 2\left(\delta^{7/2}+\Delta^{7/2}\right)}{\delta^2\left(\Delta-\delta/3\right)} = \sqrt{\Delta}\left(1 + \frac{1}{3}\frac{\delta}{\Delta} - \frac{8}{35}\left(\frac{\delta}{\Delta}\right)^{3/2} + \dots\right).$$

When $\delta \ll \Delta$, the value $C_{\delta,\Delta}$ is approximately $\sqrt{\Delta}$.

3.6 Matrix Formalism signal representation

The Matrix Formalism [13, 14] representation of the solution to the Bloch-Torrey PDE is based on the generalized Laplace eigenfunctions of the given geometrical configuration. These are the eigenfunctions of the generalized Laplace operator $-\nabla\cdot\mathbf{D}\nabla$ subject to the same boundary and interface conditions as the BTPDE. It which does not depend on the gradient sequence. The Matrix Formalism truncates the number of Laplace eigenmodes at an acceptable level, by setting a minimum length scale for the eigenfunctions. The gradient sequence dependent Bloch-Torrey operator can then be expressed in the truncated Laplace eigenfunction basis. This allows for a simple analytical expression which approximates the solution to the BTPDE down to an arbitrary length scale.

3.6.1 Eigenvalue decomposition of the generalized Laplace operator

Let $\{(\phi,\lambda)\}$ be the L^2-normalized eigenfunctions and eigenvalues of the generalized Laplace operator $\nabla\cdot\mathbf{D}\nabla$, defined almost everywhere on the domain $\Omega = \bigcup_{i=1}^{N_{\text{cmpt}}}\Omega_i$. Denoting ϕ^i the restriction of ϕ to compartment Ω_i, the eigenfunctions respect the following set of equations, for $(i,j)\in\{1,\dots,N_{\text{cmpt}}\}^2$:

$$-\nabla\cdot\mathbf{D}_i\nabla\phi^i(\boldsymbol{x}) = \lambda\phi^i(\boldsymbol{x}), \quad \boldsymbol{x}\in\Omega_i, \tag{20}$$

where the boundary conditions are the same as for the BTPDE:

$$\mathbf{D}_i \nabla \phi^i(\boldsymbol{x}) \cdot \boldsymbol{n}_i(\boldsymbol{x}) = -\mathbf{D}_j \nabla \phi^j(\boldsymbol{x}) \cdot \boldsymbol{n}_j(\boldsymbol{x}), \qquad \boldsymbol{x} \in \Gamma_{ij}, \tag{21}$$

$$\mathbf{D}_i \nabla \phi^i(\boldsymbol{x}) \cdot \boldsymbol{n}_i(\boldsymbol{x}) = \kappa_{ij} \left(c_{ij} \phi^j(\boldsymbol{x}) - c_{ji} \phi^i(\boldsymbol{x}) \right), \quad \boldsymbol{x} \in \Gamma_{ij}, \tag{22}$$

$$\mathbf{D}_i \nabla \phi^i(\boldsymbol{x}) \cdot \boldsymbol{n}_i(\boldsymbol{x}) = -\kappa_i \phi^i(\boldsymbol{x}), \qquad \boldsymbol{x} \in \Omega_i \cap \partial\Omega. \tag{23}$$

Let the solutions (ϕ, λ) to the above equations, (20)-(23), be denoted by $\{(\phi_n, \lambda_n)\}_{n \in \mathbb{N}^*}$. We assume the non-negative real-valued eigenvalues are ordered in non-decreasing order:

$$0 = \lambda_1 < \lambda_2 \leq \lambda_3 \leq \dots$$

so that $\lambda_1 = 0$ (this means the first Laplace eigenfunction is the constant function). There is only one constant eigenfunction because for simplicity, we assume that the domain Ω is connected through permeable membranes (with positive permeability coefficients).

Let $\mathbf{L}$ be the diagonal matrix containing the first N_{eig} Laplace eigenvalues:

$$\mathbf{L} = \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_{N_{\text{eig}}}) \in \mathbb{R}^{N_{\text{eig}} \times N_{\text{eig}}}. \tag{24}$$

Then the matrix $\mathbf{L}$ represents the generalized Laplace operator $-\nabla \cdot \mathbf{D} \nabla$ in the truncated Laplace eigenfunction basis.

3.6.2 Bloch-Torrey operator in the Laplace eigenfunction basis

Let $\mathbf{A}(\boldsymbol{g})$ be the $N_{\text{eig}} \times N_{\text{eig}}$ matrix defined by:

$$\mathbf{A}(\boldsymbol{g}) = g_x \mathbf{A}^x + g_y \mathbf{A}^y + g_z \mathbf{A}^z, \tag{25}$$

where $\boldsymbol{g} = (g_x, g_y, g_z)^\mathsf{T}$ is the gradient vector and $\mathbf{A}^x$, $\mathbf{A}^y$, and $\mathbf{A}^z$ are three symmetric $N_{\text{eig}} \times N_{\text{eig}}$ matrices whose entries are the first order moments in the coordinate directions of the product of pairs of eigenfunctions:

$$\begin{aligned} A^x_{mn} &= \int_\Omega x \, \phi_m(\boldsymbol{x}) \phi_n(\boldsymbol{x}) \, \mathrm{d}\Omega(\boldsymbol{x}), \\ A^y_{mn} &= \int_\Omega y \, \phi_m(\boldsymbol{x}) \phi_n(\boldsymbol{x}) \, \mathrm{d}\Omega(\boldsymbol{x}), \quad (m,n) \in \{1, \dots, N_{\text{eig}}\}^2. \\ A^z_{mn} &= \int_\Omega z \, \phi_m(\boldsymbol{x}) \phi_n(\boldsymbol{x}) \, \mathrm{d}\Omega(\boldsymbol{x}), \end{aligned} \tag{26}$$

Similarly, let $\mathbf{T}$ be the $N_{\text{eig}} \times N_{\text{eig}}$ Laplace relaxation matrix defined by

$$T_{mn} = \int_\Omega \frac{1}{\tau(\boldsymbol{x})} \phi_m(\boldsymbol{x}) \phi_n(\boldsymbol{x}) \, \mathrm{d}\Omega(\boldsymbol{x}), \quad (m,n) \in \{1, \dots, N_{\text{eig}}\}^2. \tag{27}$$

Then the general time dependent Bloch-Torrey operator

$$-\nabla \cdot \mathbf{D} \nabla + \frac{1}{\tau} + \underline{\mathrm{i}} \gamma f(t) \boldsymbol{g} \cdot \boldsymbol{x}, \quad t \in [0, T_{\text{echo}}]$$

in the truncated Laplace eigenfunction basis $(\phi_j)_{j=1,\dots,N_{\text{eig}}}$ is given by the complex-valued matrix

$$\mathbf{K}(\boldsymbol{g}, f)(t) = \mathbf{L} + \mathbf{T} + \underline{\mathrm{i}} \gamma f(t) \mathbf{A}(\boldsymbol{g}), \quad t \in [0, T_{\text{echo}}]. \tag{28}$$

For a constant time profile (PGSE, double-PGSE), the Bloch-Torrey operator

$$-\nabla \cdot \mathbf{D} \nabla + \frac{1}{\tau} + \underline{\mathrm{i}} \gamma \boldsymbol{g} \cdot \boldsymbol{x}$$

is constant. In the truncated Laplace eigenfunction basis, it is given by the complex-valued matrix

$$\mathbf{K}(\boldsymbol{g}) = \mathbf{L} + \mathbf{T} + \underline{\mathrm{i}} \gamma \mathbf{A}(\boldsymbol{g}). \tag{29}$$

3.6.3 Matrix Formalism approximation

Denoting $\boldsymbol{\phi} = (\phi_1, \ldots, \phi_{N_{\text{eig}}})^{\mathsf{T}}$ the vector of the first N_{eig} Laplace eigenfunctions, we may project the solution to to the Bloch-Torrey PDE (1) onto the truncated Laplace eigenfunction basis:

$$M^{\text{MF}}(\boldsymbol{x}, t) = \boldsymbol{\phi}^{\mathsf{T}}(\boldsymbol{x})\boldsymbol{\nu}(t), \tag{30}$$

where the vector of coefficients $\boldsymbol{\nu} = (\nu_1, \ldots, \nu_{N_{\text{eig}}})^{\mathsf{T}} \in \mathbb{C}^{N_{\text{eig}}}$ is the solution to

$$\frac{\partial \boldsymbol{\nu}}{\partial t} = -\mathbf{K}(\boldsymbol{g}, f)(t)\boldsymbol{\nu}(t), \quad t \in [0, T_{\text{echo}}]. \tag{31}$$

The initial coefficients are given by

$$\boldsymbol{\nu}(0) = \int_\Omega \rho(\boldsymbol{x})\boldsymbol{\phi}(\boldsymbol{x}) \, \mathrm{d}\Omega(\boldsymbol{x}). \tag{32}$$

By using a piece-wise constant approximation [14] of the time profile f, we obtain

$$\boldsymbol{\nu}(T_{\text{echo}}) \approx \left(\prod_{i=1}^{N_{\text{int}}} \mathrm{e}^{-\delta_i \mathbf{K}_i} \right) \boldsymbol{\nu}(0) = \mathrm{e}^{-\delta_{N_{\text{int}}} \mathbf{K}_{N_{\text{int}}}} \ldots \mathrm{e}^{-\delta_2 \mathbf{K}_2} \mathrm{e}^{-\delta_1 \mathbf{K}_1} \boldsymbol{\nu}(0), \tag{33}$$

where $\{I_i\}_{i=1,\ldots,N_{\text{int}}}$ are intervals such that $[0, T_{\text{echo}}] = \bigcup_{i=1}^{N_{\text{int}}} I_i$, $f(t) = f_i$ for $t \in I_i$, $\delta_i = |I_i|$, and $\mathbf{K}_i(\boldsymbol{g}) = \mathbf{L} + \mathrm{i}\gamma f_i \mathbf{A}(\boldsymbol{g})$. The constants may be computed through quadrature:

$$f_i = \frac{1}{\delta_i} \int_{I_i} f(t) \, \mathrm{d}t \approx \frac{1}{2} \left(f(\min I_i) + f(\max I_i) \right). \tag{34}$$

For the PGSE and double-PGSE sequences, the coefficients of the final magnetization are given by

$$\boldsymbol{\nu}(T_{\text{echo}}) = \mathrm{e}^{-\delta \mathbf{K}^*} \mathrm{e}^{-(\Delta - \delta)(\mathbf{L} + \mathbf{T})} \mathrm{e}^{-\delta \mathbf{K}} \boldsymbol{\nu}(0) \tag{35}$$

and

$$\boldsymbol{\nu}(T_{\text{echo}}) = \mathrm{e}^{-\delta \mathbf{K}^*} \mathrm{e}^{-(\Delta - \delta)(\mathbf{L} + \mathbf{T})} \mathrm{e}^{-\delta \mathbf{K}} \mathrm{e}^{-\delta \mathbf{K}^*} \mathrm{e}^{-(\Delta - \delta)(\mathbf{L} + \mathbf{T})} \mathrm{e}^{-\delta \mathbf{K}} \boldsymbol{\nu}(0) \tag{36}$$

respectively. These are the exact solutions to Eq. (31), although there may still be truncation errors from N_{eig}. We note that Eq. (35) and Eq. (36) are particular cases of Eq. (33), with ($N_{\text{int}} = 3$, $\mathcal{I}_1 = [0, \delta]$, $\mathcal{I}_2 = [\delta, \Delta]$, $\mathcal{I}_3 = [\Delta, \Delta + \delta]$) and ($N_{\text{int}} = 6$, $\mathcal{I}_1 = [0, \delta]$, $\mathcal{I}_2 = [\delta, \Delta]$, $\mathcal{I}_3 = [\Delta, \Delta + \delta]$, $\mathcal{I}_4 = [\Delta + \delta, \Delta + 2\delta]$, $\mathcal{I}_5 = [\Delta + 2\delta, 2\Delta + \delta]$, $\mathcal{I}_6 = [2\Delta + \delta, 2\Delta + 2\delta]$) respectively.

The notation * denotes the matrix complex conjugate transpose. In order to calculate the non-diagonal matrix exponential $\mathrm{e}^{-\delta \mathbf{K}}$ and its conjugate $\mathrm{e}^{-\delta \mathbf{K}^*} = \left(\mathrm{e}^{-\delta \mathbf{K}}\right)^*$, the scaling and squaring method [15, 16] is used (the built-in matrix exponential function in MATLAB, `expm`, is called). However, for the purpose of theoretical analysis, we may explicitly diagonalize the matrix $\mathbf{K}(\boldsymbol{g})$:

$$\mathbf{K}(\boldsymbol{g}) = \mathbf{V}\mathbf{B}\mathbf{V}^{-1}, \tag{37}$$

where $\mathbf{V}$ has the eigenvectors in the columns and $\mathbf{B}$ has the eigenvalues on the diagonal. Then $\mathrm{e}^{-\delta \mathbf{K}} = \mathbf{V}\mathrm{e}^{-\delta \mathbf{B}}\mathbf{V}^{-1}$ and $\mathrm{e}^{-\delta \mathbf{K}^*} = \left(\mathbf{V}^{-1}\right)^* \mathrm{e}^{-\delta \mathbf{B}^*} \mathbf{V}^*$.

The signal is given by

$$S^{\text{MF}}(f, \boldsymbol{g}) = \sqrt{|\Omega|} \, \nu_1(T_{\text{echo}}) \tag{38}$$

where ν_1 is the first entry of $\boldsymbol{\nu}$, as $\int_\Omega \boldsymbol{\phi}(\boldsymbol{x}) \, \mathrm{d}\Omega(\boldsymbol{x}) = (\sqrt{|\Omega|}, 0, \ldots, 0)^{\mathsf{T}}$. Only the first Laplace component remains after integrating, since the first eigenfunction is constant, and the other eigenfunctions are orthogonal to ϕ_1.

3.6.4 Apparent diffusion coefficient

From the Matrix Formalism signal, the analytical expression of its ADC for a gradient direction given by a unit vector $\boldsymbol{u_g}$ and a sequence f is the following:

$$D^{\text{MF}}_{\text{ADC}}(\boldsymbol{u_g}, f) = D_0 \sum_{n=1}^{N_{\text{eig}}} \frac{(\boldsymbol{u_g} \cdot \boldsymbol{a}_{1n})^2 \lambda_n \int_0^{T_{\text{echo}}} F(t) \left(\int_0^t \mathrm{e}^{-D_0 \lambda_n (t-s)} f(s) \, \mathrm{d}s \right) \mathrm{d}t}{\int_0^{T_{\text{echo}}} F^2(t) \, \mathrm{d}t},$$

where the coefficients $\boldsymbol{a}_{1n} = (A^x_{1n}, A^y_{1n}, A^z_{1n})^\mathsf{T}$ are given by

$$\begin{aligned} A^x_{1n} &= \frac{1}{\sqrt{|\Omega|}} \int_\Omega x\phi_n(\boldsymbol{x}) \, \mathrm{d}\Omega(\boldsymbol{x}), \\ A^y_{1n} &= \frac{1}{\sqrt{|\Omega|}} \int_\Omega y\phi_n(\boldsymbol{x}) \, \mathrm{d}\Omega(\boldsymbol{x}), \\ A^z_{1n} &= \frac{1}{\sqrt{|\Omega|}} \int_\Omega z\phi_n(\boldsymbol{x}) \, \mathrm{d}\Omega(\boldsymbol{x}). \end{aligned} \tag{39}$$

We remind the reader that the first Laplace eigenfunction is the constant function.

To clarify the relationship between the ADC and the diffusion encoding direction $\boldsymbol{u_g}$, we rewrite the Matrix Formalism ADC as

$$D^{\mathrm{MF}}_{\mathrm{ADC}}(\boldsymbol{u_g}, f) = \boldsymbol{u_g}^\mathsf{T} \mathbf{D}^{\mathrm{MF}}(f) \boldsymbol{u_g}, \tag{40}$$

where the Matrix Formalism effective diffusion tensor is given by

$$\mathbf{D}^{\mathrm{MF}}(f) = D_0 \sum_{n=1}^{N_{\mathrm{eig}}} J(\lambda_n, f) \boldsymbol{a}_{1n} \boldsymbol{a}_{1n}^\mathsf{T}, \tag{41}$$

with $J(\lambda_n, f)$ depending on λ_n and f:

$$J(\lambda_n, f) = \frac{\lambda_n \int_0^{T_{\mathrm{echo}}} F(t) \left(\int_0^t e^{-D_0 \lambda_n (t-s)} f(s) \, \mathrm{d}s \right) \, \mathrm{d}t}{\int_0^{T_{\mathrm{echo}}} F^2(t) \, \mathrm{d}t}. \tag{42}$$

In the case of a constant initial spin density ρ, we also allow for computing the Matrix Formalism Gaussian Approximation (MFGA) signal, given as

$$S^{\mathrm{MFGA}}(\boldsymbol{g}, f) = \rho|\Omega| \exp\left(-\boldsymbol{u_g}^\mathsf{T} \mathbf{D}^{\mathrm{MF}}(f) \boldsymbol{u_g} \, b(\|\boldsymbol{g}\|, f) \right), \quad \boldsymbol{u_g} = \frac{\boldsymbol{g}}{\|\boldsymbol{g}\|}. \tag{43}$$

3.6.5 Eigenfunction length scale and orientation

On a line segment of length L and diffusivity D_0, the eigenvalues $(\lambda_1, \lambda_2, \dots)$ of the Laplace operator with Neumann boundary conditions are

$$\lambda_n = \left(\frac{\pi(n-1)}{L} \right)^2 D_0, \quad n = 1, 2, \dots \tag{44}$$

To make the link between the computed eigenvalue and the spatial scale of the eigenmode, we will convert the computed λ_n into a length scale:

$$\ell(\lambda) = \pi \sqrt{\frac{\bar{\sigma}}{\lambda}}, \tag{45}$$

and characterize the computed eigenmode by $\ell(\lambda_n)$ instead of λ_n. The reference diffusivity $\bar{\sigma}$ is taken as a volume weighted mean of the trace average (sometimes called hydrostatic part) of $\mathbf{D}^{\mathrm{in}}$, $\mathbf{D}^{\mathrm{out}}$, and $\mathbf{D}^{\mathrm{ecs}}$:

$$\bar{\sigma} = \frac{1}{|\Omega|} \int_\Omega \frac{1}{3} \operatorname{tr}\left(\mathbf{D}(\boldsymbol{x}) \right) \, \mathrm{d}\Omega(\boldsymbol{x}). \tag{46}$$

To characterize the directional contribution of the eigenmode we use the fact that its contribution to the ADC in the direction $\boldsymbol{u_g}$ is $J(\lambda_n, f)(\boldsymbol{u_g} \cdot \boldsymbol{a}_{1n})^2$. We thus call $\boldsymbol{a}_{1n} = (A^x_{1n}, A^y_{1n}, A^z_{1n})^\mathsf{T}$ the "diffusion direction" of the nth eigenmode. We remind that the three components of $\boldsymbol{a}_{1n}$ are the first moments in the 3 canonical basis directions of the associated eigenfunction.

3.7 Finite element discretization

In SpinDoctor, the finite element mesh generation is performed using an external package called Tetgen[2]. Each finite element mesh consists of

1. a list of N_{node} nodes in three dimensions: $(\boldsymbol{q}_1, \ldots, \boldsymbol{q}_{N_{\text{node}}}) = (\boldsymbol{q}^x, \boldsymbol{q}^y, \boldsymbol{q}^z)^\mathsf{T} \in \mathbb{R}^{3\times N_{\text{node}}}$;
2. a list of N_{element} tetrahedral elements ($4 \times N_{\text{element}}$ indices referencing the nodes).

The list of nodes includes double nodes that are placed at the interfaces between compartments connected by permeable membranes. This allows for representing the two magnetization fields M_i and M_j (or ϕ^i and ϕ^j) on the same boundary Γ_{ij}. To distinguish between the different compartments, let $\{1, \ldots, N_{\text{node}}\} = \bigcup_{i=1}^{N_{\text{cmpt}}} \mathcal{I}_i$ with $\mathcal{I}_i \cap \mathcal{I}_j = \emptyset$ for $i \neq j$. The set $\mathcal{I}_i$ contains the indices of the nodes representing compartment Ω_i, including interface nodes (but not the corresponding double nodes, that represent the neighboring compartment).

In SpinDoctor[1], the finite element space is the space of compartment-wise continuous piecewise linear functions on tetrahedral elements in three dimensions (known as P_1). This space has a set of basis functions whose number is exactly the number of finite element nodes (including double nodes), and that are defined on the entire domain Ω:

$$\varphi_k : \Omega \to [0, 1], \quad k = 1, \ldots, N_{\text{node}}.$$

The basis function φ_k, $k \in \mathcal{I}_i$, is a piece-wise linear function, non-zero on the tetrahedra of Ω_i that touch the node $\boldsymbol{q}_k$, and zero on all other tetrahedra (including tetrahedra of other compartments than Ω_i that do touch $\boldsymbol{q}_k$). At the interface between two compartments Γ_{ij}, the value of φ_k is set to be the value it has inside its own compartment, and not outside. On a tetrahedron of Ω_i that touches $\boldsymbol{q}_k$, φ_k is equal to 1 on $\boldsymbol{q}_k$ and it is equal to 0 on the other 3 vertices of the tetrahedron. This completely describes the piece-wise linear function. The index sets may then be defined as $\mathcal{I}_i = \{k = 1, \ldots, N_{\text{node}} \mid \text{supp}(\varphi_k) \subset \Omega_i\}$; the set of indices of the finite element nodal functions whose supports lie entirely within Ω_i.

Any function in the finite element space can be written as a linear combination of the above basis functions:

$$\sum_{k=1}^{N_{\text{node}}} \alpha_k \varphi_k(\boldsymbol{x}).$$

To discretize the Bloch-Torrey and Laplace operators, we construct the following finite element matrices: $\mathbf{M}, \mathbf{S}, \mathbf{Q} \in \mathbb{R}^{N_{\text{node}} \times N_{\text{node}}}$, known in the FEM literature as the mass, stiffness, and flux matrices, respectively. These matrices matrices we need are defined as follows, for $(k, l) \in \{1, \ldots, N_{\text{node}}\}$:

$$M_{kl} = \int_\Omega \varphi_k(\boldsymbol{x})\varphi_l(\boldsymbol{x}) \, \mathrm{d}\Omega(\boldsymbol{x}), \tag{47}$$

$$S_{kl} = \int_\Omega \mathbf{D}(\boldsymbol{x})\nabla\varphi_k(\boldsymbol{x}) \cdot \nabla\varphi_l(\boldsymbol{x}) \, \mathrm{d}\Omega(\boldsymbol{x}), \tag{48}$$

$$Q_{kl} = \sum_{i=1}^{N_{\text{cmpt}}} \left(Q_{kl}^i + \sum_{j=1}^{N_{\text{cmpt}}} Q_{kl}^{ij} \right), \tag{49}$$

the latter being defined as the sum of interface integrals:

$$Q_{kl}^i = \kappa_i \int_{\Omega_i \cap \partial\Omega} \varphi_k(\boldsymbol{x})\varphi_l(\boldsymbol{x}) \, \mathrm{d}\Gamma(\boldsymbol{x}), \tag{50}$$

$$Q_{kl}^{ij} = \begin{cases} \kappa_{ij} c_{ji} \int_{\Gamma_{ij}} \varphi_k(\boldsymbol{x})\varphi_l(\boldsymbol{x}) \, \mathrm{d}\Gamma(\boldsymbol{x}), & (k, l) \in \mathcal{I}_i^2, \\ -\kappa_{ij} c_{ij} \int_{\Gamma_{ij}} \varphi_k(\boldsymbol{x})\varphi_l(\boldsymbol{x}) \, \mathrm{d}\Gamma(\boldsymbol{x}), & (k, l) \in \mathcal{I}_i \times \mathcal{I}_j, \\ 0, & \text{otherwise.} \end{cases} \tag{51}$$

We remind the reader that i and j are compartment indices, while k and l are finite element nodal indices. Note that the above sum formulation counts each interface twice, but only considers the contribution to one side of the boundary at the time. It also correctly accounts for corner nodes (if any) that belong to two different permeable boundaries at the same time.

The weak form of the Bloch-Torrey PDE also requires computing the first order moments of the product

of pairs of finite element basis functions. We let these three matrices be denoted by $\mathbf{J}^x$, $\mathbf{J}^y$ and $\mathbf{J}^z$:

$$J^x_{kl} = \int_\Omega x\, \varphi_k(\boldsymbol{x})\varphi_l(\boldsymbol{x})\, \mathrm{d}\Omega(\boldsymbol{x}), \tag{52}$$

$$J^y_{kl} = \int_\Omega y\, \varphi_k(\boldsymbol{x})\varphi_l(\boldsymbol{x})\, \mathrm{d}\Omega(\boldsymbol{x}), \tag{53}$$

$$J^z_{kl} = \int_\Omega z\, \varphi_k(\boldsymbol{x})\varphi_l(\boldsymbol{x})\, \mathrm{d}\Omega(\boldsymbol{x}), \tag{54}$$

where $\boldsymbol{x} = (x, y, z)^\mathsf{T}$. For a given gradient vector $\boldsymbol{g} = (g_x, g_y, g_z)^\mathsf{T}$, we define

$$\mathbf{J}(\boldsymbol{g}) = g_x\mathbf{J}^x + g_y\mathbf{J}^y + g_z\mathbf{J}^z. \tag{55}$$

In addition, we define the finite element relaxation matrix $\mathbf{R} \in \mathbb{R}^{N_\mathrm{node}\times N_\mathrm{node}}$ given by

$$R_{kl} = \int_\Omega \frac{1}{\tau(\boldsymbol{x})}\, \varphi_k(\boldsymbol{x})\varphi_l(\boldsymbol{x})\, \mathrm{d}\Omega(\boldsymbol{x}) \tag{56}$$

In SpinDoctor, these matrices are assembled from local element matrices and the assembly process is based on vectorized routines of [3], which replace expensive loops over elements by operations with 3-dimensional arrays. All local element matrices in the assembly of $\mathbf{S}$, $\mathbf{M}$, and $\mathbf{Q}$ are evaluated at the same time and stored in a full matrix of size $4 \times 4 \times N_\mathrm{element}$, where N_element denotes the number of tetrahedral elements.

The matrices $\mathbf{J}^x$, $\mathbf{J}^y$, and $\mathbf{J}^z$ are assembled as coordinate weighted mass matrices, using the same assembly procedure as for $\mathbf{M}$. The three coordinate functions $\boldsymbol{x} \mapsto x$, y, z are replaced by constant (P_0) finite element functions on each tetrahedron. The weights given to the assembly assembly function are given by $\boldsymbol{q}^x$, $\boldsymbol{q}^y$, $\boldsymbol{q}^z$; the vectors of x, y, and z coordinates of the finite element nodes. The matrix $\mathbf{J}(\boldsymbol{g})$ can thus quickly be obtained for different gradient vectors $\boldsymbol{g}$, without having to reassemble the entire matrix.

With compartment-wise constant T_2-relaxation times, $\mathbf{R}$ is a block-diagonally scaled version of the mass matrix $\mathbf{M}$, where the weights are the inverses of the relaxation times.

3.7.1 Finite element solution to the Bloch-Torrey PDE

The solution to the Bloch-Torrey partial differential equation (1) may be projected onto the finite element nodal basis $(\varphi_k)_{1\leq k\leq N_\mathrm{node}}$, in which case it is given by

$$M(\boldsymbol{x}, t) = \sum_{k=1}^{N_\mathrm{node}} \xi_k(t)\varphi_k(\boldsymbol{x}) = \boldsymbol{\xi}^\mathsf{T}(t)\boldsymbol{\varphi}(\boldsymbol{x}), \tag{57}$$

where $\boldsymbol{\varphi} = (\varphi_1, \ldots, \varphi_{N_\mathrm{node}})^\mathsf{T} \in \mathbb{R}^{N_\mathrm{node}}$ is the vector of finite element basis functions and $\boldsymbol{\xi} = (\xi_1, \ldots, \xi_{N_\mathrm{node}})^\mathsf{T} \in \mathbb{C}^{N_\mathrm{node}}$ is the solution to the following system of *ordinary* differential equations (ODE):

$$\mathbf{M}\frac{\partial \boldsymbol{\xi}}{\partial t} = -\left(\mathbf{S} + \mathbf{Q} + \mathbf{R} + \mathrm{i}\gamma f(t)\mathbf{J}\right)\boldsymbol{\xi}(t). \tag{58}$$

We note that the notation $^\mathsf{T}$ in Eq. (57) only denotes the *transpose*, as opposed to the complex conjugate transpose *. The complex transverse water proton magnetization M and thus its coefficients $\boldsymbol{\xi}$ may contain complex values. This is not the case for the finite element basis functions $\boldsymbol{\varphi}$, which are real-valued.

3.7.2 Finite element solution to the HADC model

Similarly, the solution to the HADC model (16), ω, may be obtained by solving the equation

$$\mathbf{M}\frac{\partial \boldsymbol{\zeta}}{\partial t} = -\mathbf{S}\boldsymbol{\zeta}(t) + F(t)\mathbf{G}\mathbf{D}\boldsymbol{u}_{\boldsymbol{g}}, \tag{59}$$

where $\boldsymbol{\zeta} = (\zeta_1, \ldots, \zeta_{N_{\text{node}}})$ is the unknown vector of coefficients of the solution $\omega(\boldsymbol{x}, t) = \boldsymbol{\zeta}^\mathsf{T}(t)\boldsymbol{\varphi}(\boldsymbol{x})$, $\boldsymbol{\varphi} = (\varphi_1, \ldots, \varphi_{N_{\text{node}}})$ is the vector of finite element basis functions, and $\mathbf{G} \in \mathbb{R}^{N_{\text{node}} \times 3}$ is given by

$$\mathbf{G} = \sum_{k=1}^{N_{\text{node}}} \int_{\partial\Omega} \varphi_k(\boldsymbol{x})\boldsymbol{\varphi}(\boldsymbol{x})\boldsymbol{n}^\mathsf{T}(\boldsymbol{x}) \, \mathrm{d}\Gamma(\boldsymbol{x}), \tag{60}$$

where $\boldsymbol{n} = (n_x, n_y, n_z)^\mathsf{T}$ is the outwards unit normal. This three-column matrix represents the components of the boundary integral of the quantity $F(t)\boldsymbol{u}_{\boldsymbol{g}}^\mathsf{T}\mathbf{D}\boldsymbol{n}$, where the constant diffusivity and gradient sequence dependent part $F(t)\boldsymbol{u}_{\boldsymbol{g}}^\mathsf{T}\mathbf{D}$ has been factored out. They can thus be assembled independently of the gradient sequence, and be reused when solving for multiple sequences or directions. They are assembled using the same routine as for $\mathbf{Q}$.

The boundary integral h from Eq. (15) is then given by

$$h(t) = \frac{1}{|\Omega|}\boldsymbol{\zeta}^\mathsf{T}(t)\mathbf{G}\boldsymbol{u}_{\boldsymbol{g}}, \tag{61}$$

where $|\Omega| = \sum_{j,k=1}^{N_{\text{node}}} M_{jk}$ is computed using the mass matrix.

3.7.3 Finite element Matrix Formalism signal representation

Both of the above ODEs are of dimension N_{node}, the number of finite element nodes. The finite element discretized Matrix Formalism representation uses a different approach, limiting the problem size to N_{eig}, the number of Laplace eigenfunctions (of which the choice is further explored in section 3.7.4). But first we need to solve an eigenvalue problem involving matrices of size $N_{\text{node}} \times N_{\text{node}}$. The finite element discretization changes the continuous Laplace operator eigenvalue problem (20) into the following discrete, generalized *matrix* eigenvalue problem: find $(\lambda, \boldsymbol{p}) \in \mathbb{R} \times \mathbb{R}^{N_{\text{node}}}$ such that

$$\lambda\mathbf{M}\boldsymbol{p} = (\mathbf{S} + \mathbf{Q})\boldsymbol{p}, \tag{62}$$

of which we will keep the $N_{\text{eig}} \leq N_{\text{node}}$ smallest eigenvalues λ_n and corresponding eigenvectors $\boldsymbol{p}_n$, $n = 1, \ldots, N_{\text{eig}}$. Note however that there are in total N_{node} solutions to the problem (62). Moving back to the space of functions (the function space P_1), the eigenfunction $\phi_n(\boldsymbol{x})$ associated to the eigenvalue λ_n is then

$$\phi_n(\boldsymbol{x}) = \sum_{k=1}^{N_{\text{node}}} p_n^k \varphi_k(\boldsymbol{x}), \quad 1 \leq n \leq N_{\text{eig}},$$

where the entries of the eigenvector $\boldsymbol{p}_n$ are the coefficients of the eigenfunction ϕ_n in the finite element basis. Using matrix notation, this conversion can also be written

$$\boldsymbol{\phi} = \mathbf{P}^\mathsf{T}\boldsymbol{\varphi}, \tag{63}$$

where $\boldsymbol{\varphi} = (\varphi_1, \ldots, \varphi_{N_{\text{node}}})^\mathsf{T}$, $\boldsymbol{\phi} = (\phi_1, \ldots, \phi_{N_{\text{eig}}})^\mathsf{T}$, and $\mathbf{P} = (\boldsymbol{p}_1, \ldots, \boldsymbol{p}_{N_{\text{eig}}}) \in \mathbb{R}^{N_{\text{node}} \times N_{\text{eig}}}$.

Then it is clear that the first order moments of the product of pairs of Laplace eigenfunctions can be written as:

$$\begin{aligned}
A^x_{mn} &= \sum_{k=1}^{N_{\text{node}}} \sum_{l=1}^{N_{\text{node}}} p_m^k p_n^l \left(\int_\Omega x \, \varphi_k(\boldsymbol{x})\varphi_l(\boldsymbol{x}) \, \mathrm{d}\Omega(\boldsymbol{x}) \right) = \boldsymbol{p}_m^\mathsf{T}\mathbf{J}^x\boldsymbol{p}_n, \\
A^y_{mn} &= \sum_{k=1}^{N_{\text{node}}} \sum_{l=1}^{N_{\text{node}}} p_m^k p_n^l \left(\int_\Omega y \, \varphi_k(\boldsymbol{x})\varphi_l(\boldsymbol{x}) \, \mathrm{d}\Omega(\boldsymbol{x}) \right) = \boldsymbol{p}_m^\mathsf{T}\mathbf{J}^y\boldsymbol{p}_n, \quad (m,n) \in \{1, \ldots, N_{\text{eig}}\}^2. \\
A^z_{mn} &= \sum_{k=1}^{N_{\text{node}}} \sum_{l=1}^{N_{\text{node}}} p_m^k p_n^l \left(\int_\Omega z \, \varphi_k(\boldsymbol{x})\varphi_l(\boldsymbol{x}) \, \mathrm{d}\Omega(\boldsymbol{x}) \right) = \boldsymbol{p}_m^\mathsf{T}\mathbf{J}^z\boldsymbol{p}_n,
\end{aligned} \tag{64}$$

Using matrix notation, we have

$$\mathbf{A}^u = \mathbf{P}^\mathsf{T}\mathbf{J}^u\mathbf{P}, \quad u = x, y, z. \tag{65}$$

Parameter	Variables	Unit
Length	$\boldsymbol{x}$, $\boldsymbol{q}$, ℓ	μm
Time	t, T_{echo}, δ, Δ, T_2	μs
Diffusivity	$\mathbf{D}$, D_0, $\bar{\sigma}$, ADC, STA	$\mu m^2/\mu s = \text{mm}^2/\text{s}$
Permeability	κ	$\mu m/\mu s = \text{m/s}$
Magnetic field gradient	$\boldsymbol{g}$	T/m
Q-value	$q = \gamma\|\boldsymbol{g}\|$	$(\mu s \mu m)^{-1}$
B-value	b	$\mu s/\mu m^2 = \text{s/mm}^2$
Magnetization	M, ρ	
Signal	S	

Table 1: Physical units of common quantities in SpinDoctor.

These three matrices are computed using a total of six matrix-matrix multiplications. Similarly, the eigenfunction basis relaxation matrix $\mathbf{T}$ may be obtained from the finite element nodal basis relaxation matrix $\mathbf{R}$ by

$$\mathbf{T} = \mathbf{P}^{\mathsf{T}}\mathbf{R}\mathbf{P}. \tag{66}$$

The generalized eigenvalue problem in (62) is solved using the built-in MATLAB command `eigs`, which can exploit the symmetry of $\mathbf{M}$ and compute a subset of all the eigenvalues, meaning N_{eig} can be much smaller than N_{node}. In earlier versions of SpinDoctor, we depended on the Partial Differential Equation Toolbox of MATLAB to solve the generalized eigenvalue problem, but this is no longer needed in our current implementation.

3.7.4 Choice of eigen-decomposition parameters

We do not want to use the entire set of eigenvalues and eigenvectors $\{(\lambda_n, \boldsymbol{p}_n)\}_{1 \leq n \leq N_{\text{node}}}$ of the matrix eigenvalue problem in (62), because the size of $\mathbf{M}$, $\mathbf{S}$, and $\mathbf{Q}$ is determined by the finite element discretization (it is equal to N_{node}, the number of finite element nodes). This means that most of the rapidly oscillating eigenmodes in the matrix eigenvalue problem are linked to the finite element discretization, and not to the physics of the problem. To link with the physics of the diffusion in the cell geometry, we set a restricted interval in which to keep the computed eigenvalues. We set the interval to be $[0, (\pi/\ell_{\min})^2\bar{\sigma}]$, where $\ell_{\min}$ is the shortest length scale of interest in the cell geometry. In this way, the number of computed eigenmodes, N_{eig}, will be much smaller than N_{node}.

The MATLAB command `eigs` can identify the N_{eig} smallest eigenvalues of the problem in (62). If we choose this number to be large enough such that $\ell(\lambda_{N_{\text{eig}}}) \leq \ell_{\min}$, we can be sure to have found all the modes of interest. Ideally, we thus take

$$N_{\text{eig}} = \min\{n \in \mathbb{N} \mid \ell(\lambda_n) \leq \ell_{\min}\}.$$

In order to choose the number N_{eig}, we can either make a conservative estimate (of the order of N_{node}), and then remove the largest eigenvalues, or we can start out with a smaller first guess for N_{eig} and increment it if the smallest length scale obtained is too large. In particular, having identified the correct number of eigenmodes for a given set of parameters, we can expect this number to be of the same order of magnitude if we change the some of the parameters (e.g. permeability κ), and scale linearly with the diffusivity $\mathbf{D}$.

If $N_{\text{eig}} \geq N_{\text{node}}/2$, the `eig` command is called instead, and a full decomposition is performed. In both cases, we only retain the eigenvalues whose length scales are larger than $\ell_{\min}$ to compute the Matrix Formalism signal.

3.8 Physical units

The physical units of the input and output quantities for SpinDoctor are shown in Table 1.

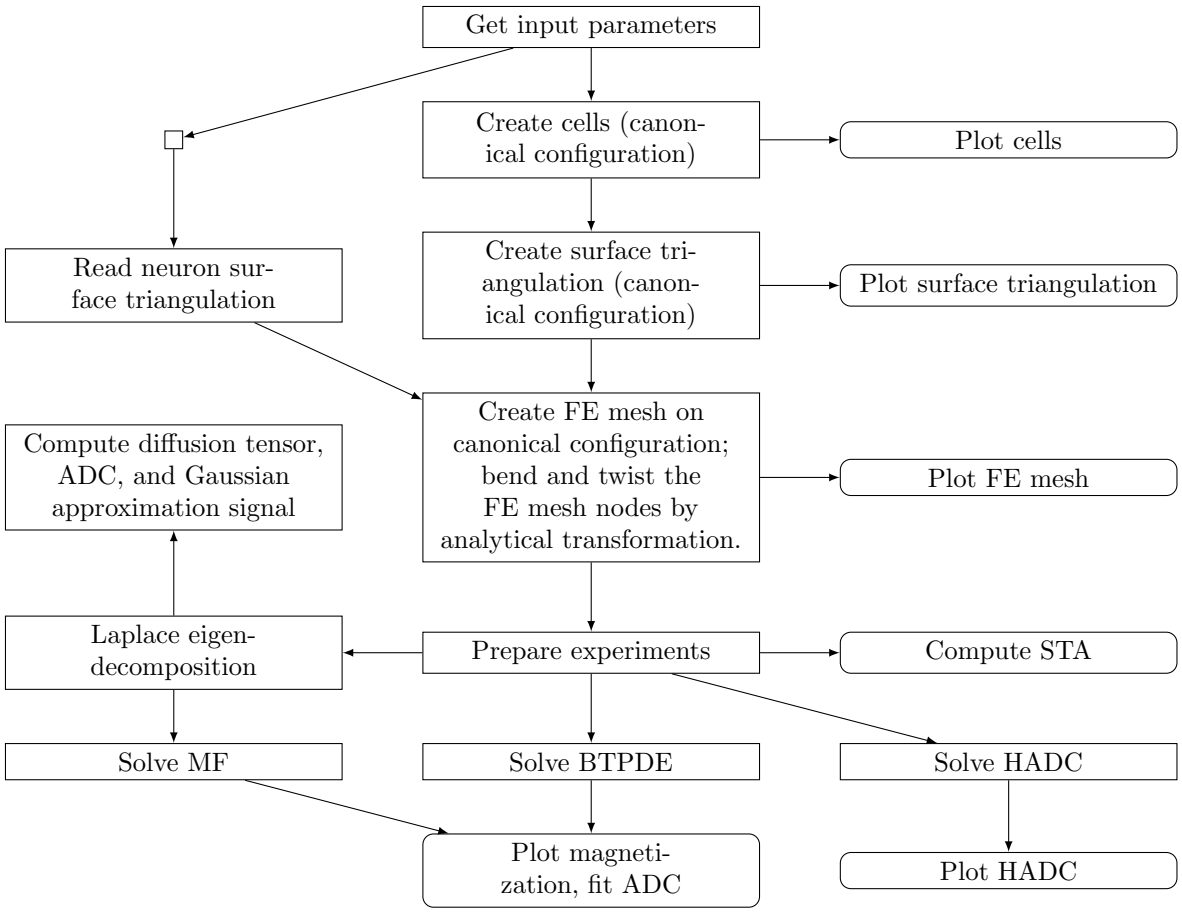

Figure 1: Flow chart describing the work flow of SpinDoctor

4 Software components and algorithms

In this section we discuss the various components of SpinDoctor in more detail.

4.1 Work flow

Figure 1 is a chart showing the work flow of SpinDoctor. A session starts with creating or loading the geometry. Plotting functions allow for inspecting the resulting geometry. At the end of this process, a finite element mesh is returned, where the nodes are sorted into their respective compartments.

The user also has to prepare the experiments. The function `prepare_pde` creates a compartment numbering system and assigns the material proper to the compartments. The function `prepare_experiments` checks the consistency of the experiments, and prepares some important quantities related to the gradient sequences (q-values, b-values, directions).

SpinDoctor provides different solvers, some of which only work under certain assumptions. The function `solve_btpde` computes the magnetization and signal at a precision only limited by the mesh refinement and ODE-solver tolerances. The ADC may be fitted from the resulting signal. The function `solve_mf` produces the same result as `solve_btpde`, but at a much lower computational cost. This requires having previously computed the Laplace eigenvalues and eigenfunctions, from which a diffusion tensor and Gaussian approximation of the signal may be computed. The functions `solve_hadc` computes the ADC using a homogenized model, that assumes negligible permeability between compartments.

4.2 User provided input parameters

SpinDoctor expects the user to define a setup structure `setup` with different substructures. In the folder `setups`, there are several example setups. Some important fields of `setup` are:

1. `name` (string) file name for saving or loading geometry and results
2. `geometry`: structure containing geometry parameters. Format is given in Table 2. If the cells are spheres or cylinders, the parameters for the geometrical configuration are provided in this structure.
3. `pde`: structure containing the PDE coefficients (material properties). Format is given in Table 3. This structure provides the material properties for each of the compartments `in`, `out`, `ecs`, as well as their interfaces. If the `in` or `ecs` compartments are not included, the corresponding parameters are ignored.
4. `gradient`: structure defining the gradient sequences. Format is given in Table 4. These gradients are parameterized by their amplitude, time profile (sequence), and direction.

The sequences (time profiles) are instances of the MATLAB class `Sequence`, which can be of type `PGSE`, `DoublePGSE`, `CosOGSE`, `SinOGSE`, or `CustomSequence`. The latter requires providing a function handle representing the time profile. All the sequences can be called at time t through the `call` method, whose integrals and b-values are obtained by the `integral` and `bvalue_no_q` methods. `echotime` provides the echo time T_{echo} of the sequence. Each sequence comes with utility functions for string representation and information about their characteristic intervals. By default, a `Sequence` only requires a `call` method, while the other quantities can be computed numerically. However, for specific sequences, analytical expressions may be provided for the integral quantities.

In addition, the following possible substructures of `setup` defines the experiments to be carried out. Their precense triggers the corresponding experiment. They can be found in Table 5:

- `btpde` Solve BTPDE using P1 finite elements and built-in MATLAB ODE solvers. More details can be found in section 4.8.
- `hadc` Solve HADC using P1 finite elements and built-in MATLAB ODE solvers. More details can be found in section 4.9.
- `mf` Solve for the matrix formalism magnetization, given a P1-discretized Laplace eigendecomposition. More details can be found in section 4.10.
- `analytical` Compute signal from analytical expressions for one multilayered cylinder or sphere. More details can be found in section 4.11.
- `karger` Solve FPK model.

4.3 Important output quantities

In Table 6 we list some useful quantities that are the outputs of SpinDoctor. The braces in the "Size" column denote MATLAB cell array structures and the brackets denote MATLAB matrix data structures.

4.4 Important functions

In Table 7 we list some important functions of SpinDoctor. For detailed information about them, including argument lists, please read the online documentation.

In Table 8 we list important functions of the Matrix Formalism Module. For detailed information about them, including argument lists, please read the online documentation.

Field name	Example	Explanation
`cell_shape`	`"cylinder"`	`"sphere"`, `"cylinder"` or `"neuron"`
`ncell`	`10`	Number of cells (1 for neurons)
`rmin`	`1.5`	Minimum cell radius
`rmax`	`2.5`	Maximum cell radius
`dmin`	`1.5`	Minimum (%) distance between cells $\frac{d_{\min}(r_{\min}+r_{\max})}{2}$
`dmax`	`2.5`	Maximum (%) distance between cells $\frac{d_{\max}(r_{\min}+r_{\max})}{2}$
`height`	`20`	Cylinder height (ignored if not cylinder)
`deformation`	`[0.05 0.05]`	$[\alpha \quad \beta]$; α defines the amount of bend; β defines the amount of twist (in radians)
`include_in`	`true`	Indicator for including in-compartment
`in_ratio`	`0.6`	$\frac{r_{\text{in}}}{r_{\text{out}}} \in]0, 1[$
`ecs_shape`	`"no_ecs"`	Do not include ECS: `"no_ecs"`; Box wrap ECS: `"box"`; Convex hull: `"convex_hull"` Tight wrap: `"tight_wrap"`;
`ecs_ratio`	`0.3`	ECS thickness as percentage of mean radius
`refinement`	`0.5`	Requested TetGen maximum tetrahedron volume (comment line to use default)

Table 2: Structure containing geometry parameters.

Field name	Example	Explanation
`initial_density_in`	`1`	Initial spin density ρ
`initial_density_out`	`1`	
`initial_density_ecs`	`1`	
`diffusivity_in`	`0.002`	Diffusion coefficient σ. Can also be a 3x3-tensor
`diffusivity_out`	`0.002`	
`diffusivity_ecs`	`0.002`	
`relaxation_in`	`Inf`	T_2-relaxation times (for no relaxation: Inf)
`relaxation_out`	`5000`	
`relaxation_ecs`	`5000`	
`permeability_in_out`	`1e-3`	Permeability coefficient κ
`permeability_out_ecs`	`1e-4`	
`permeability_in`	`0`	
`permeability_out`	`0`	
`permeability_ecs`	`0`	

Table 3: Structure containing PDE parameters.

Field name	Example	Explanation
`ndirection`	`20`	Number of gradient directions
`flat_dirs`	`false`	Uniformly distribute directions in 3D or 2D (if more than one direction)
`remove_opposite`	`false`	Only solve for half of the directions, copy results to opposite directions
`direction`	`[1.0; 0.0; 0.0]`	Gradient direction (if $n_{\mathrm{dir}} = 1$), no need to normalize
`values`	`[10 100 400 1000]`	$\|\boldsymbol{g}\|$-values, q-values or b-values to consider
`values_type`	`"b"`	`"g"`, `"q"` or `"b"`
`sequences`	`{PGSE(100, 5000)}`	Gradient sequences; `PGSE(delta, Delta)`, `DoublePGSE(delta, Delta)`, `CosOGSE(delta, Delta, nperiod)`, `SinOGSE(delta, Delta, nperiod)` or `CustomSequence(delta, Delta, @timeprofile)`

Table 4: Structure defining gradient sequences.

Field name	Example	Explanation
`btpde`	`struct`	Substructure for solving the BTPDE (comment out to skip)
`btpde.ode_solver`	`@ode23t`	Function handle for ODE solver (comment to use default)
`btpde.reltol`	`1e-4`	Relative tolerance for ODE solver
`btpde.abstol`	`1e-6`	Absolute tolerance for ODE solver
`hadc`	`struct`	Substructure for solving the HADC model (comment out to skip)
`hadc.ode_solver`	`@ode15s`	Function handle for ODE solver (comment to use default)
`hadc.reltol`	`1e-4`	Relative tolerance for ODE solver
`hadc.abstol`	`1e-4`	Absolute tolerance for ODE solver
`mf`	`struct`	Substructure for solving for the Matrix Formalism (comment out to skip)
`mf.length_scale`	`3`	Minimum length scale of Laplace eigenfunctions
`mf.neig_max`	`100`	Maximum number of eigenvalues. To compute all: `Inf`
`mf.ninterval`	`50`	Number of intervals to discretize time profile (if not PGSE)
`analytical`	`struct`	Substructure for solving for the analytical model (comment out to skip)
`analytical.length_scale`	`3`	Minimum length scale of radial Laplace eigenfunctions
`analytical.eigstep`	`1e-6`	Minimium distance between two radial eigenvalues
`karger`	`struct`	Substructure for solving for the FPK model (comment out to skip)
`karger.ndirection`	`50`	Number of direction to compute effective diffusion tensor
`karger.ode_solver`	`@ode43s`	Function handle for ODE solver (comment to use default)
`karger.reltol`	`1e-4`	Relative tolerance for ODE solver
`karger.abstol`	`1e-6`	Absolute tolerance for ODE solver

Table 5: Structures containing simulation experiment parameters.

Variable name	Size	Explanation
`magnetization`	{ncmpt × namp × nseq × ndir}[nnode]	Solution of the BTPDE for each compartment
`signal`	[ncmpt × namp × nseq × ndir]	Signal (integral of magnetization) at T_{echo} in each compartment.
`signal_allcmpts`	[namp × nseq × ndir]	Signal (integral of magnetization) at T_{echo} summed over all compartments.
`adc`	[ncmpt × nseq × ndir]	ADC in each compartment.
`adc_allcmpts`	[nseq × ndir]	ADC accounting for all compartments.
`lap_eig.values`	[neig × 1]	Laplace eigenvalues
`lap_eig.funcs`	[nnode × neig]	Laplace eigenfunctions
`lap_eig.moments`	[neig × neig × 3]	First order moments of the product pairs of Laplace eigenfunctions
`diffusion_tensor`	[3 × 3 × nseq]	Effective diffusion tensors

Table 6: Some important SpinDoctor output quantities.

Function name	Purpose
`prepare_experiments`	Prepare magnetic gradient sequences, directions, and BTPDE, HADC and MF experiments
`prepare_pde`	Set up the PDE model in the geometrical compartments
`create_geometry`	Create or load cells, surface geometry and finite element mesh
`solve_btpde`	Compute the BTPDE magnetization and signal
`solve_hadc`	Compute the ADC from the HADC model
`fit_signal`	Fit the ADC and signal for zero b-weighting from the BTPDE signal.
`compute_adc_sta`	Compute the short time approximation in one diffusion-encoding direction.
`compute_free_diffusion`	Compute the free diffusion ADC and signal.
`plot_cells`	
`plot_surface_triangulation`	
`plot_femesh`	Display the finite elements mesh.
`plot_geometry_info`	
`plot_signal`	Display the simulated signal in one diffusion-encoding direction.
`plot_timing`	
`plot_adc`	Display the simulated ADC in one diffusion-encoding direction.
`plot_field`	Display the PDE solution on the finite elements mesh. If the finite elements mesh is too large, the magnetization should not be outputted from the solution of the BTPDE, and this function cannot be used.
`plot_hardi`	Display simulation results in multiple diffusion-encoding directions.

Table 7: Some important functions in SpinDoctor.

Function name	Purpose
`compute_laplace_eig`	Compute the Laplace eigenvalues, eigenfunctions and the first order moments of the products of the eigenfunctions.
`eig2length`, `length2eig`	Convert the computed eigenvalues into a length scale, and vice-versa
`mf_jn`	Compute the quantity $J(\lambda_n, f)$
`compute_mf_diffusion_tensor`	Computes the effective diffusion tensor
`plot_diffusion_tensor`	Display the effective diffusion tensor
`solve_mf`	Compute the Matrix Formalism magnetization and signal
`compute_mf_signal`	Compute the Matrix Formalism total signal only
`compute_mfga_signal`	Computes the Matrix Formalism Gaussian Approximation total signal

Table 8: Some important functions in the Matrix Formalism Module.

4.5 Create cells (canonical configuration)

SpinDoctor supports the placement of a group of non-overlapping cells in close vicinity to each other. There are two proposed configurations, one composed of spheres, the other composed of cylinders. The algorithm is described in Algorithm 1.

Algorithm 1: Placing N_{cell} non-overlapping cells.

Generate a large number of possible cell centers.
Compute the minimum distance, d, between the current center and previously accepted cells.
Find the intersection of $[d - d_{\max} r_{\text{mean}},\ d - d_{\min} r_{\text{mean}}]$ and $[r_{\min}, r_{\max}]$, where $r_{\text{mean}} = \frac{r_{\min} + r_{\max}}{2}$. If the intersection is not empty, then take the middle of the intersection as the new radius and accept the new center. Otherwise, reject the center.
Loop through the possible centers until get N_{cell} accepted cells.

SpinDoctor provides a routine to plot the cells to see if the configuration is acceptable (see Fig. 2).

4.6 Create surface triangulation

Finite element mesh generation software requires a good surface triangulation. This means the surface triangulation needs to be water-tight and does not self-intersect. How closely these requirements are met in floating point arithmetic has a direct impact on the quality of the finite element mesh generated.

It is often difficult to produce a good surface triangulation for arbitrary geometries. Thus, we restrict the allowed shapes to cylinders and spheres. Below in Algorithms 2 and 3 we describe how to obtain a surface triangulation for spherical cells with nucleus, cylindrical cells with myelin layer, and the ECS (box or tightly wrapped). We describe a canonical configuration where the cylinders are placed parallel to the z-axis. More general shapes are obtained from the canonical configuration by coordinate transformation in a later step.

SpinDoctor provides a routine to plot the surface triangulation (see Fig. 3).

4.7 Finite element mesh generation

SpinDoctor calls Tetgen [2], an external package (executable files are included in the toolbox package), to create a tetrahedra finite elements mesh from the surface triangulation generated by Algorithms 2 and 3. The FE mesh is generated on the canonical configuration. The numbering of the compartments and boundaries used by SpinDoctor are given in Tables 9 and 10. The labels are related to the values of the intrinsic diffusion coefficient, the initial spin density, and the permeability requested by the user. Then the FE mesh nodes are deformed analytically by a coordinate transformation, described in Algorithm 4.

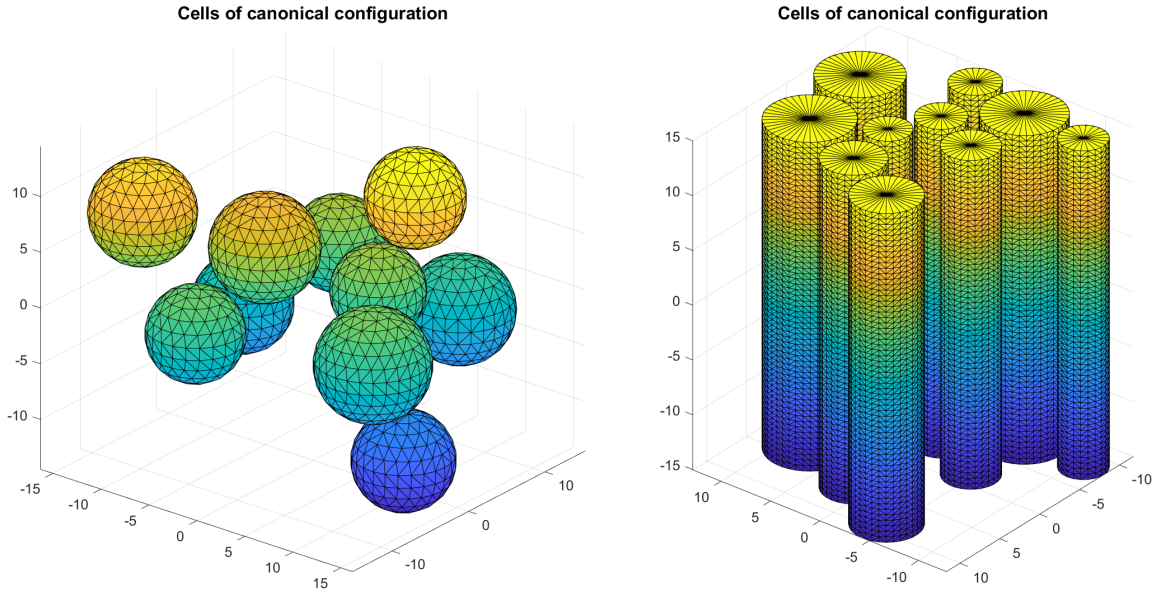

Figure 2: SpinDoctor plots cells in the canonical configuration.

Algorithm 2: Surface triangulation of spherical cells and ECS.

Suppose we have N_{cell} spherical cells with nucleus. Denote a sphere with center c and radius R by $S(c, R)$, we use the built-in functions (convex hull, delaunay triangulation) in MATLAB to get its surface triangulation, $T(c, R)$. Call the radii of the nucleus $r_1, \ldots, r_{N_{\text{cell}}}$ and the radii of the cells $R_1, \ldots, R_{N_{\text{cell}}}$. Then the boundaries between the cytoplasm and the nucleus are

$$\{\Gamma_i = T(c_i, r_i)\}, \quad i = 1, \ldots, N_{\text{cell}};$$

and between the cytoplasm and the ECS

$$\{\Sigma_i = T(c_i, R_i)\}, \quad i = 1, \ldots, N_{\text{cell}};$$

For the box ECS, we find the coordinate limits of the set

$$\bigcup_{i=1}^{N_{\text{cell}}} S(c_i, R_i) \subset [x_0, x_f] \times [y_0, y_f] \times [z_0, z_f]$$

and add a gap $k = \text{ecs\_gap} \times \max\{x_f - x_0, y_f - y_0, z_f - z_0\}$ to make a box

$$B = [x_0 - k, x_f + k] \times [y_0 - k, y_f + k] \times [z_0 - k, z_f + k].$$

We put 2 triangles on each face of B to make a surface triangulation Ψ with 12 triangles. For the tight-wrap ECS, we increase the cell radius by a gap size and take the union

$$W = \bigcup_{i=1}^{N_{\text{cell}}} S(c_i, R_i + \text{ecs\_gap} \times R_{\text{mean}}),$$

where $R_{\text{mean}} = \frac{R_{\min} + R_{\max}}{2}$. We use the alphaShape function in MATLAB to find a surface triangulation Ψ that contains W.

Algorithm 3: Surface triangulation of cylindrical cells and ECS.

Suppose we have N_{cell} cylindrical cells with a myelin layer, all with height H. Denote a disk with center c and radius R by $D(c, R)$, and the circle with the same center and radius by $C(c, R)$. Let the radii of the axons be $r_1, \ldots, r_{N_{\text{cell}}}$ and the radii of the cells be $R_1, \ldots, R_{N_{\text{cell}}}$, meaning the thickness of the myelin layer is $R_i - r_i$.

The boundary between the axon and the myelin layer is:

$$C(c_i, r_i) \times [-H/2, H/2]$$

We discretize $C(c_i, r_i)$ as a polygon $P(c_i, r_i)$ and place one at $z = -H/2$ and one at $z = H/2$. Then we connect the corresponding vertices of $P(c_i, r_i) \times \{-H/2\}$ and $P(c_i, r_i) \times \{H/2\}$ and add a diagonal on each panel to get a surface triangulation Γ_i.

Between the myelin layer and the ECS we discretize $C(c_i, R_i)$ as a polygon and place one at $z = -H/2$ and one at $z = H/2$ to get a surface triangulation Σ_i.

For the box ECS, we find the coordinate limits of the union of $D(c_i, r_i)$ and add a gap to make a rectangle in two dimensions. Then we place the rectangle at $z = -H/2$ and at $z = H/2$ to get a box. Finally, the box is given a surface triangulation with 12 triangles.

For tight-wrap ECS, we increase the cell radius by a gap size and take the union

$$W = \bigcup_{i=1}^{N_{\text{cell}}} D(c_i, R_i + kR_{\text{mean}}).$$

We use the alphaShape function in MATLAB to find a two dimensional polygon Q that contains W. We place Q at $z = -H/2$ and at $z = H/2$ and connect corresponding vertices, adding a diagonal on each panel. Suppose Q is a polygon with n vertices, then the surface triangulation of the side of the ECS will have $2n$ triangles.

The above procedure produces a surface triangulation for the boundaries that are parallel to z-axis. We now must close the top and bottom. The top and bottom boundaries is just the interior of Q. However, the surface triangulation cannot be done on Q directly. We must cut out $D(c_i, r_i)$, the disk which touches the axon, and $A_i = D(c_i, R_i) - D(c_i, r_i)$, the annulus which touches the myelin. Then we triangulate $Q - \bigcup_{i=1}^{N_{\text{cell}}} D(c_i, R_i)$ using the MATLAB built-in function that triangulates a polygon with holes to get the boundary that touches the ECS. The surface triangulation for A_i and $D(c_i, r_i)$ is straightforward.

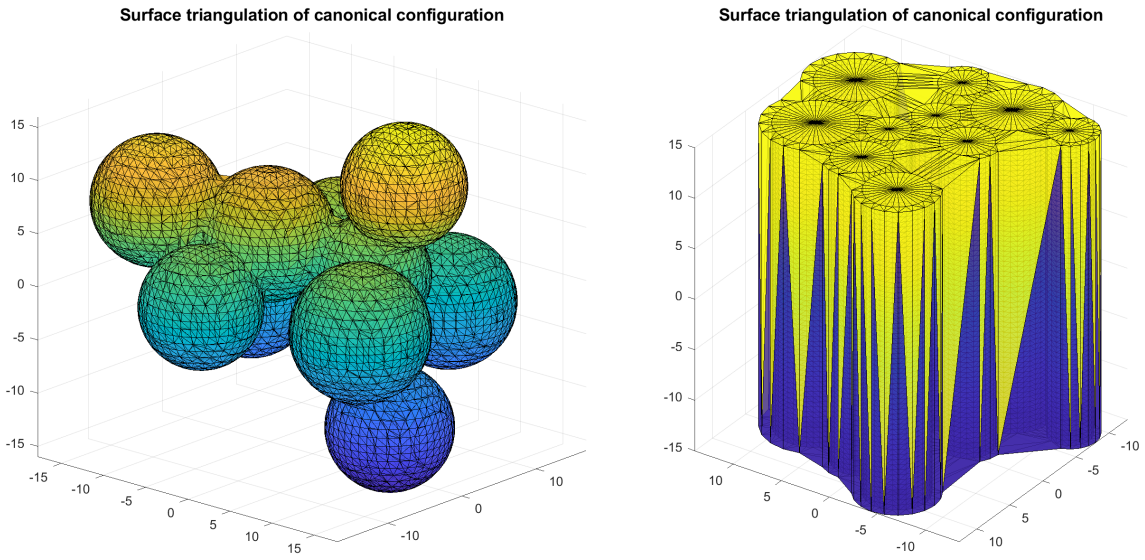

Figure 3: SpinDoctor plots the surface triangulation of the canonical configuration. Left: spherical cells with ECS; Right: cylindrical cells with ECS.

Spherical cells without nucleus

Cmpt		Nucleus	ECS
Label		out	ecs
Number		$[1, \ldots, n_{\text{cell}}]$	$n_{\text{cell}} + 1$

Spherical cells with nucleus

Cmpt	Nucleus	Cytoplasm	ECS
Label	in	out	ecs
Number	$[1, \ldots, n_{\text{cell}}]$	$[n_{\text{cell}} + 1, \ldots, 2n_{\text{cell}}]$	$2n_{\text{cell}} + 1$

Cylindrical cells without myelin

Cmpt		Axon	ECS
Label		out	ecs
Number		$[1, \ldots, n_{\text{cell}}]$	$n_{\text{cell}} + 1$

Cylindrical cells with myelin

Cmpt	Axon	Myelin	ECS
Label	in	out	ecs
Number	$[1, \ldots, n_{\text{cell}}]$	$[n_{\text{cell}} + 1, \ldots, 2n_{\text{cell}}]$	$2n_{\text{cell}} + 1$

Neuron

Cmpt		Neuron	ECS
Label		out	ecs
Number		1	2

Table 9: The labels and numbers of compartments.

Spherical cells without nucleus

Label		out,ecs			ecs
Number		$[1, \ldots, n_{\text{cell}}]$			$n_{\text{cell}} + 1$

Spherical cells with nucleus

Label	in,out	out,ecs			ecs
Number	$[1, \ldots, n_{\text{cell}}]$	$[n_{\text{cell}} + 1, \ldots, 2n_{\text{cell}}]$			$2n_{\text{cell}} + 1$

Cylindrical cells without myelin

Boundary		Cylinder side wall		Cylinder top and bottom	Outer ECS boundary
Label		out,ecs		out	ecs
Number		$[1, \ldots, n_{\text{cell}}]$		$[n_{\text{cell}} + 1, \ldots, 2n_{\text{cell}}]$	$2n_{\text{cell}} + 1$

Cylindrical cells with myelin

Boundary	Inner cylinder side wall	Outer cylinder side wall	Inner cylinder top and bottom	Outer cylinder top and bottom	Outer ECS boundary
Label	in,out	out,ecs	in	out	ecs
Number	$[1, \ldots, n_{\text{cell}}]$	$[n_{\text{cell}} + 1, \ldots, 2n_{\text{cell}}]$	$[2n_{\text{cell}} + 1, \ldots, 3n_{\text{cell}}]$	$[3n_{\text{cell}} + 1, \ldots, 4n_{\text{cell}}]$	$4n_{\text{cell}} + 1$

Neuron

Boundary				Neuron	
Label				out	
Number				1	

Neuron with ECS

Boundary		Neuron			ECS
Label		out,ecs			ecs
Number		1			2

Table 10: The labels and numbers of boundaries.

Algorithm 4: Bending and twisting of the FE mesh of the canonical configuration.

The external package Tetgen [2] generates the finite element mesh that keeps track of the different compartments and the interfaces between them. The mesh is saved in several text files.
The connectivity matrices of the finite elements and facets are not modified by the coordinates transformation described below. The nodes are transformed by bending and twisting as described next.
The set of FE mesh nodes $\{x_i, y_i, z_i\}$ are transformed in the following ways:
Twisting around the z-axis with a user-chosen twisting parameter α_{twist} is defined by

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \rightarrow \begin{bmatrix} \cos(\alpha_{\text{twist}} z) & -\sin(\alpha_{\text{twist}} z) & 0 \\ \sin(\alpha_{\text{twist}} z) & \cos(\alpha_{\text{twist}} z) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}.$$

Bending on the $x - z$ plane with a user-chosen bending parameter α_{bend} is defined by

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \rightarrow \begin{bmatrix} x + \alpha_{\text{bend}} z^2 \\ y \\ z \end{bmatrix}.$$

Given $[\alpha_{\text{bend}}, \alpha_{\text{twist}}]$, bending is performed after twisting.

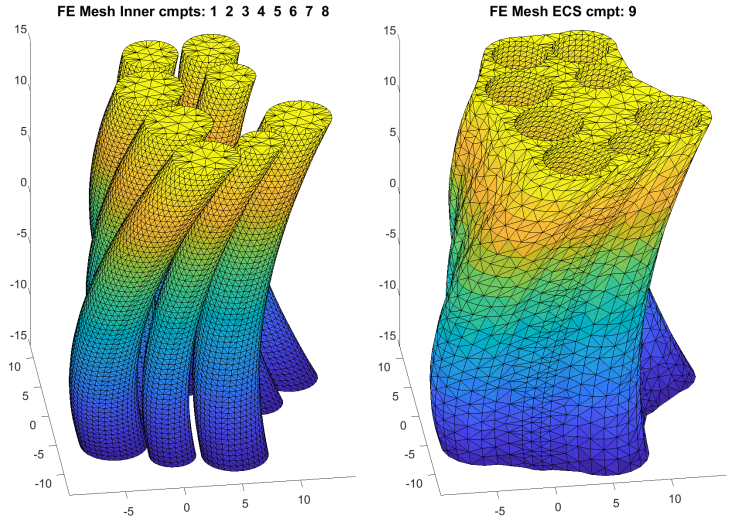

Figure 4: FE mesh of cylinders and ECS after bending and twisting. Compartment number is 1 to 8 for the cylinders and 9 for the ECS.

SpinDoctor provides a routine to plot the FE mesh (see Fig. 4 for cylinders and ECS that have been bent and twisted).

4.8 Solve BTPDE

The BTPDE-solver `solve_btpde` is triggered by the presence of the field `setup.btpde`. The solver first assembles all the finite element matrices, before it loops over gradient amplitudes $\|\boldsymbol{g}\|$ (given by `setup.values` of type `"g"`, `"q"` or `"b"`), pulse sequences f (given by `setup.sequences`), and directions $\boldsymbol{u_g}$ (given by `setup.gradient.directions`). The loop is traversed using linear indices, meaning that the three loops are combined into one. If the Parallel Computing Toolbox is available, the loop is parallelized.

For each iteration, the ODE-solver is set up with the appropriate functions and parameters. The ODE is then solved using one of the built-in ODE routines in MATLAB (provided in `setup.btpde.ode_solver`), the default being `ode15s`. Only the final magnetization is stored – this assures limited memory usage if multiple solver calls are running in parallel. The signal is computed from the final magnetization.

In addition, the time interval $[0, T_{\text{echo}}]$ is split into multiple subintervals if the sequence f is of type

`PGSE`, `DoublePGSE`, `CosOGSE`, or `SinOGSE`. This is done to take advantage of the sequence beeing constant on certain intervals, resulting in a simplified ODE-function and a matrix Jacobian (instead of a time dependent Jacobian). The magnetization at the end of one interval is used as initial conditions for the beginning of the next interval. The computational advantage of this scheme is only visible for iterations that would normally take a long time, especially for higher b-values (higher gradient amplitudes or longer pulses). For lower b-values, this procedure may be slower, when the setup of the ODE-solver is larger compared to the actual time-stepping than for higher b-values.

The user may toggle the switch `setup.gradient.remove_opposite`, which skips solving for directions opposite to those already computed. This is based on the symmetry of the BTPDE around the real axis – if the initial spin density is real, flipping the gradient direction only flips the sign of the imaginary part of the complex magnetization. The complex conjugate is thus returned for these opposite directions.

4.9 Solve HADC model

Similarly, the diffusion equation of the HADC model is discretized using P_1 finite elements. The solver is controlled by the field `setup.hadc`, of which the precense triggers the solver `solve_hadc`. The ODE-solver is given by `setup.hadc.ode_solver` (default: `ode15s`), with tolerances `setup.hadc.abstol` and `setup.hadc.reltol`.

The solver assembles the finite element matrices $\mathbf{M}$, $\mathbf{S}$, and $\mathbf{G}$, before it loops over compartments, sequences and directions using linear indices, possible using parallel iterations if the Parallel Computing Toolbox is available.

The HADC model differs from the BTDPE in that the compartments are assumed to be separated, and we therefore solve for each compartment separately. The HADC does not depend on the gradient amplitudes $\|\boldsymbol{g}\|$. In addition, the HADC solver has to store all the time steps from the ODE-solver, as time integrals are required.

Each iteration consists of setting up the ODE for the diffusion equation for the given sequence f (with integral F) and direction $\boldsymbol{u_g}$. The solution to the diffusion equation $\boldsymbol{\zeta}(t)$ is computed and stored for all time steps. The integral quantity $\int_0^{T_{\text{echo}}} F(t)h(t)\,\mathrm{d}t$ is then computed numerically, using a trapezoidal rule with the time steps $t_1, \ldots, t_{N_{\text{step}}}$. The denominator in Eq. (14) is available through the `bvalue_no_q` method of the sequence.

4.10 Solve matrix formalism model

The function `compute_laplace_eig` computes the Laplace eigenvalues $(\lambda_n)_{n=1,\ldots,N_{\text{eig}}}$, eigenfunction coefficients $\mathbf{P} \in \mathbb{R}^{N_{\text{node}} \times N_{\text{eig}}}$, and first order moments of products of pairs of eigenfunctions $\mathbf{A}^x$, $\mathbf{A}^y$, and $\mathbf{A}^z$, given a length scale ℓ_{min} and guiding number N_{eig}. The function `solve_mf` can then compute the resulting magnetization and signal, using the same parallel looping scheme as for `solve_btpde` (see Section 4.8). Once the Laplace eigendecomposition has been performed, computing the the magnetization is straightforward, and generally requires only a fraction of the time it takes to call `solve_btpde` directly. It does still however requiring assembling the mass matrix for projecting the initial spin density onto the Laplace eigenbasis and computing the signal.

4.11 Solve analytical model

SpinDoctor also allows for solving an analytical multilayer model, based on the article and corresponding code in [17]. This is a particular form of the previously specified matrix formalism model, where the geometry is assumed to consist of structures only varying in the radial direction, with dimension one (slabs), two (cylinders), or three (spheres). The matrix formalism is reduced to finding radial Laplace eigenvalues and eigenfunctions, providing an analytical solution for the signal attenuation.

The function `solve_analytical` is triggered by the precense of the field `setup.analytical`. In order to call this function, the geometry has to be of type `"cylinder"` or `"sphere"`, with $N_{\text{cell}} = 1$. The geometry can optionally include `"in"` and/or `"ecs"` compartments. For cylinders, the axon, myelin, and ECS tops and bottoms are assumed to be reflective, i.e. $\kappa^{\text{in}} = \kappa^{\text{out}} = \kappa^{\text{ecs}} = 0$. For spheres, the outer

ECS boundary can have a surface relaxivity $\kappa^{\text{ecs}} \geq 0$, or if the ECS is not included, the out compartment can have a surface relaxivity $\kappa^{\text{out}} \geq 0$.

This model assumes a radial gradient direction, which is always the case for a sphere, but requires a gradient direction in the x-y plane for cylinders. Currently, only `PGSE` is supported. The output is $S^{\text{ML}}(f, \|\boldsymbol{g}\|)$, and can be used to compare the signals from other simulations. While the geometry independent finite element approach used in `solve_mf` has errors from both eigenbasis truncation and finite element discretization, the multilayer approach in `solve_analytical` only has errors from the eigenvalue truncation. The solver parameters are `setup.analytical.length_scale` and `setup.analytical.eigstep`, the latter giving the smallest expected distance between to radial eigenvalues $\lambda = \alpha^2$. For eigenvalues that are closer than this distance, only one will be identified. For more information about the spacing between the eigenvalues, consult [17].

4.12 Visualization of results

To display results in one diffusion-encoding direction, the user can choose the following options:

1. the signal is plotted against the b-values;
2. the ADC is plotted against the geometrical compartment number.

We note that if the ADC from the short time approximation is negative (meaning the short diffusion time assumption is not satisfied), we do not display the negative ADC in the plot.

For results in multiple diffusion-encoding directions uniformly distributed on the three dimensional unit sphere, the user can call the function `plot_hardi`, which plots the signals are plotted as a surface plot. The dots are the end points of the vectors in the diffusion-encoding direction multiplied the signal. The color gives the magnitude of the signal, which can be the signal or the ADC.

The user can also choose to plot the magnetization solution at the surface nodes of the finite elements mesh, using the functions `plot_field` or `plot_field_everywhere`. These functions can also be used to plot eigenfunctions.

5 Finite elements meshes of neurons

The diffusion MRI signal arising from neurons can be numerically simulated by solving the Bloch-Torrey partial differential equation. In order to facilitate the diffusion MRI simulation of realistic neurons by the research community, we constructed finite element meshes for a group of 36 pyramidal neurons and a group of 29 spindle neurons whose morphological descriptions were found in the publicly available neuron repository *NeuroMorpho.Org* [18]. These finite elements meshes range from having 15163 nodes to 622553 nodes. We also broke the neurons into the soma and dendrite branches and created finite elements meshes for these cell components. Through the Neuron Module, these neuron and components finite element meshes can be seamlessly coupled with the functionalities of SpinDoctor to provide the diffusion MRI signal that can be attributed to spins inside neurons.

As this time, we have not implemented the Neuron Module for coupled compartments linked by permeable membranes. Rather, the diffusion MRI signal is computed with zero permeability on the compartment boundaries. The current emphasis of the Neuron Module is to show how the geometrical structure of neurons affect the diffusion MRI signal. Thus, some of the input parameters related to multiple compartment models in SpinDoctor are not applicable in the current version of the Neuron Module. However, we have kept the exactly same input formats as SpinDoctor in anticipation of the future development of the Neuron Module for permeable membranes. In particular, the various compartments in SpinDoctor have designations as IN, OUT, and ECS, and in the Neuron Module, the geometry defined by the finite element mesh is designated as the OUT compartment.

Table 11 lists the names and the finite element mesh sizes of the group of 36 pyramidal neurons and the group of 29 spindle neurons. Table 12 shows the morphological characteristics for the neurons. The neuron models and the measurement data are from [19] and [20].

Neuron ID	Num of FE mesh nodes	Neuron ID	Num of FE mesh nodes
03a_spindle2aFI	38202	*02a_pyramidal2aFI*	119156
03a_spindle6aFI	44000	*02b_pyramidal1aACC*	45216
03b_spindle4aACC	17370	*02b_pyramidal1aFI*	105384
03b_spindle5aACC	26345	*03a_pyramidal9aFI*	81530
03b_spindle6aACC	26792	*03b_pyramidal2aACC*	28183
03b_spindle7aACC	21618	*03b_pyramidal3aACC*	27607
04b_spindle3aFI	51265	*03b_pyramidal3aFI*	151362
05b_spindle5aFI	22457	*03b_pyramidal4aFI*	96177
06b_spindle8aACC	15163	*03b_pyramidal9aFI*	66162
07b_spindle9aACC	54952	*04a_pyramidal4aACC*	150897
08a_spindle13aACC	46293	*04a_pyramidal5aACC*	89256
09o_spindle7aFI	38992	*04b_pyramidal5aFI*	95784
09o_spindle8aFI	60755	*04b_pyramidal6aACC*	87195
10a_spindle18aACC	25797	*04b_pyramidal6aFI*	90482
12a_spindle19aACC	31841	*04b_pyramidal7aACC*	622553
12o_spindle9aFI	29320	*05a_pyramidal10aACC*	201506
13o_spindle10aFI	43081	*05a_pyramidal8aACC*	139975
15o_spindle12aFI	101548	*05b_pyramidal7aFI*	208203
16o_spindle13aFI	18266	*05b_pyramidal8aFI*	124350
19o_spindle14aFI	25786	*05b_pyramidal9aACC*	366659
21o_spindle15aFI	28822	*06a_pyramidal11aACC*	319574
23o_spindle16aFI	30073	*06b_pyramidal10aFI*	106808
25o_spindle17aFI	52919	*06b_pyramidal12aACC*	277718
26o_spindle18aFI	36239	*07a_pyramidal13aACC*	155854
27o_spindle19aFI	50807	*07b_pyramidal14aACC*	309789
28o_spindle20aFI	56036	*08o_pyramidal11aFI*	419651
28o_spindle21aFI	17581	*10a_pyramidal15aACC*	56184
29o_spindle22aFI	18414	*11a_pyramidal16aACC*	222732
30o_spindle23aFI	26357	*11o_pyramidal12aFI*	380293
22o_pyramidal16aFI	389878	*17o_pyramidal13aFI*	326989
24o_pyramidal17aFI	245058	*18o_pyramidal14aFI*	338453
25o_pyramidal18aFI	71209	*20o_pyramidal15aFI*	247116
31o_pyramidal19aFI	619390		

Table 11: Names and sizes of all the neuron finite elements meshes generated by Tetgen with default settings (H_{tetgen} is not defined). The number of FE elements (not shown) is approximately four times the number of FE nodes.

Neuron ID	Brain region	Average diam. (μm)	Height (μm)	Soma vol. (μm^3)	Total vol. (μm^3)
02a_pyramidal2aFI	fronto-insula	1.27	404.85	19701.05	25639.61
02b_pyramidal1aACC	anterior cingulate	1.58	363.08	9065.56	11579.71
02b_pyramidal1aFI	fronto-insula	1.62	381.56	22475.52	29804.96
03a_pyramidal9aFI	fronto-insula	2.15	532.30	22557.27	30189.28
03a_spindle2aFI	fronto-insula	1.74	387.16	13406.27	17684.23
03a_spindle6aFI	fronto-insula	1.66	501.47	33458.19	37812.72
03b_pyramidal2aACC	anterior cingulate	1.48	189.29	2977.21	4487.44
03b_pyramidal3aACC	anterior cingulate	1.14	188.45	6005.06	6891.04
03b_pyramidal3aFI	fronto-insula	1.84	496.35	32510.62	46154.08
03b_pyramidal4aFI	fronto-insula	1.33	414.70	35253.85	39324.87
03b_pyramidal9aFI	fronto-insula	1.92	430.06	15263.14	20532.57
03b_spindle4aACC	anterior cingulate	1.43	336.33	3098.39	4070.19
03b_spindle5aACC	anterior cingulate	1.49	221.52	11925.78	13242.53
03b_spindle6aACC	anterior cingulate	1.33	398.36	4027.74	6058.67
03b_spindle7aACC	anterior cingulate	1.18	369.51	4982.41	6076.52
04a_pyramidal4aACC	anterior cingulate	1.52	705.96	5684.55	13637.33
04a_pyramidal5aACC	anterior cingulate	1.86	410.59	15010.43	24648.35
04b_pyramidal5aFI	fronto-insula	1.78	480.13	10312.87	17184.26
04b_pyramidal6aACC	anterior cingulate	1.41	465.88	3129.97	7497.12
04b_pyramidal6aFI	fronto-insula	1.56	310.21	14718.05	21708.21
04b_pyramidal7aACC	anterior cingulate	1.3	610.42	17060.60	28552.49
04b_spindle3aFI	fronto-insula	2.71	391.14	22569.99	28404.13
05a_pyramidal10aACC	anterior cingulate	1.74	281.20	16604.06	21826.41
05a_pyramidal8aACC	anterior cingulate	1.29	430.37	24709.77	29778.79
05b_pyramidal7aFI	fronto-insula	2.18	281.02	25720.11	32731.05
05b_pyramidal8aFI	fronto-insula	1.66	361.45	32527.06	44679.46
05b_pyramidal9aACC	anterior cingulate	1.56	650.60	23948.05	40014.54
05b_spindle5aFI	fronto-insula	2.35	381.88	15383.08	18190.63
06a_pyramidal11aACC	anterior cingulate	1.46	437.60	17222.02	29995.02
06b_pyramidal10aFI	fronto-insula	1.92	365.18	43127.81	52179.53
06b_pyramidal12aACC	anterior cingulate	1.52	324.94	17181.33	24931.32
06b_spindle8aACC	anterior cingulate	1.92	342.21	18237.49	19462.92
07a_pyramidal13aACC	anterior cingulate	1.37	325.73	6254.53	8738.01
07b_pyramidal14aACC	anterior cingulate	1.67	350.40	16053.07	22772.96
07b_spindle9aACC	anterior cingulate	1.75	437.87	21344.83	27307.48
08a_spindle13aACC	anterior cingulate	1.74	814.45	9911.07	14113.32
08o_pyramidal11aFI	fronto-insula	1.91	421.68	11512.38	24326.94
09o_spindle7aFI	fronto-insula	2.90	472.87	22052.10	27905.89
09o_spindle8aFI	fronto-insula	2.05	376.73	11923.76	15189.32
10a_pyramidal15aACC	anterior cingulate	1.40	341.48	8522.11	10960.84
10a_spindle18aACC	anterior cingulate	1.57	457.90	5895.17	7219.28
11a_pyramidal16aACC	anterior cingulate	1.27	486.31	8807.01	12263.84
11o_pyramidal12aFI	fronto-insula	1.91	369.34	70786.62	79516.92
12a_spindle19aACC	anterior cingulate	2.05	431.22	12178.08	15618.67
12o_spindle9aFI	fronto-insula	3.41	305.31	29983.79	36678.18
13o_spindle10aFI	fronto-insula	2.69	516.92	39866.55	46022.15
15o_spindle12aFI	fronto-insula	3.60	604.57	53192.65	79170.43
16o_spindle13aFI	fronto-insula	2.17	364.66	17467.88	18888.13
17o_pyramidal13aFI	fronto-insula	1.89	340.77	11004.30	21167.19
18o_pyramidal14aFI	fronto-insula	1.74	288.41	69851.56	78999.20
19o_spindle14aFI	fronto-insula	2.18	232.21	10507.15	12905.43
20o_pyramidal15aFI	fronto-insula	1.82	383.18	22344.32	27667.19
21o_spindle15aFI	fronto-insula	2.36	286.33	17567.69	29466.53

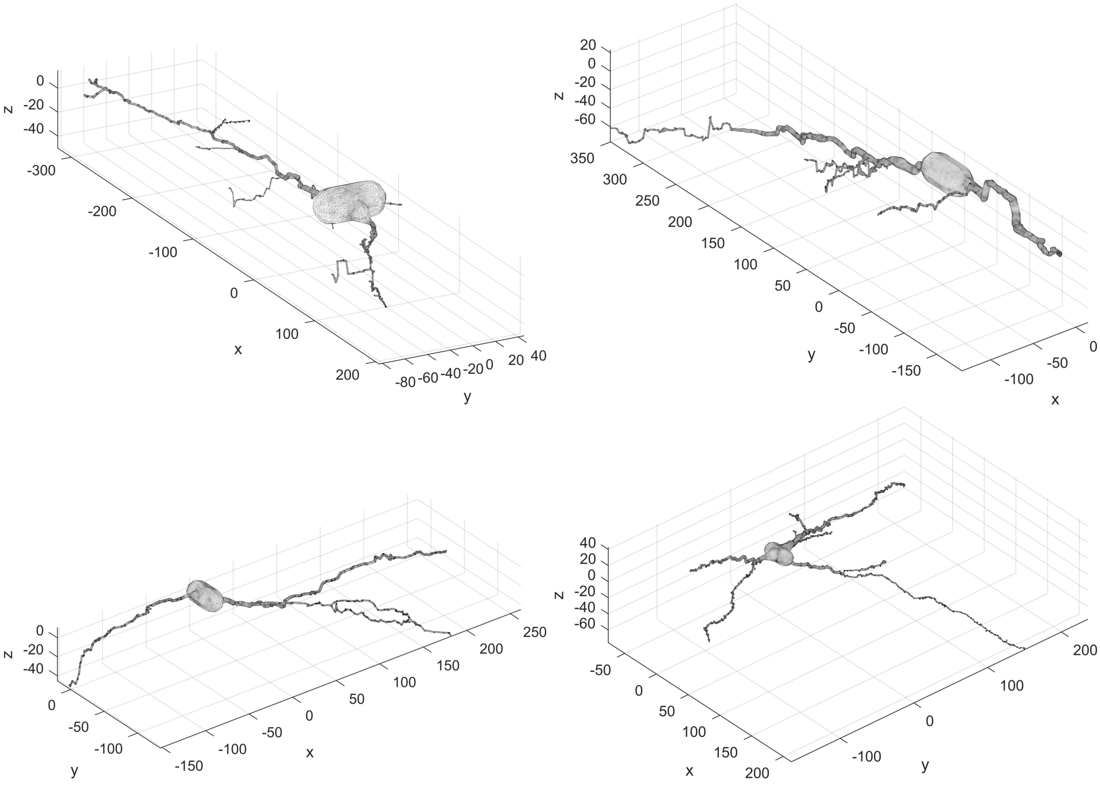

Figure 5: The finite elements meshes of four spindle neurons. The unit is μm. Top left: *03a_spindle6aFI*. Top right: *28o_spindle20aFI*. Bottom left: *09o_spindle8aFI*. Bottom right: *25o_spindle17aFI*.

22o_pyramidal16aFI	fronto-insula	1.94	585.35	18776.05	29441.43
23o_spindle16aFI	fronto-insula	1.67	420.05	10429.13	13482.93
24o_pyramidal17aFI	fronto-insula	2.04	371.99	40986.40	47377.09
25o_pyramidal18aFI	fronto-insula	1.80	364.05	18587.13	23572.15
25o_spindle17aFI	fronto-insula	1.79	358.70	7897.44	13563.26
26o_spindle18aFI	fronto-insula	2.27	442.65	52911.93	56084.44
27o_spindle19aFI	fronto-insula	1.73	275.08	20640.14	25423.96
28o_spindle20aFI	fronto-insula	3.00	520.69	35442.59	51267.07
28o_spindle21aFI	fronto-insula	2.62	298.57	35579.06	37783.31
29o_spindle22aFI	fronto-insula	3.52	402.84	62928.22	83279.12
31o_pyramidal19aFI	fronto-insula	2.26	303.55	65950.80	86376.72

Table 12: The morphological characteristics of the neurons.

As an illustration, we show the finite elements meshes of 4 spindle neurons in Figure 5, and in Figure 6 we show the finite elements meshes of 4 pyramidal neurons.

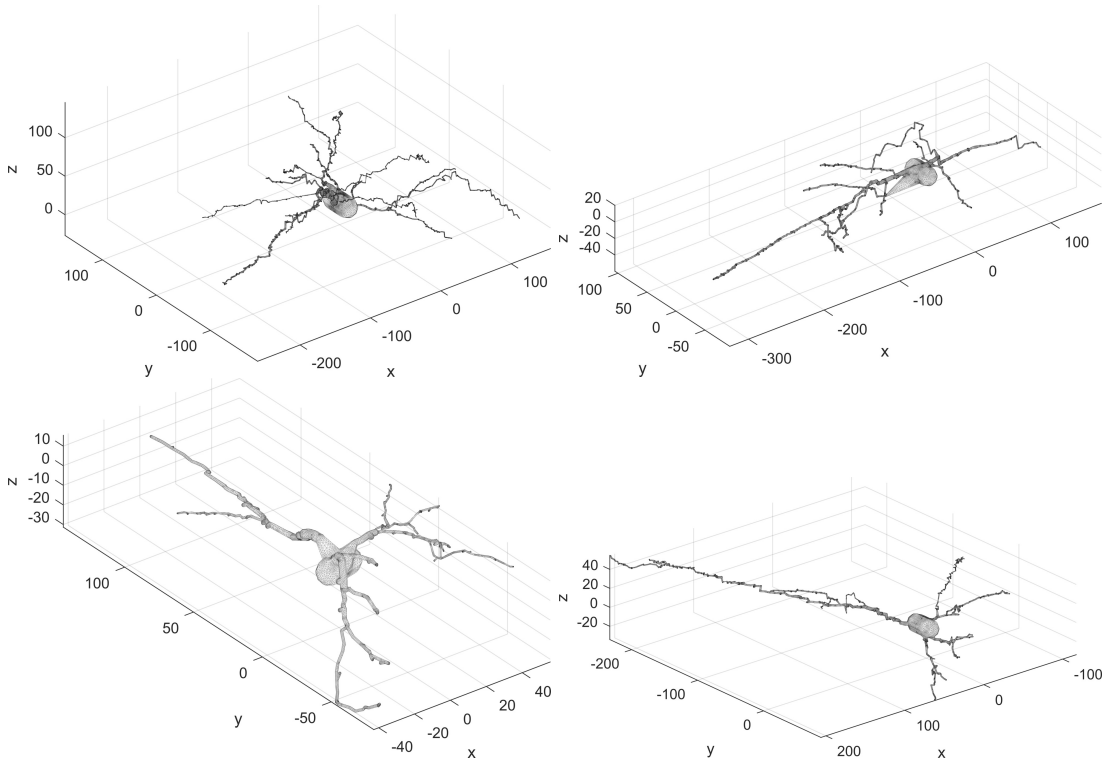

Figure 6: The finite elements meshes of four pyramidal neurons. The unit is μm. Top left: *02a_pyramidal2aFI*. Top right: *03b_pyramidal9aFI*. Bottom left: *03b_pyramidal2aACC*. Bottom right: *10a_pyramidal15aACC*.

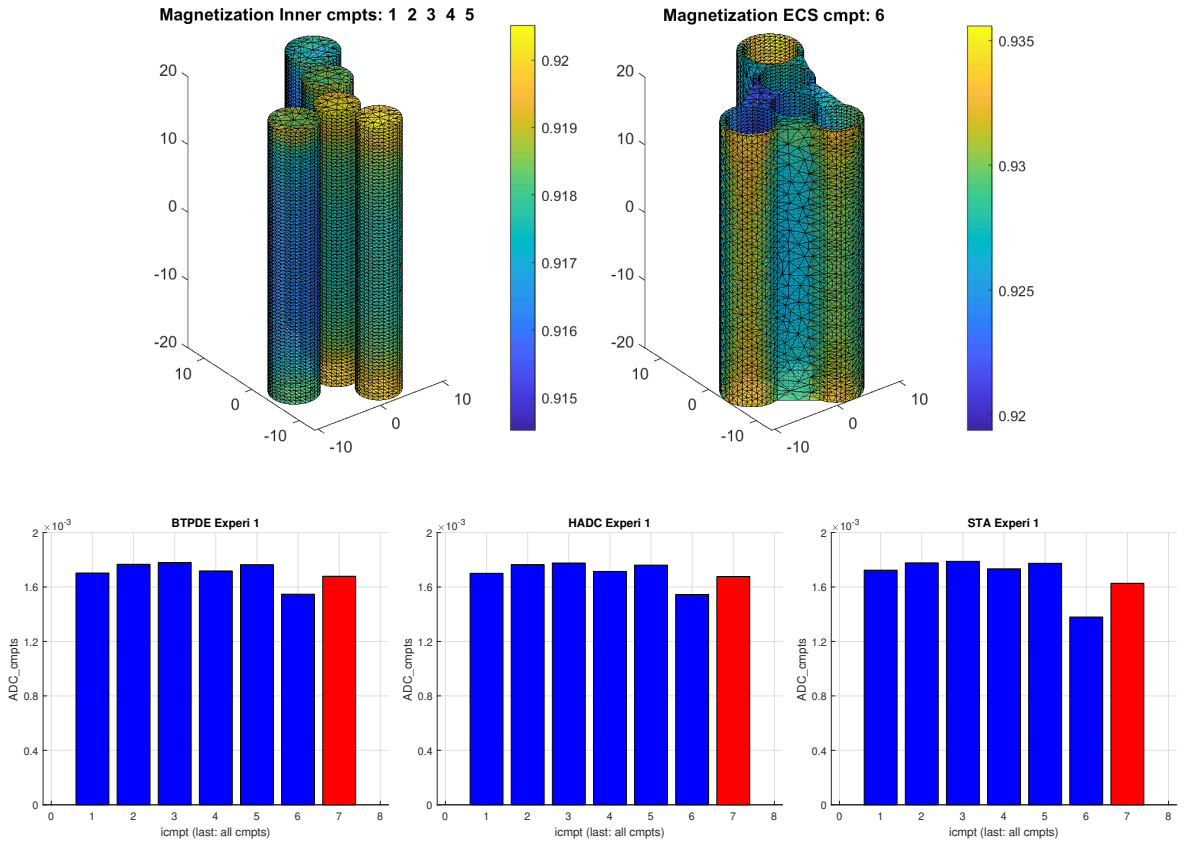

Figure 7: Geometry: 5 cylinders, tight wrap ECS, ecs_gap $= 0.2$, $\boldsymbol{u_g} = (1,1,1)^{\mathsf{T}}/\sqrt{3}$, $\sigma^{\text{out}} = \sigma^{ecs} = 2\times 10^{-3}\text{mm}^2/\text{s}$, $\kappa = 0\text{m/s}$, OGSE cosine ($\delta = 14\text{ms}, \Delta = 14\text{ms}$, nperiod $= 6$). The vertical bars indicate the ADC in each compartment. The ADC in the rightmost position is the ADC that takes into account the diffusion in all the compartments.

6 SpinDoctor examples

In this section we show some prototypical examples using the available functionalities of SpinDoctor.

6.1 Comparison of BTPDE and HADC with Short Time Approximation

In Fig. 7 we show that both BTPDE and HADC solutions match the STA values at short diffusion times for cylindrical cells (compartments 1 to 5). We also show that for the ECS (compartment 6), the STA is too low, because it does not account for the fact that spins in the ECS can diffuse around several cylinders. This also shows that when the interfaces are impermeable, the BTPDE ADC and that from the HADC model are identical. The diffusion-encoding sequence here is cosine OGSE with 6 periods.

6.2 Permeable membranes

In Fig. 8 we show the effect of permeability: the Bloch-Torrey PDE model includes permeable membranes ($\kappa = 1\times 10^{-3}\text{m/s}$) whereas the HADC has impermeable membranes. We see in the permeable case, the ADC in the spheres are higher than in the impermeable case, whereas the ECS show reduced ADC because the faster diffusing spins in the ECS are allowed to move into the slowly diffusing spherical cells. We note that in the permeable case, the ADC in each compartment is obtained by using the fitting formula involving the logarithm of the dMRI signal, and we defined the "signal" in a compartment as the total magnetization in that compartment at T_{echo}, which is just the integral of the solution of the Bloch-Torrey PDE in that compartment.

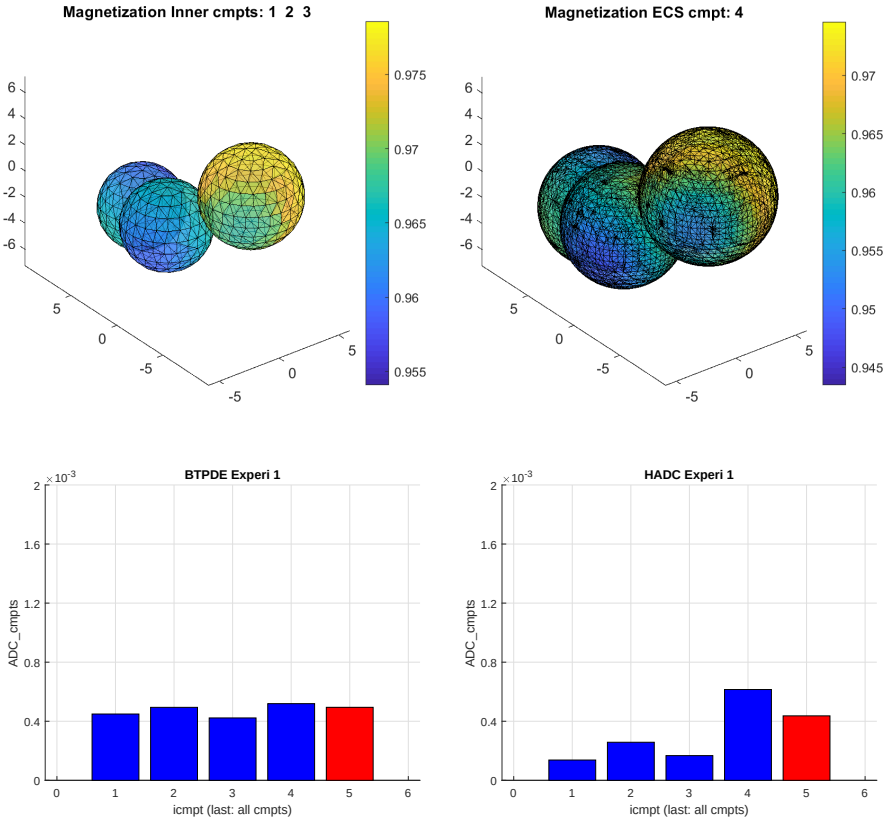

Figure 8: Geometry: 3 spheres, tight wrap ECS, ecs_gap $= 0.3$, $\boldsymbol{u_g} = (1,1,0)^{\mathsf{T}}/\sqrt{2}$, $\sigma^{\text{in}} = \sigma^{ecs} = 2 \times 10^{-3}\text{mm}^2/\text{s}$, $\kappa = 1 \times 10^{-3}\text{m/s}$ (left), $\kappa = 0\text{m/s}$ (right). PGSE ($\delta = 5\text{ms}, \Delta = 5\text{ms}$). The vertical bars indicate the ADC in each compartment. The ADC in the rightmost position is the ADC that takes into account the diffusion in all the compartments.

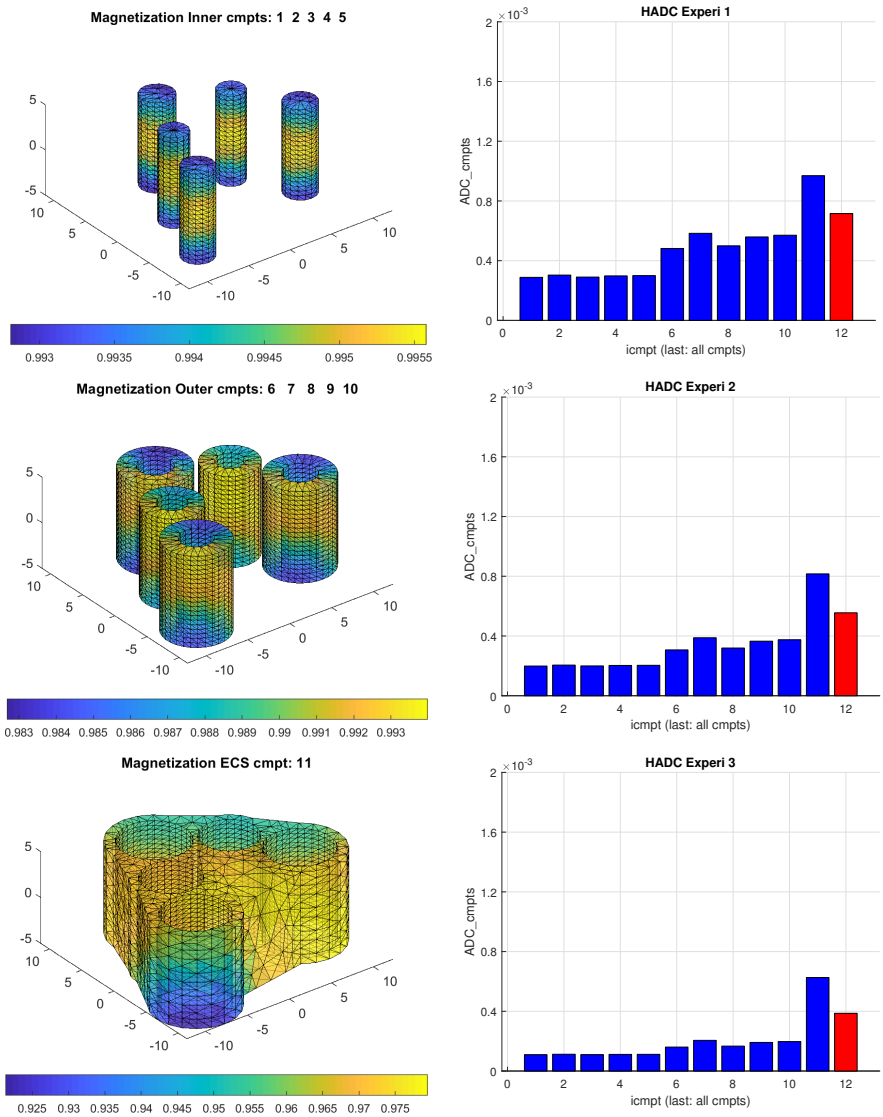

Figure 9: Geometry: 5 cylinders, myelin layer, $r_{\text{in}}/r_{\text{out}} = 0.5$, tight wrap ECS, ecs_gap $= 0.3$, $\kappa = 0$m/s, $\boldsymbol{u_g} = (1,1,1)^{\mathsf{T}}/\sqrt{3}$, $\sigma^{\text{in}} = \sigma^{\text{out}} = \sigma^{\text{ecs}} = 2 \times 10^{-3}$mm^2/s, 3 experiments: PGSE ($\delta = 5$ms, $\Delta = 5, 10, 20$ms). Left: the magnetization at $\Delta = 5$ms. Right: the ADC values. The vertical bars indicate the ADC in each compartment. The ADC in the rightmost position is the ADC that takes into account the diffusion in all the compartments.

6.3 Myelin layer

In Fig. 9 we show the diffusion in cylindrical cells, the myelin layer, and the ECS. The ADC is higher in the myelin layer than in the cells, because for spins in the myelin layer diffusion occurs in the tangential direction (around the circle). At longer diffusion times, the ADC of both the myelin layer and the cells becomes very low. The ADC is the highest in the ECS, because the diffusion distance can be longer than the diameter of a cell, since the diffusing spins can move around multiple cells.

6.4 Twisting and bending

In Fig. 10 we show the effect of bending and twising in cylindrical cells in multiple gradient directions. The HADC is obtained in 20 directions uniformly distributed in the sphere. We used spherical harmonics interpolation to interpolate the HADC in the entire sphere. Then we deformed the radius of the unit sphere to be proportional to the interpolated HADC and plotted the 3D shape. The color axis also indicates the value of the interpolated HADC.

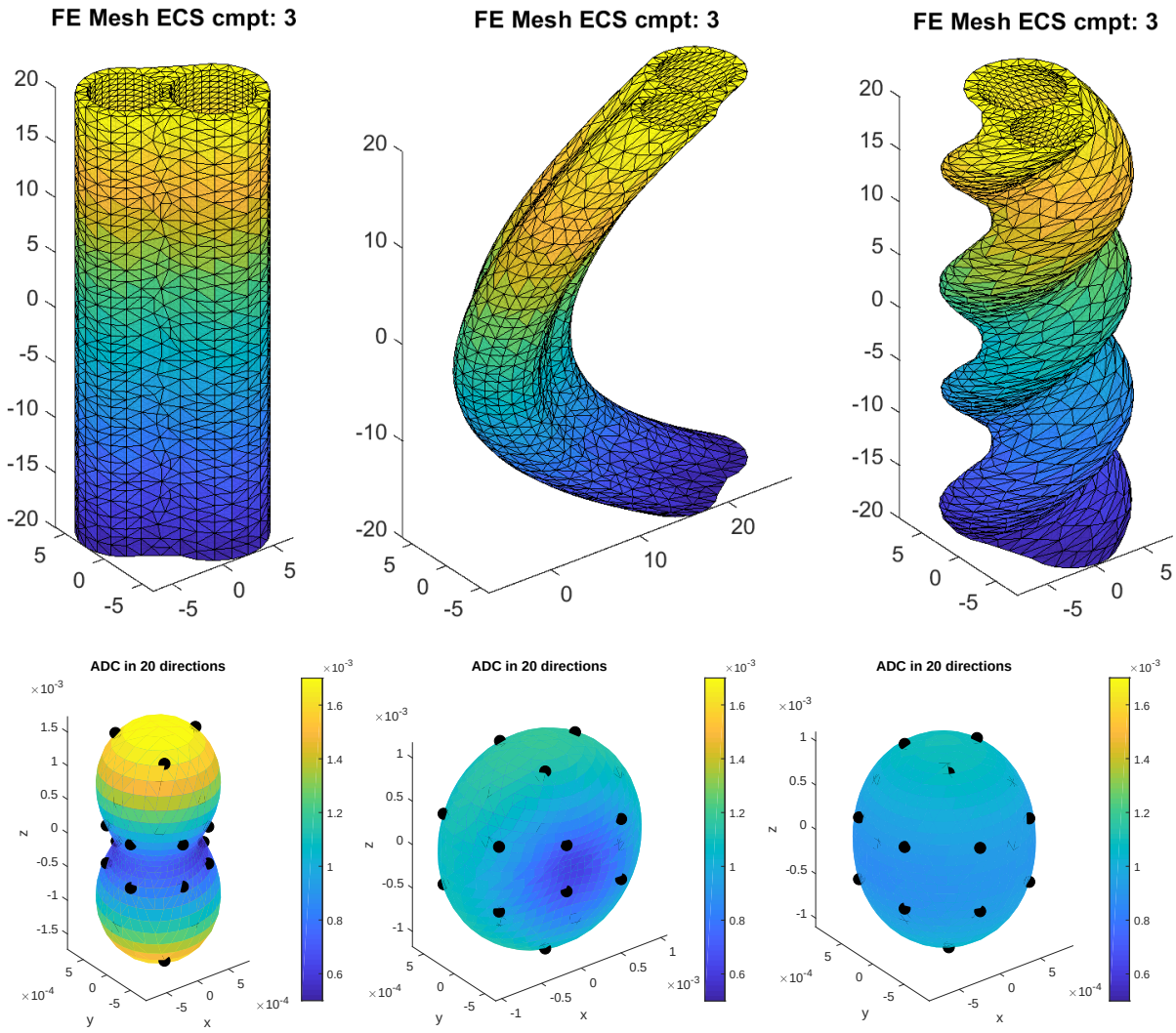

Figure 10: Geometry: 2 cylinders, no myelin layer, tight wrap ECS, ecs_gap = 0.3, $\kappa = 0$m/s, $\sigma^{\text{out}} = \sigma^{\text{ecs}} = 2 \times 10^{-3}$mm^2/s, PGSE ($\delta = 2.5$ms, $\Delta = 5$ms). Left: canonical configuration. Middle: bend parameter = 0.05. Right: twist parameter = 0.30. Top: FE mesh of the ECS (the FE mesh of the axon compartments numbered 1 and 2 not shown). Bottom: interpolated values of the HADC on the unit sphere, and then the sphere was distorted to reflect the value of the HADC. The color axis also gives the value of the HADC in the various gradient directions. The black dots indicate the 20 original gradient-directions in which the HADC was simulated. The spherical harmonics interpolation takes the 20 original directions into 900 directions uniformly distributed on the sphere.

6.5 Neuron examples

In Figures 11, 12, and 13 we display figures from the functions plot_femesh, plot_field, plot_signal, plot_hardi. The geometrical configuration is the spindle neuron *03b_spindle6aFI*. The intrinsic diffusion coefficient is set to $\mathcal{D}_0 = 2 \times 10^{-3}$mm^2/s, with nonpermeable membrane. We simulated 2 diffusion-encoding sequences:

1. f_1 is PGSE ($\delta = 10.6$ms, $\Delta = 13$ms);
2. f_2 is PGSE ($\delta = 10.6$ms, $\Delta = 73$ms).

6.6 Matrix Formalism examples

Below we display some example outputs from the Matrix Formalism Module. The geometrical configuration is one dendrite branch of a spindle neuron, *03b_spindle6aACC_dendrites_2*, shown in Figure 14.

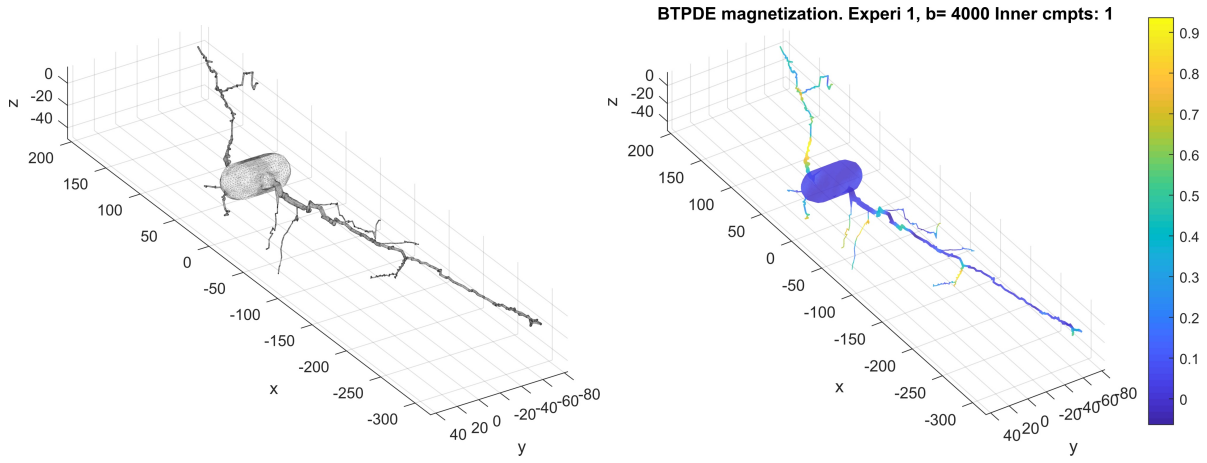

Figure 11: Left: The finite elements mesh of the neuron *03b_spindle6aFI* (using the command plot_femesh). The unit is μm. Right: The PDE solution (magnetization) on the neuron *03b_spindle6aFI* in the diffusion-encoding direction (1, 1, 1) (using the command plot_field). The diffusion-encoding sequence is PGSE ($\delta = 10.6$ms, $\Delta = 13$ms). The b-value is $b = 4000$s/mm^2.

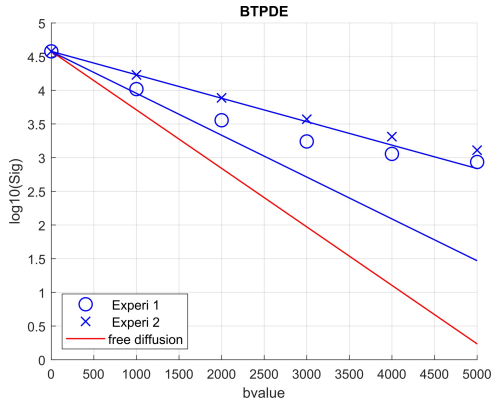

Figure 12: The (non-normalized) simulated signal of the neuron *03b_spindle6aFI* in the diffusion-encoding direction (1, 1, 1) (using the command plot_signal). For Experiment 1, the diffusion-encoding sequence is PGSE ($\delta = 10.6$ms, $\Delta = 13$ms). For Experiment 2, the diffusion-encoding sequence is PGSE ($\delta = 10.6$ms, $\Delta = 73$ms).

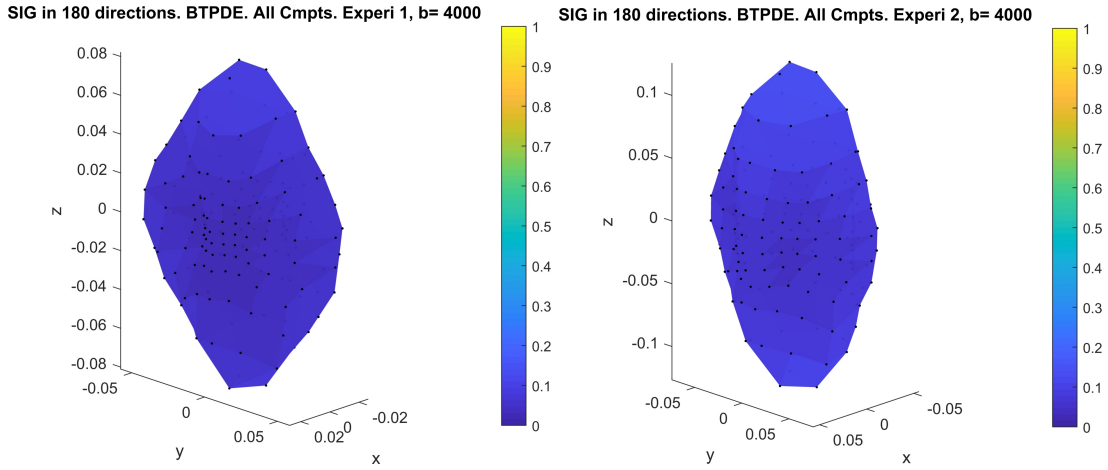

Figure 13: The simulated signal in 180 diffusion-encoding directions for the neuron *03b_spindle6aFI* (using the command plot_hardi). Left: The signal in 180 directions with PGSE ($\delta = 10.6$ms, $\Delta = 13$ms) and $b = 4000$s/mm^2. Right: The signal in 180 directions with PGSE ($\delta = 10.6$ms, $\Delta = 73$ms) and $b = 4000$s/mm^2.

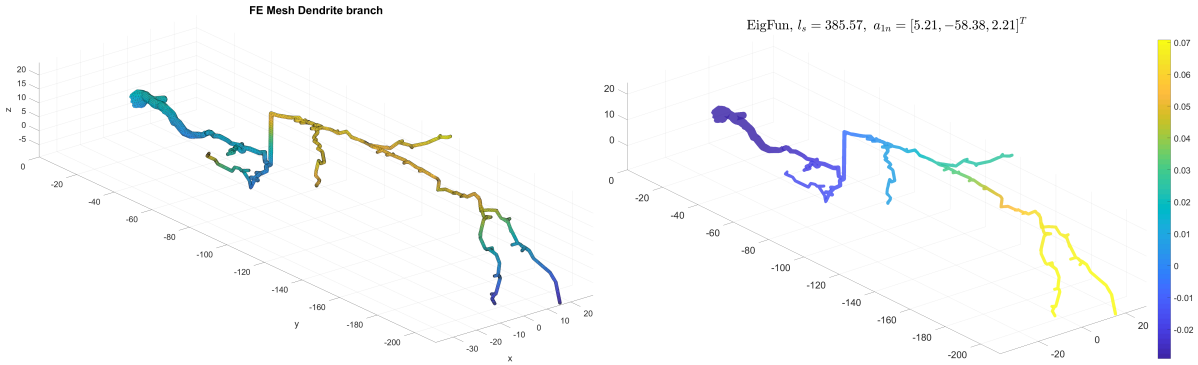

Figure 14: Left: The finite elements mesh of the spindle neuron dendrite branch *03b_spindle6aACC_dendrites_2* (using the command plot_femesh). Right: The eigenfunction with the associated length scale $L = 383.57\mu$m and the first moments vector $\boldsymbol{a}_{1n} = [5.21, -58.38, 2.21]^{\mathsf{T}}$ (using the command plot_field).

The intrinsic diffusion coefficient is set to $\mathcal{D}_0 = 2 \times 10^{-3}\text{mm}^2/\text{s}$, with an impermeable membrane. We simulated 2 diffusion-encoding sequences:

1. f_1 is PGSE ($\delta = 10.6$ms, $\Delta = 13$ms),
2. f_2 is PGSE ($\delta = 10.6$ms, $\Delta = 73$ms).

7 Commented drivers for typical simulations

7.1 Solve BTPDE, HADC, MF or multilayer models and compare apparent diffusion coefficients

```
%DRIVER_SPINDOCTOR Solve BTPDE, HADC, MF or analytical.
%   Compare different ADC. Plot results in many directions.
%
%   It is highly recommended to read this driver to understand the
    workflow
%   of SpinDoctor.
%
%   The user is advised to read the latest version
%   from \url{https://github.com/jingrebeccali/SpinDoctor}

clear
restoredefaultpath

% Add SpinDoctor
addpath(genpath("src"));


%% Define inputs

% Get setup
addpath setups

% setup_1axon_analytical;
% setup_1sphere_analytical;
% setup_15spheres;
% setup_2axons_deform;
```

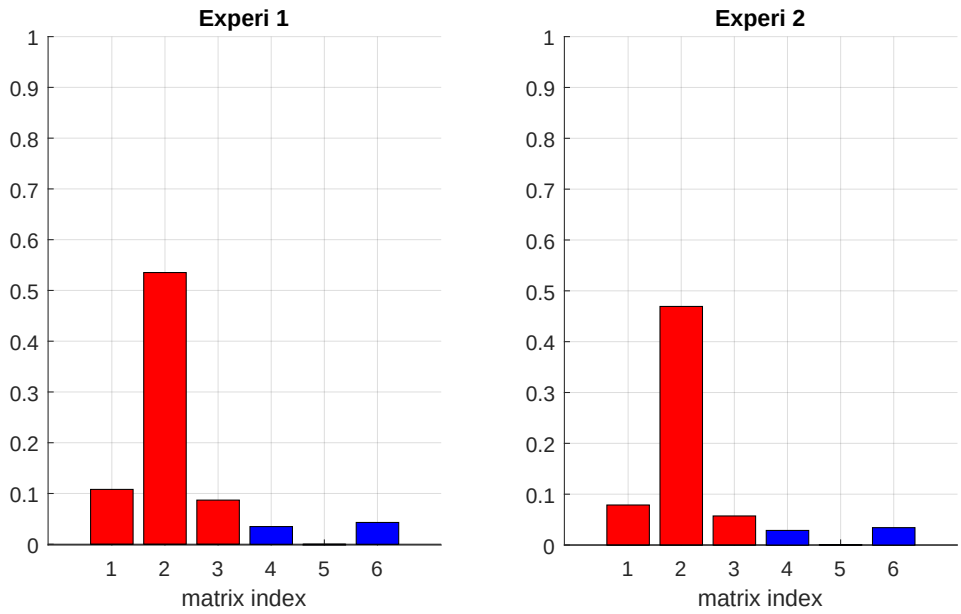

Figure 15: The 6 entries of the normalized diffusion tensor $\mathbf{D}^{\mathrm{MF}}(f)/\mathcal{D}_0$ for the dendrite branch. Indices 1 to 6 are in the order of $\{D^{\mathrm{MF}}_{xx}(f), D^{\mathrm{MF}}_{yy}(f), D^{\mathrm{MF}}_{zz}(f), D^{\mathrm{MF}}_{xy}(f), D^{\mathrm{MF}}_{xz}(f), D^{\mathrm{MF}}_{yz}(f)\}/\mathcal{D}_0$. The diagonal entries of $\mathbf{D}^{\mathrm{MF}}(f)/\mathcal{D}_0$ are shown in red and the off-diagonal entries shown in blue (if non-zero). From left to right: Experiment 1: f_1 is PGSE ($\delta = 10.6$ms, $\Delta = 13$ms), Experiment 2: f_2 is PGSE ($\delta = 10.6$ms, $\Delta = 73$ms). (Using the command plot_diffusion_tensor).

```
% setup_5axons_myelin_relax;
% setup_neuron;
setup_4axons_flat;
% setup_30axons_flat;

% Choose to see some of the typical plots or not
do_plots = true;


%% Prepare experiments

% Set up the PDE model in the geometrical compartments.
setup.pde = prepare_pde(setup);

% Prepare experiments (gradient sequence, bvalues, qvalues, solvers)
setup = prepare_experiments(setup);

% Create or load finite element mesh
[femesh, surfaces, cells] = create_geometry(setup);

% Get volume and surface area quantities from mesh
[volumes, surface_areas] = get_vol_sa(femesh);

% Compute volume weighted mean of diffusivities over compartments
% Take trace of each diffusion tensor, divide by 3
mean_diffusivity = trace(sum(setup.pde.diffusivity .* shiftdim(volumes,
    -1), 3)) ...
    / (3 * sum(volumes));

% Initial total signal
initial_signal = setup.pde.initial_density * volumes';
```

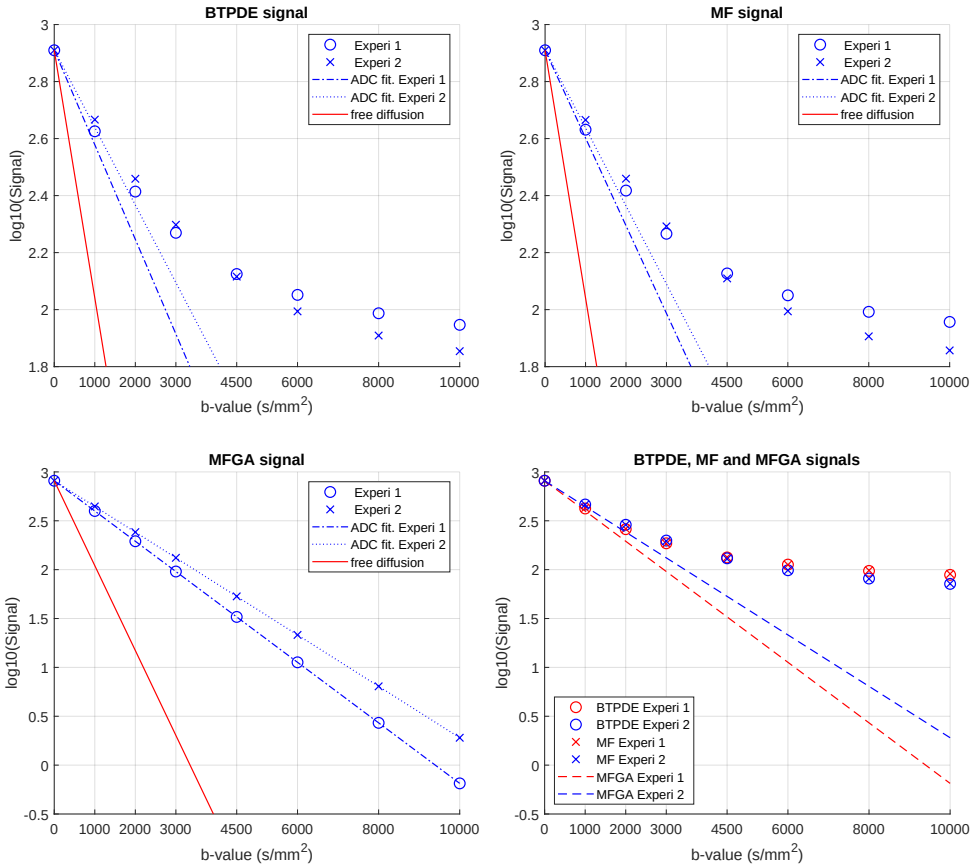

Figure 16: Top left: the BTPDE signals. Top right: the MF signals. Bottom left: the MFGA signals. The markers indicate the values of the simulated signals, the blue lines indicate the ADC fit of those signals, the red line is the signal of free diffusion at the intrinsic diffusion coefficient. Bottom right: the BTPDE, the MF, the MFGA signals plotted together. Experiment 1: f_1 is PGSE ($\delta = 10.6$ms, $\Delta = 13$ms), Experiment 2: f_2 is PGSE ($\delta = 10.6$ms, $\Delta = 73$ms). Diffusion-encoding direction $\boldsymbol{u_g} = (1,1,0)^\mathsf{T}/\sqrt{2}$ (using the command plot_signal).

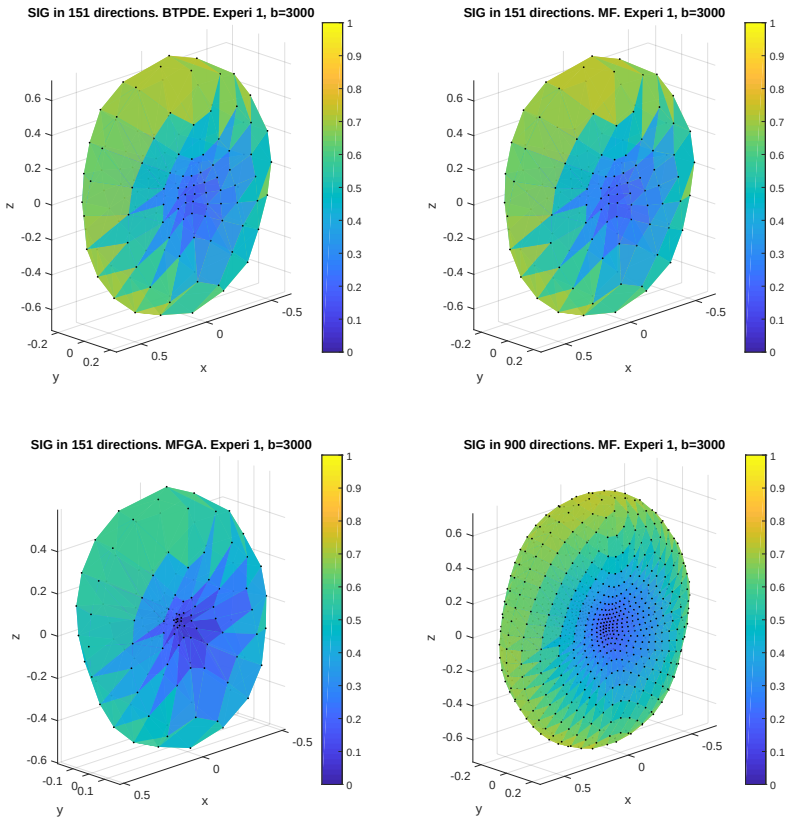

Figure 17: Top left: the BTPDE signals, S^{BTPDE}/S_0, in 151 diffusion-encoding directions. Top right: the MF signals, S^{MF}/S_0, in 151 diffusion-encoding directions. Bottom left: the MFGA signals, S^{MFGA}/S_0, in 151 diffusion-encoding directions. Bottom right: the MF signals, S^{MF}/S_0, in 900 diffusion-encoding directions. The black points are the magnitude of the signal attenuation multiplied by the diffusion-encoding direction. The color indicates the value of the signal attenuation. $b = 3000\mathrm{s/mm}^2$, PGSE, $\delta = 10.6\mathrm{ms}$, $\Delta = 13\mathrm{ms}$. (Using the command plot_hardi).

```
56
57
58  %% Perform experiments
59
60  % Short time approximation (STA) of the ADC
61  [sta_adc, sta_adc_allcmpts] = compute_adc_sta(femesh, setup);
62
63  % Free diffusion signal
64  free = compute_free_diffusion(setup.gradient.bvalues, setup.pde.
        diffusivity, ...
65      volumes, setup.pde.initial_density);
66
67  % Perform BTPDE experiments
68  if isfield(setup, "btpde")
69      % Solve BTPDE
70      btpde = solve_btpde(femesh, setup);
71
72      % Fit ADC from signal
73      btpde_fit = fit_signal(btpde.signal, btpde.signal_allcmpts, setup.
            gradient.bvalues);
74
75      % BTPDE direction averaged magnetization
76      btpde.magnetization_avg = average_magnetization(btpde.magnetization
            );
77  end
78
79  % Perform HADC experiment
80  if isfield(setup, "hadc")
81      % Solve HADC model
82      hadc = solve_hadc(femesh, setup);
83  end
84
85  % Perform MF experiments
86  if isfield(setup, "mf")
87      % Perform Laplace eigendecomposition
88      eiglim = length2eig(setup.mf.length_scale, mean_diffusivity);
89      lap_eig = compute_laplace_eig(femesh, setup.pde, eiglim, setup.mf.
            neig_max);
90
91      % Compute length scales of eigenvalues
92      lap_eig.length_scales = eig2length(lap_eig.values, mean_diffusivity
            );
93
94      % Compute the JN value that relates the eigenmodes to their
            contribution
95      % to the Matrix Formalism signal for a diffusion-encoding sequence
96      mf_jn = compute_mf_jn(lap_eig.values, mean_diffusivity, setup);
97
98      % Compute the Matrix Formalism effective diffusion tensor
99      diffusion_tensor = compute_mf_diffusion_tensor(lap_eig, mf_jn,
            mean_diffusivity);
100
101     % Compute MF magnetization
102     mf = solve_mf(femesh, setup, lap_eig);
103
104     % Fit ADC from MF signal
105     mf_fit = fit_signal(mf.signal, mf.signal_allcmpts, setup.gradient.
            bvalues);
```

```
106
107     % MF direction averaged magnetization
108     mf.magnetization_avg = average_magnetization(mf.magnetization);
109
110     % Compute MFGA signal
111     mfga = compute_mfga_signal(setup, initial_signal, diffusion_tensor)
            ;
112 end
113
114 % Perform analytical experiment
115 if isfield(setup, "analytical")
116     % Solve analytical analytical model
117     analytical_signal = solve_analytical(setup);
118 end
119
120
121 %% Postprocess
122
123 if ~do_plots
124     return
125 end
126
127 if setup.geometry.cell_shape == "sphere" || setup.geometry.cell_shape
        == "cylinder"
128     % Plot cells in canonical configuration
129     plot_cells(cells, setup);
130 end
131
132 % Plot surface triangulation
133 plot_surface_triangulation(surfaces);
134
135 % Plot the finite element mesh
136 plot_femesh(femesh, setup.pde.compartments);
137 % plot_femesh_everywhere(femesh, "");
138
139 % Plot information about the geometry
140 plot_geometry_info(setup, volumes, surface_areas);
141
142 % Get sizes
143 ncompartment = length(setup.pde.compartments);
144 namplitude = length(setup.gradient.values);
145 nsequence = length(setup.gradient.sequences);
146 ndirection = setup.gradient.ndirection;
147
148 % Plot BTPDE magnetization in some directions
149 if isfield(setup, "btpde")
150     for idir = 1 % :ndirection
151         for iseq = 1 % :nsequence
152             for iamp = 1 % :namplitude
153                 b = setup.gradient.bvalues(iamp, iseq);
154                 title_str = sprintf(...
155                     "BTPDE magnetization. Sequence %d of %d, b=%.2f",
                            ...
156                     iseq, nsequence, b);
157                 field = btpde.magnetization(:, iamp, iseq, idir);
158                 % plot_field(femesh, field, setup.pde.compartments,
                        title_str);
159                 plot_field_everywhere(femesh, field, title_str);
```

```
                % caxis([0 1]);
            end
        end
    end
    clear field
end

% Plot results in one or many directions
if ndirection == 1
    % Plot ADC short time approximation
    plot_adc(sta_adc, sta_adc_allcmpts, "STA");

    % Plot BTPDE results
    if isfield(setup, "btpde")
        % Plot the BTPDE signal, the S0*exp(-ADC*b) curve, and the
        % free diffusion curves together.
        plot_signal(setup.gradient.bvalues, btpde.signal_allcmpts, free
            .signal_allcmpts, ...
             btpde_fit.S0_allcmpts, btpde_fit.adc_allcmpts, "BTPDE")

        % Plot ADC fitted from BTPDE
        plot_adc(btpde_fit.adc, btpde_fit.adc_allcmpts, "BTPDE");

        % Plot computational time
        plot_timing(btpde.itertimes, femesh, "BTPDE", "B-value");
    end

    % Plot results from HADC experiment
    if isfield(setup, "hadc")
        % Plot ADC
        plot_adc(hadc.adc, hadc.adc_allcmpts, "HADC");

        % Plot computational time
        plot_timing(hadc.itertimes, femesh, "HADC", "Compartment");
    end
else
    % Plot STA ADC
    plot_hardi(setup.gradient.directions, sta_adc_allcmpts, "STA ADC
        all compartments");

    % Plot BTPDE signal in all directions
    if isfield(setup, "btpde")
        % Signal before applying the gradient pulse sequence
        S0 = sum(setup.pde.initial_density .* volumes);

        % Plot normalized signals
        title_str = "BTPDE total magnetization (normalized)";
        plot_hardi(setup.gradient.directions, btpde.signal_allcmpts /
            S0, title_str);
    end

    % Plot HADC in all directions
    if isfield(setup, "hadc")
        % Plot normalized ADC (ADC/D0)
        title_str = sprintf("HADC all compartments");
        plot_hardi(setup.gradient.directions, hadc.adc_allcmpts /
            mean_diffusivity, title_str);
    end
```

```
end % One or many directions


if isfield(setup, "mf")
    % Plot Matrix Formalism effective diffusion tensor
    plot_diffusion_tensor(diffusion_tensor, mean_diffusivity);

    % Relative error between BTPDE and MF signal
    signal_allcmpts_relerr = abs(mf.signal_allcmpts - btpde.
        signal_allcmpts) ...
        ./ max(abs(btpde.signal_allcmpts), [], 3);

    % Difference between BTPDE and MF signal, normalized by initial
        signal
    signal_allcmpts_abserr_vol = abs(mf.signal_allcmpts - btpde.
        signal_allcmpts) ./ initial_signal;

    if ndirection == 1
        % Plot BTPDE, MF and MFGA signals
        plot_signal(setup.gradient.bvalues, btpde.signal_allcmpts, free
            .signal_allcmpts, btpde_fit.S0_allcmpts, btpde_fit.adc, "
            BTPDE signal");
        plot_signal(setup.gradient.bvalues, mf.signal_allcmpts, free.
            signal_allcmpts, mf_fit.S0_allcmpts, mf_fit.adc, "MF signal
            ");
        % plot_signal(setup.gradient.bvalues, mfga.signal_allcmpts,
            free.signal_allcmpts, initial_signal, mfga.adc_allcmpts, "
            MFGA signal");
        plot_signal_btpde_mf(setup, btpde.signal_allcmpts, mf.
            signal_allcmpts, mfga.signal_allcmpts);

        % Display difference
        disp("Error (normalized by initial signal)");
        disp(signal_allcmpts_abserr_vol);
        disp("Relative error:");
        disp(signal_allcmpts_relerr);
    else
        % Plot HARDI signal
        plot_hardi(setup.gradient.directions, real(btpde.
            signal_allcmpts) / initial_signal, "BTPDE signal")
        plot_hardi(setup.gradient.directions, real(mf.signal_allcmpts)
            / initial_signal, "MF signal")

        % Plot relative difference
        fig_title = "Rel diff between BTPDE and MF";
        plot_hardi(setup.gradient.directions, signal_allcmpts_relerr,
            fig_title);

        % Plot difference normalized by volume
        fig_title = "Diff between BTPDE and MF normalized by volume";
        plot_hardi(setup.gradient.directions,
            signal_allcmpts_abserr_vol, fig_title);
    end
end
```

References

[1] J.-R. Li, V.-D. Nguyen, T. N. Tran, J. Valdman, C.-B. Trang, K. V. Nguyen, D. T. S. Vu, H. A. Tran, H. T. A. Tran, T. M. P. Nguyen, SpinDoctor: A MATLAB toolbox for diffusion MRI simulation, NeuroImage 202 (2019) 116120. doi:https://doi.org/10.1016/j.neuroimage.2019.116120.
URL http://www.sciencedirect.com/science/article/pii/S1053811919307116

[2] H. Si, Tetgen, a delaunay-based quality tetrahedral mesh generator, ACM Trans. Math. Softw. 41 (2) (2015) 11:1–11:36. doi:10.1145/2629697.
URL http://doi.acm.org/10.1145/2629697

[3] T. Rahman, J. Valdman, Fast matlab assembly of fem matrices in 2d and 3d: nodal elements, Applied Mathematics and Computation 219 (13) (2013) 7151–7158.

[4] D. V. Nguyen, J.-R. Li, D. Grebenkov, D. L. Bihan, A finite elements method to solve the bloch–torrey equation applied to diffusion magnetic resonance imaging, Journal of Computational Physics 263 (0) (2014) 283 – 302. doi:10.1016/j.jcp.2014.01.009.
URL http://www.sciencedirect.com/science/article/pii/S0021999114000308

[5] H.-H. Lee, E. Fieremans, D. S. Novikov, Realistic microstructure simulator (rms): Monte carlo simulations of diffusion in three-dimensional cell segmentations of microscopy images, Journal of Neuroscience Methods 350 (2021) 109018. doi:https://doi.org/10.1016/j.jneumeth.2020.109018.
URL https://www.sciencedirect.com/science/article/pii/S0165027020304416

[6] E. O. Stejskal, J. E. Tanner, Spin diffusion measurements: Spin echoes in the presence of a time-dependent field gradient, The Journal of Chemical Physics 42 (1) (1965) 288–292. doi:10.1063/1.1695690.

[7] P. T. Callaghan, J. Stepianik, Frequency-domain analysis of spin motion using modulated-gradient NMR, Journal of Magnetic Resonance, Series A 117 (1) (1995) 118–122.
URL http://www.sciencedirect.com/science/article/pii/S1064185885799597

[8] M. D. Does, E. C. Parsons, J. C. Gore, Oscillating gradient measurements of water diffusion in normal and globally ischemic rat brain, Magn. Reson. Med. 49 (2) (2003) 206–215. doi:10.1002/mrm.10385.

[9] J. Xu, M. Does, J. Gore, Numerical study of water diffusion in biological tissues using an improved finite difference method, Physics in medicine and biology 52 (7) (Apr. 2007).
URL http://view.ncbi.nlm.nih.gov/pubmed/17374905

[10] S. Schiavi, H. Haddar, J.-R. Li, A macroscopic model for the diffusion mri signal accounting for time-dependent diffusivity, SIAM Journal on Applied MathematicsAccepted. (2016).

[11] P. P. Mitra, P. N. Sen, L. M. Schwartz, P. Le Doussal, Diffusion propagator as a probe of the structure of porous media, Physical review letters 68 (24) (1992) 3555–3558.

[12] P. P. Mitra, P. N. Sen, L. M. Schwartz, Short-time behavior of the diffusion coefficient as a geometrical probe of porous media, Phys. Rev. B 47 (1993) 8565–8574.

[13] P. Callaghan, A simple matrix formalism for spin echo analysis of restricted diffusion under generalized gradient waveforms, Journal of Magnetic Resonance 129 (1) (1997) 74–84.
URL http://dx.doi.org/10.1006/jmre.1997.1233

[14] A. V. Barzykin, Theory of spin echo in restricted geometries under a step-wise gradient pulse sequence, Journal of Magnetic Resonance 139 (2) (1999) 342–353.
URL http://www.sciencedirect.com/science/article/pii/S1090780799917780

[15] N. J. Higham, The scaling and squaring method for the matrix exponential revisited, SIAM Journal on Matrix Analysis and Applications 26 (4) (2005) 1179–1193. doi:10.1137/04061101x.
URL https://doi.org/10.1137/04061101x

[16] A. Al-Mohy, N. Higham, A new scaling and squaring algorithm for the matrix exponential, SIAM Journal on Matrix Analysis and Applications 31 (01 2009). doi:10.1137/09074721X.

[17] D. S. Grebenkov, Pulsed-gradient spin-echo monitoring of restricted diffusion in multilayered structures, Journal of Magnetic Resonance 205 (2) (2010) 181–195.
URL http://www.sciencedirect.com/science/article/pii/S1090780710001199

[18] C. Fang, V.-D. Nguyen, D. Wassermann, J.-R. Li, Diffusion MRI simulation of realistic neurons with SpinDoctor and the Neuron Module, NeuroImage 222 (2020) 117198. doi:https://doi.org/10.1016/j.neuroimage.2020.117198.
URL http://www.sciencedirect.com/science/article/pii/S1053811920306844

[19] G. A. Ascoli, D. E. Donohue, M. Halavi, Neuromorpho.org: A central resource for neuronal morphologies, Journal of Neuroscience 27 (35) (2007) 9247–9251. arXiv:http://www.jneurosci.org/content/27/35/9247.full.pdf, doi:10.1523/JNEUROSCI.2055-07.2007.
URL http://www.jneurosci.org/content/27/35/9247

[20] K. K. Watson, T. K. Jones, J. M. Allman, Dendritic architecture of the von economo neurons, Neuroscience 141 (3) (2006) 1107–1112.