SpinDoctor User Guide

Updated November 11, 2019

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 assumes negligible permeabilty);
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).

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

MRI - magnetic resonance imaging

dMRI - diffusion magnetic resonance imaging

ADC - apparent diffusion coefficient

HADC - homogenized ADC

PGSE - pulsed gradient spin echo

OGSE - oscillating gradient

ECS - extra-cellular space

BTPDE - Bloch-Torrey partial differential equation

PDE - partial differential equation

ODE - ordinary differential equation

HARDI - high angular resolution diffusion imaging

STA - short time approximation

FE - finite elements

FEM - finite elements method

2 Installation

The SpinDoctor Toolbox has been tested with MATLAB R2017b. 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 following software is publicly available:

1. The original SpinDoctor Toolbox for simulations of brain white matter geometries is in the *master* branch of

   `https://github.com/jingrebeccali/SpinDoctor`

2. The Neuron Module is in the *NeuronModule* branch:

`https://github.com/jingrebeccali/SpinDoctor/tree/NeuronModule`

3. The Matrix Formalism Module is in the *MatrixFormalismModule* branch:

   `https://github.com/jingrebeccali/SpinDoctor/tree/MatrixFormalismModule`

   Currently, the Matrix Formalism Module allows the computation of the Matrix Formalism signal and the Matrix Formalism Gaussian Approximation signal for realistic neuron (impermeable membranes) with the PGSE sequence. Matrix Formalism for permeable membranes and for general diffusion-encoding sequences are under development and will be released in the future.

The SpinDoctor Toolbox and the Neuron Module have been developed in the MATLAB R2017b and require no additional MATLAB toolboxes. However, the current version of the Matrix Formalism Module requires the MATLAB PDE Toolbox (2017 or later) due to certain difficulties of implementing the matrix eigenvalue solution on a restricted eigenvalue interval. This technical issue will be addressed in a future release.

3 Mathematical background

Suppose the user would like to simulate a geometrical configuration of cells with an optional myelin layer or a nucleus. If spins will be leaving the cells or if the user wants to simulate the extra-cellular space (ECS), then the ECS will enclose the geometrical shapes. Let Ω^e be the ECS, Ω_i^{in} the nucleus (or the axon) and Ω_i^{out} the cytoplasm (or the myelin layer) of the ith cell. We denote the interface between Ω_i^{in} and Ω_i^{out} by Γ_i and the interface between Ω_i^{out} and Ω^e by Σ_i, finally the outside boundary of the ECS by Ψ.

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 $f(t)$, 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:

$$\frac{\partial}{\partial t} M_i^{in}(\boldsymbol{x},t) = -I\gamma f(t)\boldsymbol{g}\cdot\boldsymbol{x}\, M_i^{in}(\boldsymbol{x},t) + \nabla\cdot(\sigma^{in}\nabla M_i^{in}(\boldsymbol{x},t)), \quad \boldsymbol{x}\in\Omega_i^{in}, \tag{1}$$

$$\frac{\partial}{\partial t} M_i^{out}(\boldsymbol{x},t) = -I\gamma f(t)\boldsymbol{g}\cdot\boldsymbol{x}\, M_i^{out}(\boldsymbol{x},t) + \nabla\cdot(\sigma^{out}\nabla M_i^{out}(\boldsymbol{x},t)), \quad \boldsymbol{x}\in\Omega_i^{out}, \tag{2}$$

$$\frac{\partial}{\partial t} M^{e}(\boldsymbol{x},t) = -I\gamma f(t)\boldsymbol{g}\cdot\boldsymbol{x}\, M^{e}(\boldsymbol{x},t) + \nabla\cdot(\sigma^{e}\nabla M^{e}(\boldsymbol{x},t)), \quad \boldsymbol{x}\in\Omega^{e}, \tag{3}$$

where $\gamma = 2.67513\times 10^8\,\mathrm{rad\,s^{-1}T^{-1}}$ is the gyromagnetic ratio of the water proton, I is the imaginary unit, σ^l is the intrinsic diffusion coefficient in the compartment Ω_i^l. 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 $f(t)$. We denote the restriction of the magnetization in Ω_i^{in} by M_i^{in}, and similarly for M_i^{out} and M^e.

Some commonly used time profiles (diffusion-encoding sequences) are:

1. The pulsed-gradient spin echo (PGSE) [5] sequence, with two rectangular pulses of duration δ, separated by a time interval $\Delta-\delta$, for which the profile $f(t)$ is

$$f(t) = \begin{cases} 1, & t_1 \le t \le t_1+\delta, \\ -1, & t_1+\Delta < t \le t_1+\Delta+\delta, \\ 0, & \text{otherwise}, \end{cases} \tag{4}$$

   where t_1 is the starting time of the first gradient pulse with $t_1+\Delta > T_E/2$, T_E is the echo time at which the signal is measured.

2. The oscillating gradient spin echo (OGSE) sequence [6, 7] was introduced to reach short diffusion times. An OGSE sequence usually consists of two oscillating pulses of duration T, each containing n periods, hence the frequency is $\omega = n\frac{2\pi}{T}$, separated by a time interval $\tau - T$. For a cosine OGSE, the profile $f(t)$ is

$$f(t) = \begin{cases} \cos\left(n\frac{2\pi}{T}t\right), & t_1 < t \le t_1 + T, \\ -\cos\left(n\frac{2\pi}{T}(t-\tau)\right), & \tau + t_1 < t \le t_1 + \tau + T, \\ 0, & \text{otherwise}, \end{cases} \tag{5}$$

where $\tau = T_E/2$.

The Bloch-Torrey PDE needs to be supplemented by interface conditions. We recall the interface between Ω_i^{in} and Ω_i^{out} is Γ_i, the interface between Ω_i^{out} and Ω^e is Σ_i, and the outside boundary of the ECS is Ψ. The two interface conditions on Γ_i are the flux continuity and a condition that incorporates a permeability coefficient $\kappa^{in,out}$ across Γ_i: :

$$\begin{aligned} \sigma^{in}\nabla M_i^{in}(\boldsymbol{x},t)\cdot\boldsymbol{n}_i^{in} &= -\sigma^{out}\nabla M_i^{out}(\boldsymbol{x},t)\cdot\boldsymbol{n}_i^{out}, && \boldsymbol{x}\in\Gamma_i, \\ \sigma^{in}\nabla M_i^{in}(\boldsymbol{x},t)\cdot\boldsymbol{n}_i^{in} &= \kappa^{in,out}\Big(M_i^{out}(\boldsymbol{x},t) - M_i^{in}(\boldsymbol{x},t)\Big), && \boldsymbol{x}\in\Gamma_i, \end{aligned}$$

where $\boldsymbol{n}$ is the unit outward pointing normal vector. Similarly, between Ω_i^{out} and Ω^e we have

$$\begin{aligned} \sigma^{out}\nabla M_i^{out}(\boldsymbol{x},t)\cdot\boldsymbol{n}_i^{out} &= -\sigma^{e}\nabla M^{e}(\boldsymbol{x},t)\cdot\boldsymbol{n}^{e}, && \boldsymbol{x}\in\Sigma_i, \\ \sigma^{out}\nabla M_i^{out}(\boldsymbol{x},t)\cdot\boldsymbol{n}_i^{out} &= \kappa^{out,e}\Big(M^{e}(\boldsymbol{x},t) - M_i^{out}(\boldsymbol{x},t)\Big), && \boldsymbol{x}\in\Sigma_i. \end{aligned}$$

Finally, on the outer boundary of the ECS we have

$$0 = \sigma^e\nabla M^e(\boldsymbol{x},t)\cdot\boldsymbol{n}^e, \quad \boldsymbol{x}\in\Psi.$$

The Bloch-Torrey PDE also needs initial conditions:

$$M_i^{in}(\boldsymbol{x},0) = \rho^{in}, \quad M_i^{out}(\boldsymbol{x},0) = \rho^{out}, \quad M^e(\boldsymbol{x},0) = \rho^e.$$

where ρ is the initial spin density.

The dMRI signal is measured at echo time $t = T_E > \Delta + \delta$ for PGSE and $T_E > 2\sigma$ for OGSE. This signal is the integral of $M(\boldsymbol{x}, T_E)$:

$$S := \int_{\boldsymbol{x}\in\bigcup\{\Omega_i^{in},\,\Omega_i^{out},\,\Omega^e\}} M(\boldsymbol{x},T_E)\,d\boldsymbol{x}. \tag{6}$$

In a dMRI experiment, the pulse sequence (time profile $f(t)$) 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 $f(t)$ and is defined as

$$b(\boldsymbol{g}) = \gamma^2\|\boldsymbol{g}\|^2\int_0^{T_E} du\left(\int_0^u f(s)ds\right)^2.$$

For PGSE, the b-value is [5]:

$$b(\boldsymbol{g},\delta,\Delta) = \gamma^2\|\boldsymbol{g}\|^2\delta^2\left(\Delta - \delta/3\right). \tag{7}$$

For the cosine OGSE with *integer* number of periods n in each of the two durations σ, the corresponding b-value is [8]:

$$b(\boldsymbol{g},\sigma) = \gamma^2\|\boldsymbol{g}\|^2\frac{\sigma^3}{4n^2\pi^2} = \gamma^2\|\boldsymbol{g}\|^2\frac{\sigma}{\omega^2}. \tag{8}$$

The reason for these definitions is that in a homogeneous medium, the signal attenuation is $e^{-\sigma b}$, where σ is the intrinsic diffusion coefficient.

3.2 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:

$$ADC := -\frac{\partial}{\partial b}\log\frac{S(b)}{S(0)}\bigg|_{b=0}. \tag{9}$$

We numerically compute ADC by a polynomial fit of

$$\log S(b) = c_0 + c_1 b + \cdots + c_n b^n,$$

increasing n from 1 onwards until we get the value of c_1 to be stable within a numerical tolerance.

3.3 HADC model

In a previous work [9], 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

$$HADC = \sigma - \frac{1}{\int_0^{TE} F(t)^2 dt}\int_0^{TE} F(t)\, h(t)\, dt, \tag{10}$$

where $F(t) = \int_0^t f(s)\, ds$, and

$$h(t) = \frac{1}{|\Omega|}\int_{\partial\Omega}\omega(\boldsymbol{x},t)\,(\boldsymbol{u_g}\cdot\boldsymbol{n})\, ds \tag{11}$$

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:

$$\begin{aligned} \frac{\partial}{\partial t}\omega(\boldsymbol{x},t) - \nabla\left(\sigma\nabla\omega(\boldsymbol{x},t)\right) &= 0, && \boldsymbol{x}\in\Omega,\\ \sigma\nabla\omega(\boldsymbol{x},t)\cdot\boldsymbol{n} &= \sigma F(t)\,\boldsymbol{u_g}\cdot\boldsymbol{n}, && \boldsymbol{x}\in\partial\Omega,\\ \omega(\boldsymbol{x},0) &= 0, && \boldsymbol{x}\in\Omega, \end{aligned} \tag{12}$$

$\boldsymbol{n}$ being the outward normal and $t\in[0,TE]$, $\boldsymbol{u_g}$ is the unit gradient direction. The above set of equations, (10)-(12), comprise the homogenized model that we call the HADC model.

3.4 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) [10, 11]:

$$STA = \sigma\left(1 - \frac{4\sqrt{\sigma}}{3\sqrt{\pi}}\sqrt{\Delta}\frac{A}{dim\, V}\right),$$

where $\dfrac{A}{V}$ is the surface to volume ratio and σ is the intrinsic diffusivity coefficient. In the above formula the pulse duration δ is assumed to be very small compared to Δ. A recent correction to the above formula [9], taking into account the finite pulse duration δ and the gradient direction $\boldsymbol{u_g}$, is the following:

$$STA = \sigma\left[1 - \frac{4\sqrt{\sigma}}{3\sqrt{\pi}}C_{\delta,\Delta}\frac{A_{\boldsymbol{u_g}}}{V}\right], \tag{13}$$

where

$$A_{\boldsymbol{u_g}} = \int_{\partial\Omega}(\boldsymbol{u_g}\cdot\boldsymbol{n})^2\, ds,$$

and

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

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

4 Work flow

Figure 1 is a chart showing the work flow of SpinDoctor.

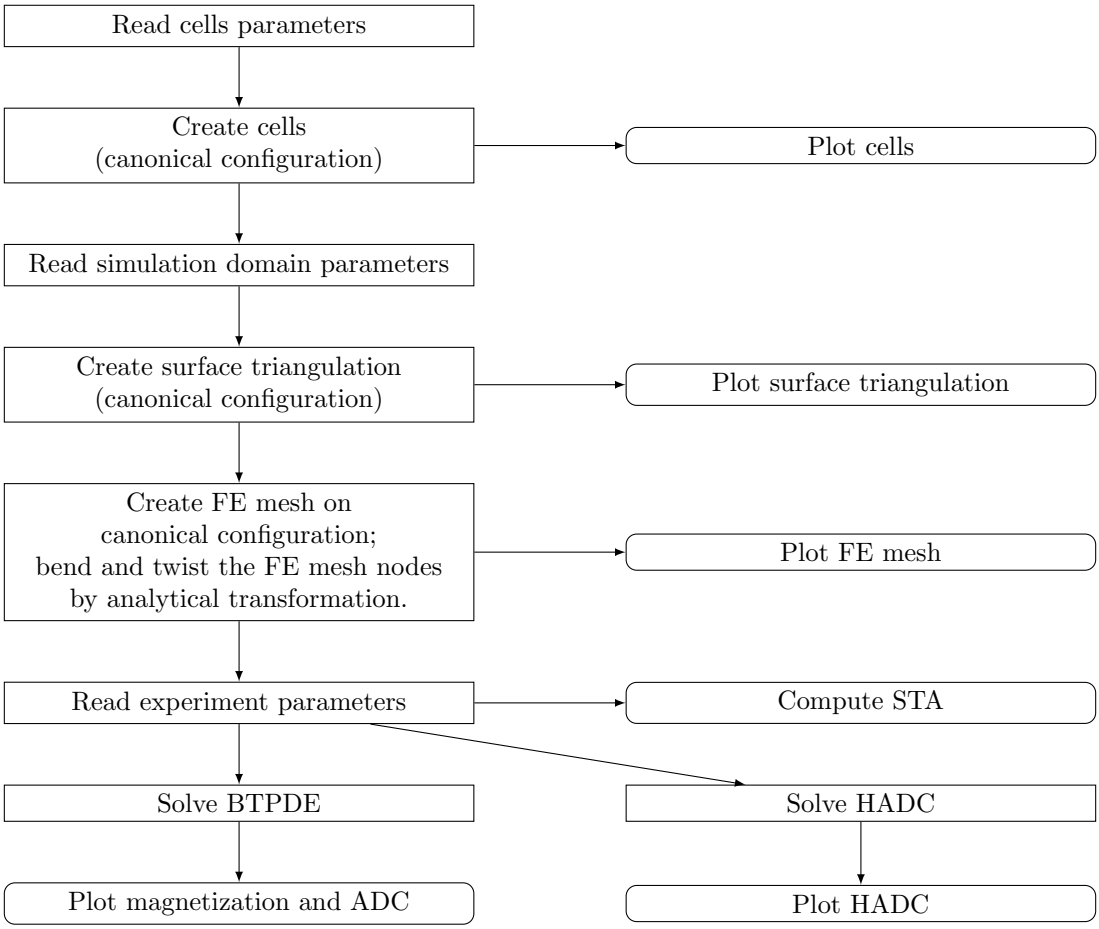

Figure 1: Flow chart describing the work flow of SpinDoctor

4.1 Physical units

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

Parameter	Unit
length	μm
time	μs
diffusion coefficient	$\mu m^2/\mu s$ = mm^2/s
permeability coefficient	$\mu m/\mu s$ = m/s
b-value	$\mu s/\mu m^2$ = s/mm^2
q-value	$(\mu s \mu m)^{-1}$

Table 1: Physical units of the input and output quantities for SpinDoctor.

4.2 User provided input files

There are three input files SpinDoctor expects the user to provide:

1. Input file containing *cells parameters*. Format is given in Table 2;

2. Input file containing *simulation domain parameters*. Format is given in Table 3;

3. Input file containing *simulation experiment parameters*. Format is given in Table 4;

Line	Variable name	Example	Explanation
1	cell_shape	1	1 = spheres; 2 = cylinders;
2	fname_params_cells	'current_cells'	file name to store cells description
3	ncell	10	number of cells
4	Rmin	1.5	min Radius
5	Rmax	2.5	max Radius
6	dmin	1.5	min (%) distance between cells: $dmin \times \frac{(Rmin+Rmax)}{2}$
7	dmax	2.5	max (%) distance between cells $dmax \times \frac{(Rmin+Rmax)}{2}$
8	para_deform	0.05 0.05	$[\alpha \quad \beta]$; α defines the amount of bend; β defines the amount of twist
9	Hcyl	20	height of cylinders

Table 2: Input file containing cells parameters.

Line	Variable name	Example	Explanation
1	Rratio	0.0	if Rratio is outside [0,1], it is set to 0; else $Rratio = \frac{R_{in}}{R_{out}}$;
2	include_ECS	2	0 = no ECS; 1 = box ECS; 2 = tight wrap ECS;
3	ECS_gap	0.3	ECS thickness: a. if box: as percentage of domain length; b. if tight wrap: as percentage of mean radius
4	dcoeff_IN	0.002	diffusion coefficient in IN cmpt: a. nucleus; b. axon (if there is myelin);
5	dcoeff_OUT	0.002	diffusion coefficient in OUT cmpt: a. cytoplasm; b. axon (if there is no myelin);
6	dcoeff_ECS	0.002	diffusion coefficient in ECS cmpt;
7	ic_IN	1	initial spin density in In cmpt: a. nucleus; b. axon (if there is myelin)
8	ic_OUT	1	initial spin density in OUT cmpt: a. cytoplasm; b. axon (if there is no myelin);
9	ic_ECS	1	initial spin density in ECS cmpt:
10	kappa_IN_OUT	1e-5	permeability between IN and OUT cmpts: a. between nucleus and cytoplasm; b. between axon and myelin;
11	kappa_OUT_ECS	1e-5	permeability between OUT and ECS cmpts: a. if no nucleus: between cytoplasm and ECS; b. if no myelin: between axon and ECS;
12	Htetgen	-1	Requested tetgen mesh size; -1 = Use tetgen default;
13	tetgen_cmd	'SRC/TETGEN/ tetGen/win64/ tetgen'	path to tetgen_cmd

Table 3: Input file of simulation domain parameters.

Line	Variable name	Example	Explanation
1	ngdir	20	number of gradient direction; if $ngdir > 1$, the gradient directions are distributed uniformly on a sphere; if $ngdir = 1$, take the gradient direction from the line below;
2	gdir	1.0 0.0 0.0	gradient direction; No need to normalize;
3	nexperi	3	number of experiments;
4	sdeltavec	2500 10000 10000	small delta;
5	bdeltavec	2500 10000 10000	big delta;
6	seqvec	1 2 3	diffusion sequence of experiment; 1 = PGSE; 2 = OGSEsin; 3 = OGSEcos;
7	npervec	0 10 10	number of period of OGSE;
8	solve_hadc	1	0 = do not solve HADC; Otherwise solve HADC;
9	rtol_deff, atol_deff	1e-4 1e-4	$[r_{tol} \quad a_{tol}]$; relative and absolute tolerance for HADC ODE solver;
10	solve_btpde	1	0 = do not solve BTPDE; Otherwise solve BTPDE;
11	rtol_bt, atol_bt	1e-5 1e-5	$[r_{tol} \quad a_{tol}]$; relative and absolute tolerance for BTPDE ODE solver;
12	nb	2	number of b-values;
13	blimit	0	0 = specify bvec; 1 = specify [bmin,bmax]; 2 = specify [gmin,gmax];
14	const_q	0	0: use input bvalues for all experiments; 1: take input bvalues for the first experiment and use the same q for the remaining experiments
15	bvalues	0 50 100 200	bvalues or [bmin, bmax] or [gmin, gmax]; depending on line 13;

Table 4: Input file for simulation experiment parameters.

4.3 Important output quantities

In Table 17 we list some useful quantities that are the outputs of SpinDoctor. The braces in the "Size" column denote MATLAB cell data structure and the brackets denote MATLAB matrix data structure. We note that if the finite elements mesh is very big, the user should not output the magnetization after solving the BTPDE, rather, only the space integral of the magnetization which is the dMRI signal.

Variable name	Size	Explanation
SOL	{nexperi×nb×Ncmpt}[Nnodes]	Solution of the PDE (magnetization) . This variable should not be outputted from the solution of the BTPDE if the finite elements mesh is too large.
SIG_cmpts	[Ncmpt × nexperi × nb]	Signal (integral of magnetization) at TE in each compartment.
SIG_allcmpts	[nexperi × nb]	Signal (integral of magnetization) at TE summed over all compartments.
SIG_cmpts_hardi	[ngdir × Ncmpt × nexperi × nb]	Signal in each compartment in each direction.
SIG_allcmpts_hardi	[ngdir × nexperi × nb]	Signal accounting for all compartments in each direction.
ADC_cmpts	[Ncmpt × nexperi]	ADC in each compartment.
ADC_allcmpts	[nexperi]	ADC accounting for all compartments.
ADC_cmpts_hardi	[ngdir × Ncmpt × nexperi]	ADC in each compartment in each direction.
ADC_allcmpts_hardi	[ngdir × nexperi]	ADC accounting for all compartments in each direction.

Table 5: Some important SpinDoctor output quantities.

4.4 Important functions

In Table 6 we list some important functions of SpinDoctor. For detailed information about them, including argument lists, please read the online documentation.

Function name	Purpose
create_geom	Reads the params_cells input file and creates the geometrical configuration.
create_femesh_fromcells	Creates a finite elements mesh.
read_params_simul_domain	Reads the params_simul_domain input file.
read_params_simul_experi	Reads the params_simul_experi input file.
read_tetgen	Reads the finite elements mesh.
PREPARE_PDE	Set up the PDE model in the geometrical compartments.
GET_VOL_SA	Gets the volume and the surface area quantities from the finite elements mesh.
BTPDE	Computes the BTPDE signal in one diffusion-encoding direction.
HADC	Computes the HADC in one diffusion-encoding direction.
STA	Computes the short time approximation in one diffusion-encoding direction.
ADCFREE	Computes the free diffusion ADC.
FIT_SIGNAL	Fits the S_0 and the ADC of the signal.
HARDI_PTS	Provides gradient directions uniformly distributed in unit 3D sphere.
SIG_BTPDE_HARDI	Computes the BTPDE signal in multiple diffusion-encoding directions.
HADC_HARDI	Computes the HADC in multiple diffusion-encoding directions.
PLOT_FEMESH	Displays the finite elements mesh.
PLOT_SIGNAL	Displays the simulated signal in one diffusion-encoding direction.
PLOT_ADC	Displays the simulated ADC in one diffusion-encoding direction.
PLOT_PDESOLUTION	Displays 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_PT	Displays simulation results in multiple diffusion-encoding directions.

Table 6: Some important functions of SpinDoctor.

5 Software components and algorithms

Below we discuss the various components of SpinDoctor in more detail.

5.1 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, $dist$, between the current center and previously accepted cells.
Find the intersection of $[dist - dmax \times R_{mean}, dist - dmin \times R_{mean}]$ and $[Rmin, Rmax]$, where $R_{mean} = \frac{Rmin+Rmax}{2}$. 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.

5.2 Plot cells

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

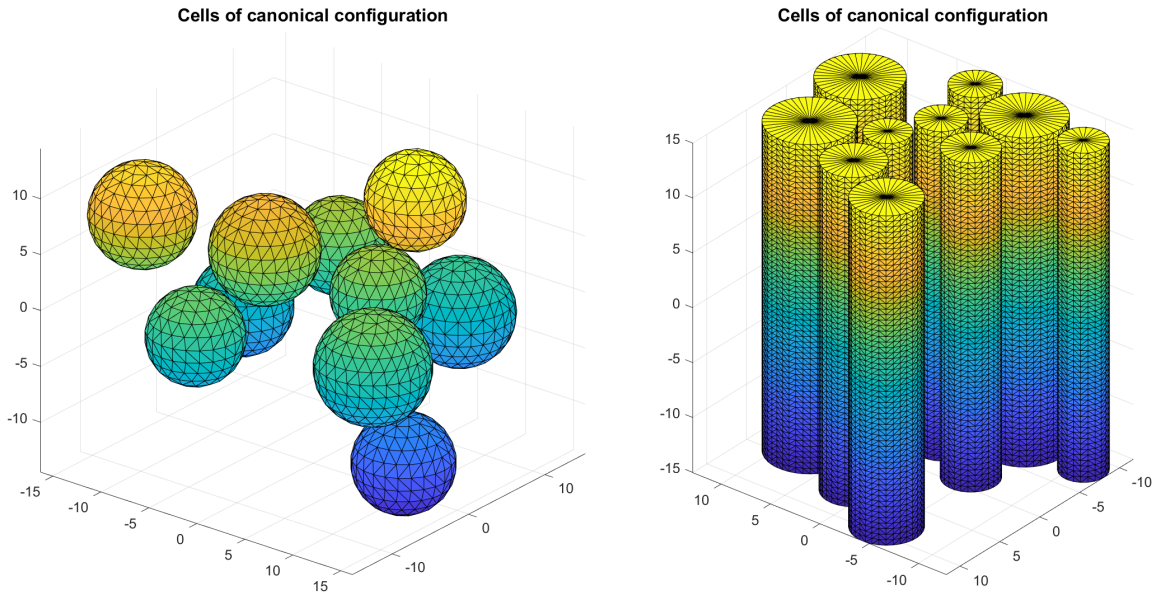

Figure 2: SpinDoctor plots cells in the canonical configuration.

5.3 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.

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 $r_1, \cdots, r_{ncell}$ and the radii of the cells $R_1, \cdots, R_{ncell}$. Then the boundaries between the cytoplasm and the nucleus are

$$\{\Gamma_i = T(c_i, r_i)\}, i = 1, \cdots, ncell;$$

and between the cytoplasm and the ECS

$$\{\Sigma_i = T(c_i, R_i)\}, i = 1, \cdots, ncell;$$

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

$$\bigcup_i S(c_i, R_i) \in [x_0, x_f] \times [y_0, y_f] \times [z_0, z_f]$$

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

$$W = \bigcup_i S(c_i, R_i + \text{ECS\_gap} \times R_{mean}),$$

where $R_{mean} = \frac{Rmin+Rmax}{2}$. 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 $r_1, \cdots, r_{ncell}$ and the radii of the cells be $R_1, \cdots, R_{ncell}$, 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

$$W = \bigcup_i D(c_i, R_i + kR_{mean}).$$

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 $Q - \bigcup_i D_i - \bigcup_i A_i$ 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 $D(c_i, r_i)$ are straightforward.

5.4 Plot surface triangulation

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

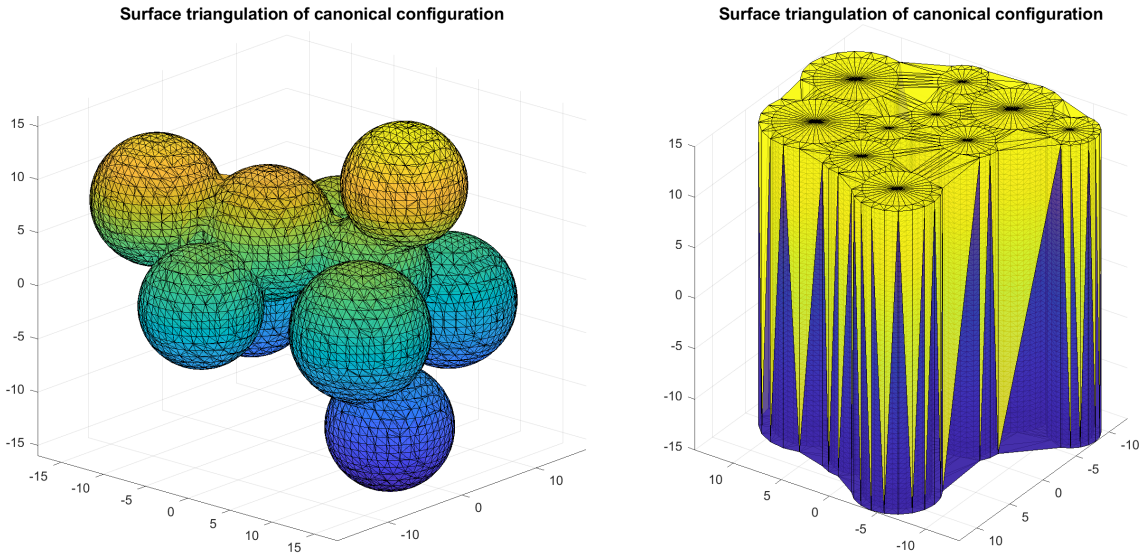

Figure 3: SpinDoctor plots the surface triangulation of the canonical configuration. Left: spherical cells with ECS; Right: cylindrical cells with ECS.

5.5 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 7 and 8. 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.

Spherical cells without nucleus

Cmpt	Cytoplasm	Nucleus	ECS
Label	OUT		ECS
Number	$[1 : n_{cell}]$		$n_{cell} + 1$

Spherical cells with nucleus

Cmpt	Cytoplasm	Nucleus	ECS
Label	OUT	IN	ECS
Number	$[1 : n_{cell}]$	$[n_{cell} + 1 : 2n_{cell}]$	$2n_{cell} + 1$

Cylindrical cells without myelin

Cmpt	Axon	Myelin	ECS
Label	OUT		ECS
Number	$[1 : n_{cell}]$		$n_{cell} + 1$

Cylindrical cells with myelin

Cmpt	Axon	Myelin	ECS
Label	IN	OUT	ECS
Number	$[1 : n_{cell}]$	$[n_{cell} + 1 : 2n_{cell}]$	$2n_{cell} + 1$

Table 7: The labels and numbers of compartments.

Spherical cells without nucleus

Boundary	Sphere		Outer ECS boundary
Label	OUT_ECS		$\kappa = 0$
Number	$1 : n_{cell}$		$n_{cell} + 1$

Spherical cells with nucleus

Boundary	Outer sphere	Inner sphere	Outer ECS boundary
Label	OUT_ECS	IN_OUT	$\kappa = 0$
Number	$1 : n_{cell}$	$n_{cell} + 1 : 2n_{cell}$	$2n_{cell} + 1$

Cylindrical cells without myelin

Boundary	Cylinder side wall	Cylinder top and bottom	Outer ECS boundary minus cylinder top/bottom
Label	OUT_ECS	$\kappa = 0$	$\kappa = 0$
Number	$2[1 : n_{cell}] - 1$	$2[1 : n_{cell}]$	$2n_{cell} + 1$

Cylindrical cells with myelin

Boundary	Inner cylinder side wall	Inner cylinder top and bottom	
Label	IN_OUT	$\kappa = 0$	
Number	$4[1 : n_{cell}] - 3$	$4[1 : n_{cell}] - 2$	
	Outer cylinder side wall	Outer cylinder top and bottom	Outer ECS boundary minus cylinder top/bottom
Label	OUT_ECS	$\kappa = 0$	$\kappa = 0$
Number	$4[1 : n_{cell}] - 1$	$4[1 : n_{cell}]$	$4n_{cell} + 1$

Table 8: 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

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

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

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

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

5.6 Plot FE mesh

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

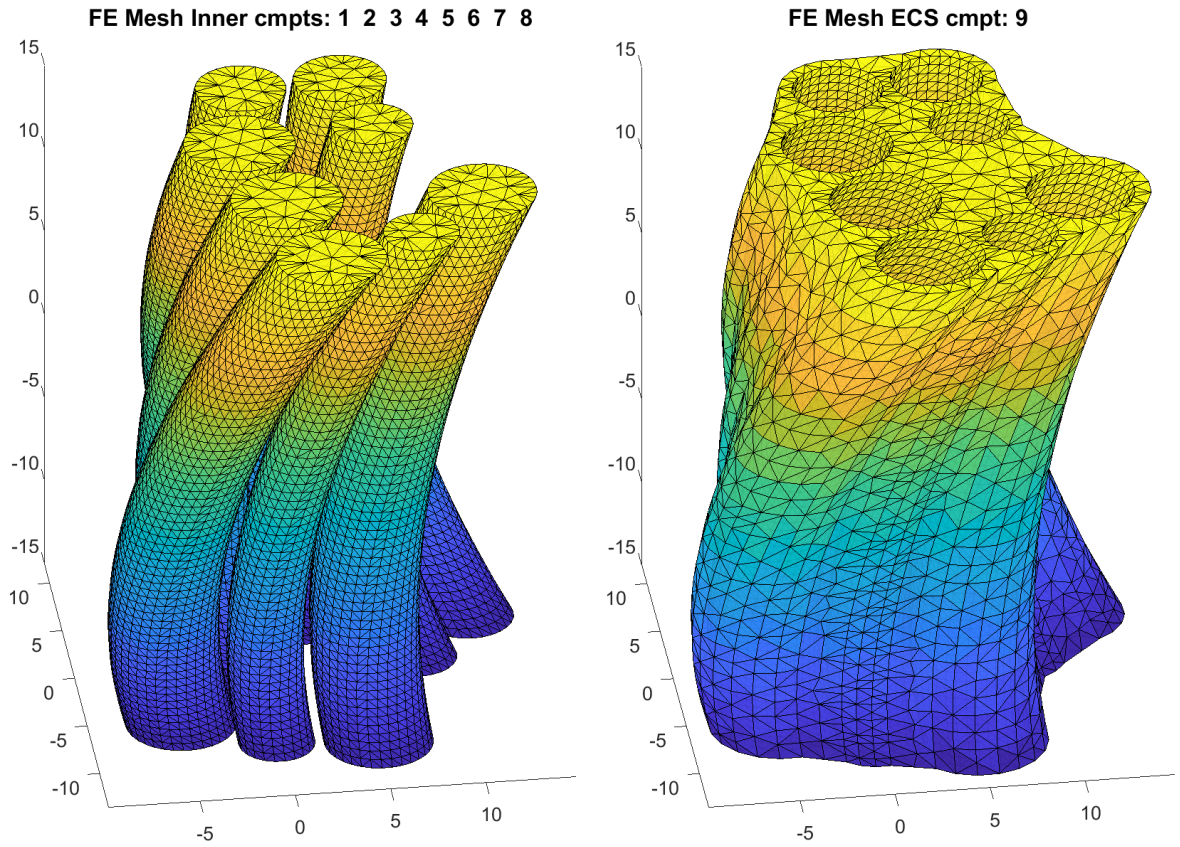

Figure 4: FE mesh of cylinders and ECS after bending and twisting. Compartment number is 1 to 8 for the cylinders and 9 for the ECS.

5.7 BTPDE

The spatial discretization of the Bloch-Torrey PDE is based on a finite element method where *interface (ghost) elements* [12] are used to impose the permeable interface conditions. The time stepping is done using the MATLAB built-in ODE routine ode23t. See Algorithm 5.

Algorithm 5: BTPDE.

FE matrices are generated for each compartment by the finite element method with continuous piecewise linear basis functions (known as P_1). The basis functions are denoted as φ_k for $k = 1, \ldots, N_v$, where N_v denotes the number of mesh nodes (vertices). All matrices are sparse matrices. $\boldsymbol{M}$ and $\boldsymbol{S}$ are known in the FEM literature as mass and stiffness matrices which are defined as follows:

$$\boldsymbol{M}_{ij} = \int_\Omega \varphi_i \varphi_j \, d\boldsymbol{x}, \qquad \boldsymbol{S}_{ij} = \int_\Omega \sigma_i \, \nabla\varphi_i \cdot \nabla\varphi_j \, d\boldsymbol{x}.$$

$\boldsymbol{J}$ has a similar form as the mass matrix but it is scaled with the coefficient $\boldsymbol{g} \cdot \boldsymbol{x}$, we therefore call it the scaled-mass matrix

$$\boldsymbol{J}_{ij} = \int_\Omega \boldsymbol{g} \cdot \boldsymbol{x} \, \varphi_i \varphi_j \, d\boldsymbol{x}.$$

We construct the matrix based on the flux matrix $\boldsymbol{Q}$

$$\boldsymbol{Q}_{ij} = \int_{\partial\Omega} w \, \varphi_i \varphi_j \, ds$$

where a scalar function w is used as an interface marker. The 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 elements matrices in the assembly of $\boldsymbol{S}, \boldsymbol{M}, \boldsymbol{J}$ are evaluated at once and stored in a full matrix of size $4 \times 4 \times N_e$, where N_e denotes the number of tetrahedral elements. The assembly of $\boldsymbol{Q}$ is even simpler; all local matrices are stored in a full matrix of size $3 \times 3 \times n_{be}$, where n_{be} denotes the number of boundary triangles.

Double nodes are placed at the interfaces between compartments connected by permeable membrane. $\overline{\boldsymbol{Q}}$ is used to impose the interface conditions and it is associated with the *interface (ghost) elements*. Specifically, assume that the double nodes are defined in a pair of indices $\{i, \bar{i}\}$, $\overline{\boldsymbol{Q}}$ is defined as the following

$$\overline{\boldsymbol{Q}}_{ij} = \begin{cases} \boldsymbol{Q}_{ij}, & \text{if vertex } i \text{ and } j \text{ belong to one interface} \\ -\boldsymbol{Q}_{\bar{i}\bar{j}} & \text{if vertex } i \text{ and } j \text{ belong to two different interfaces} \end{cases}$$

The fully coupled linear system has the following form

$$\boldsymbol{M} \frac{\partial \xi}{\partial t} = -\Big(I\gamma f(t) \, \boldsymbol{J} + \boldsymbol{S} + \overline{\boldsymbol{Q}} \Big) \, \xi \tag{14}$$

where ξ is the approximation of the magnetization M. SpinDoctor calls MATLAB built-in ODE routine ode23t to solve the semi-discretized system of equations.

5.8 HADC model

Similarly, the diffusion equation of the HADC model is discretized by finite elements. See Algorithm 6.

Algorithm 6: HADC model.

Eq. (12) can be discretized similarly as described for the Bloch-Torrey PDE and has the matrix form

$$\boldsymbol{M} \frac{\partial \zeta}{\partial t} = -\boldsymbol{S} \, \zeta + \boldsymbol{Q} \, \bar{\zeta} \tag{15}$$

where ζ is the approximation of w and $\bar{\zeta}_i = \sigma_i \, F(t) \, \boldsymbol{u}_g \cdot \boldsymbol{n}(\boldsymbol{x}_i)$. We note that the matrices here are assembled and solved separately for each compartment. SpinDoctor calls MATLAB built-in ODE routine ode23t to solve the semi-discretized equation.

5.9 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 choose the following options:

1. 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;
2. the ADCs are plotted as a surface plot. The dots are the end points of the vectors in the diffusion-encoding direction multiplied the ADC. The color gives the magnitude of the ADC.

The user can also choose to plot the magnetization solution at the surface nodes of the finite elements mesh.

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. 5 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.

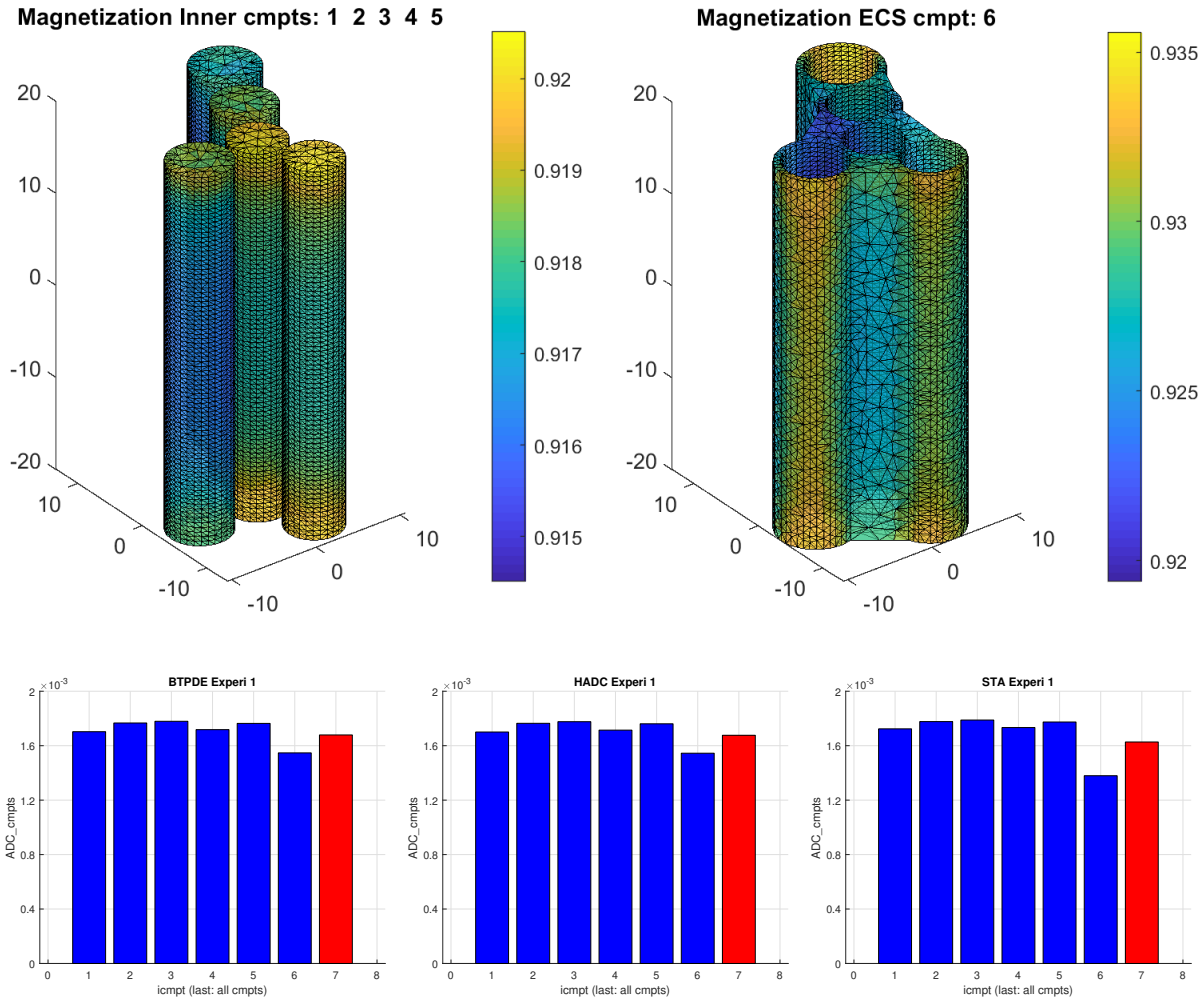

Figure 5: Geometry: 5 cylinders, tight wrap ECS, ECS gap = 0.2, $\boldsymbol{u_g} = [1, 1, 1]$, $\sigma^{out} = \sigma^{ecs} = 2 \times 10^{-3}\,\mathrm{mm}^2/\mathrm{s}$, $\kappa = 0\,\mathrm{m/s}$, OGSE cosine ($\delta = 14\mathrm{ms}, \Delta = 14\mathrm{ms}$, number of periods = 6). The vertical bars 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.

6.2 Permeable membranes

In Fig. 6 we show the effect of permeability: the Bloch-Torrey PDE model includes permeable membranes ($\kappa = 1 \times 10^{-3}\,\mathrm{m/s}$) 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 TE, which is just the integral of the solution of the Bloch-Torrey PDE in that compartment.

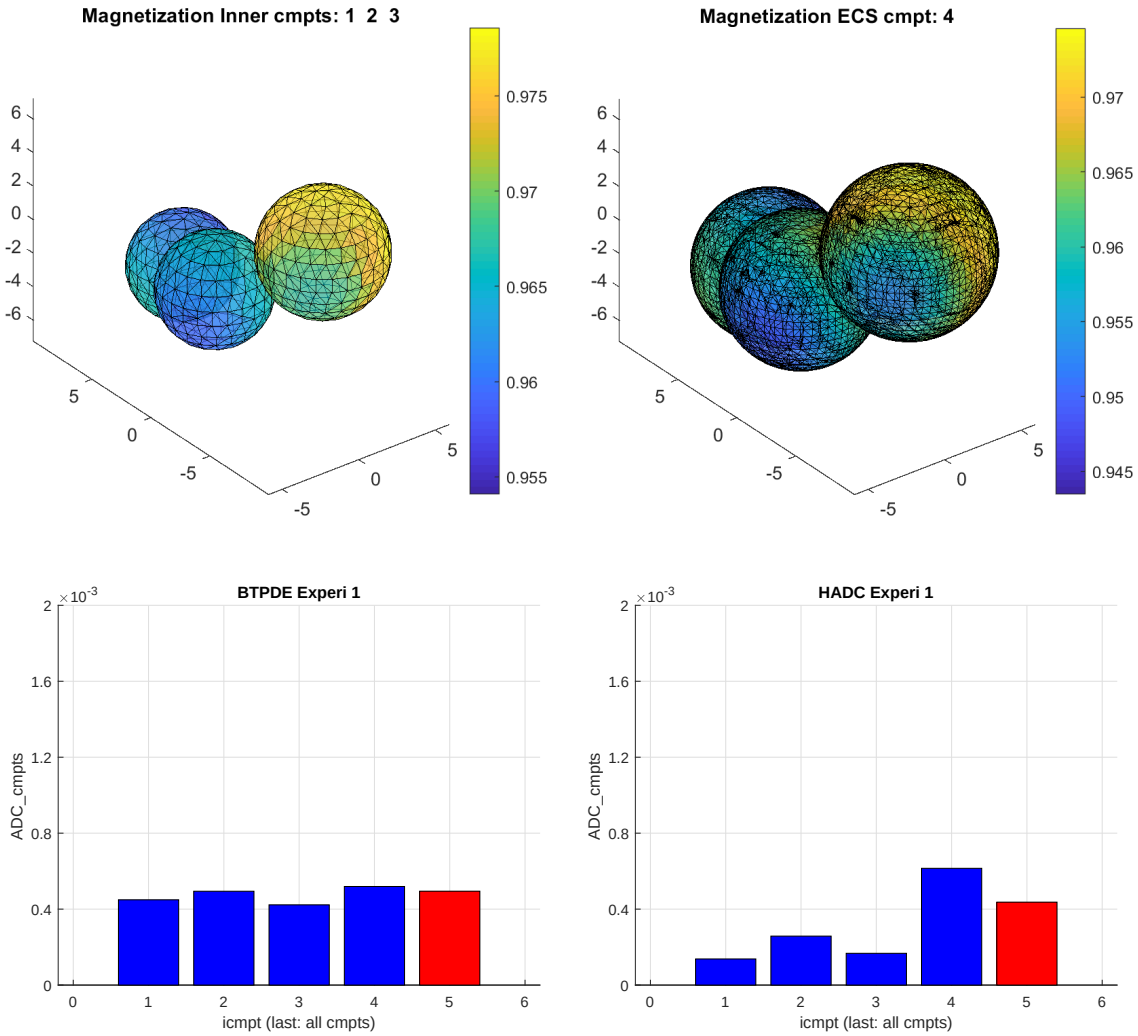

Figure 6: Geometry: 3 spheres, tight wrap ECS, ECS gap = 0.3, $\boldsymbol{u_g} = [1, 1, 0]$, $\sigma^{in} = \sigma^{ecs} = 2 \times 10^{-3}\,\mathrm{mm}^2/\mathrm{s}$, $\kappa = 1 \times 10^{-3}\,\mathrm{m/s}$ (left), $\kappa = 0\,\mathrm{m/s}$ (right). PGSE ($\delta = 5\mathrm{ms}, \Delta = 5\mathrm{ms}$). 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.

6.3 Myelin layer

In Fig. 7 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.

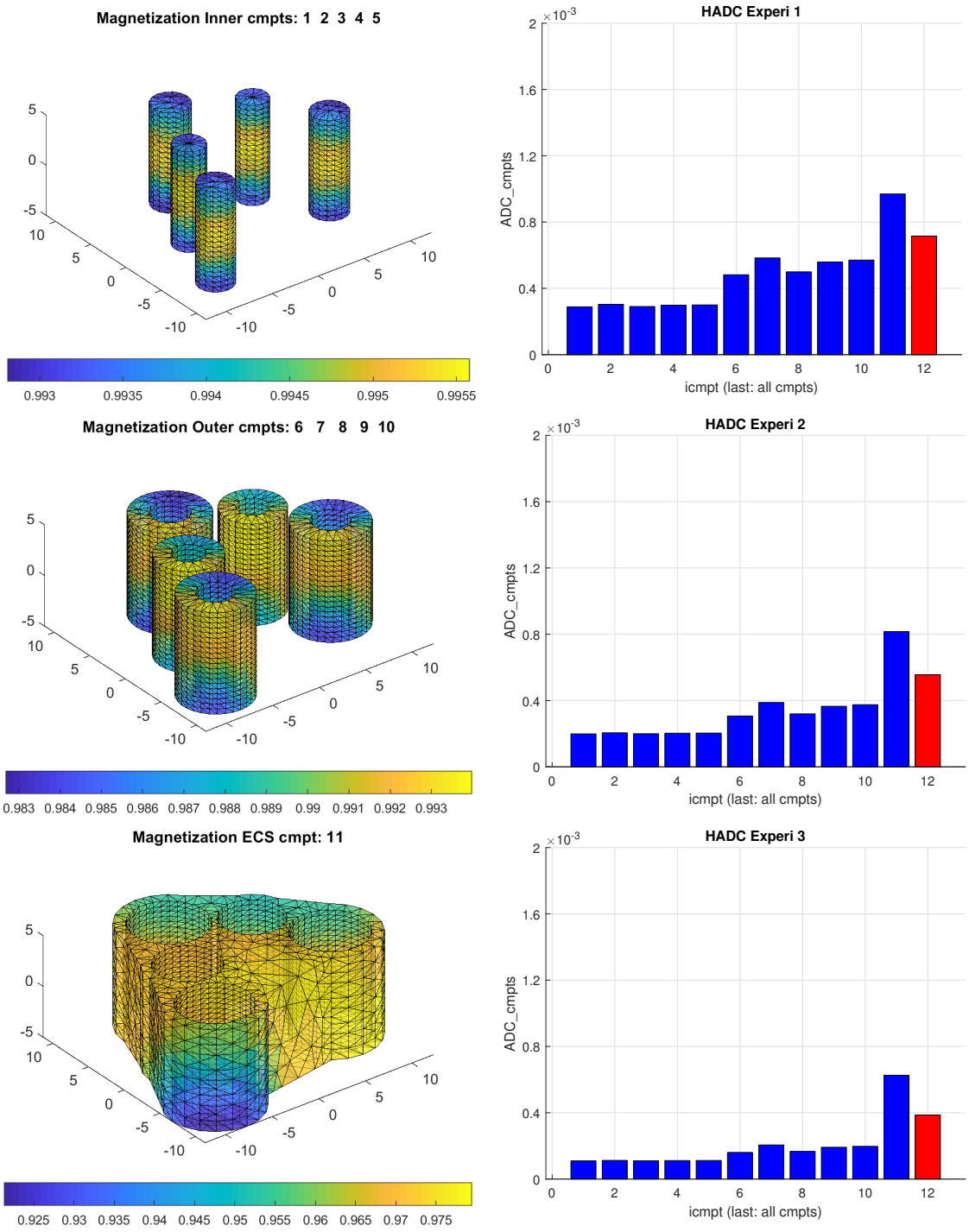

Figure 7: Geometry: 5 cylinders, myelin layer, $R_{in}/R_{out} = 0.5$, tight wrap ECS, ECS gap = 0.3, $\kappa = 0\,\mathrm{m/s}$, $\boldsymbol{u_g} = [1, 1, 1]$, $\sigma^{in} = \sigma^{out} = \sigma^{ecs} = 2 \times 10^{-3}\,\mathrm{mm^2/s}$, 3 experiments: PGSE ($\delta = 5\mathrm{ms}, \Delta = 5, 10, 20\mathrm{ms}$). Left: the magnetization at $\Delta = 5\mathrm{ms}$. 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.

6.4 Twisting and bending

In Fig. 8 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.

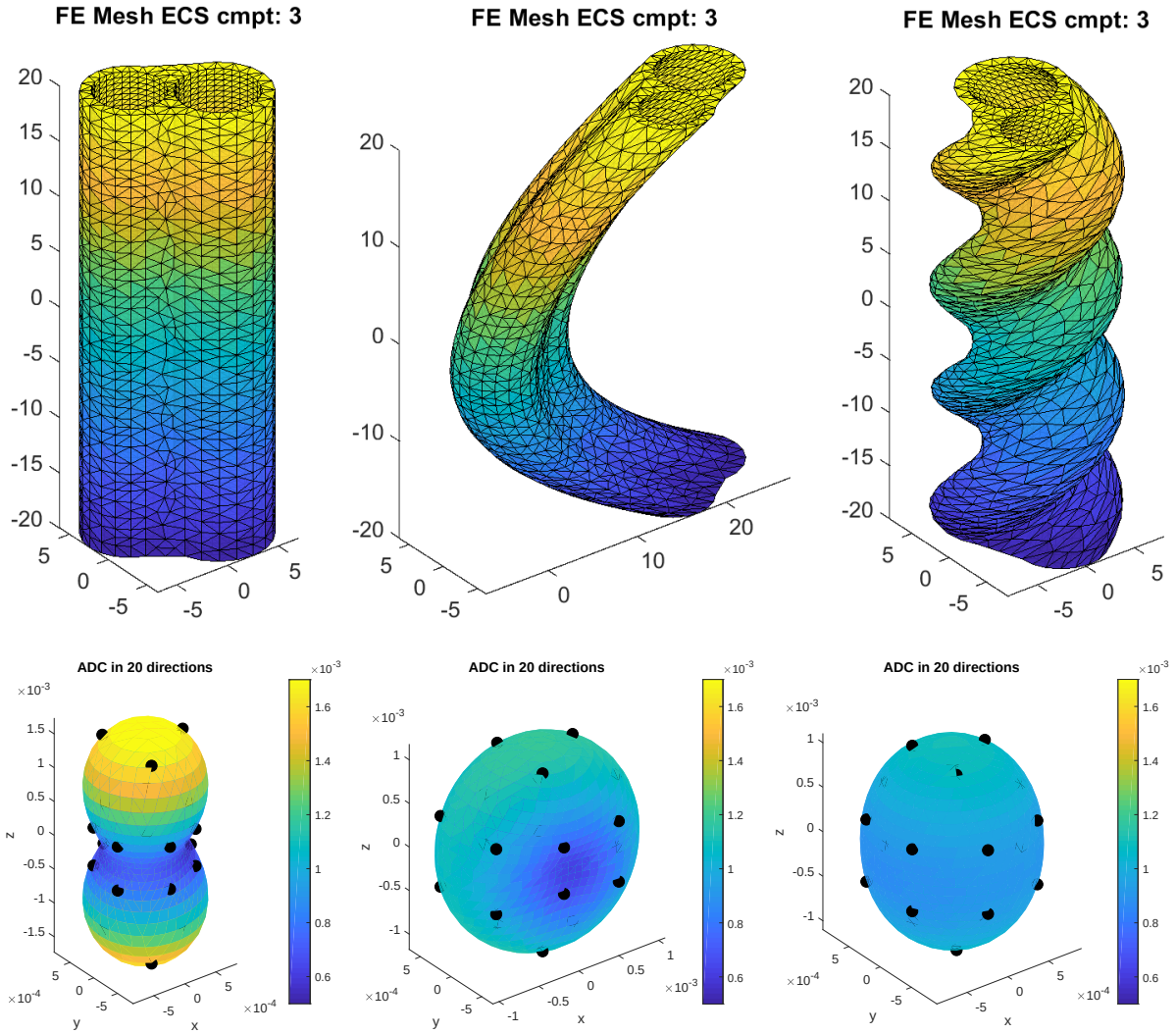

Figure 8: Geometry: 2 cylinders, no myelin layer, tight wrap ECS, ECS gap = 0.3, $\kappa = 0\,\mathrm{m/s}$, $\sigma^{out} = \sigma^{ecs} = 2 \times 10^{-3}\,\mathrm{mm^2/s}$, PGSE ($\delta = 2.5\mathrm{ms}, \Delta = 5\mathrm{ms}$).
Left: canonical configuration. Middle: bend parameter = 0.05. Right: 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.

7 Neuron Module

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*. 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 attributable to spins inside neurons.

7.1 Work flow

The Neuron Module follows the same workflow as SpinDoctor and builds upon the functionalities of SpinDoctor. The Neuron Module uses the same set of three input files as SpinDoctor. SpinDoctor allows the easy construction of multiple compartment models of the brain white matter, with the possibility of coupling water diffusion between the geometrical compartments by permeable membranes. 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 file 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.

7.2 User provided input files

In SpinDoctor, there are three input files in which the user specifies the parameters of the desired simulations. They are :

1. *params_cells.in*: contains the cells parameters
2. *params_simul_domain.in*: contains the simulation domain parameters
3. *params_simul_experi.in*: contains the simulation experiment parameters

We list the input files the user must provide in order to use the Neuron Module, noting where relevant, the input parameters that are not applicable (marked by "na") to the current version of the Neuron Module.

7.2.1 Read cells parameters

The user provides an input file for the cells parameters, in the format described in Table 14. To simulate the diffusion MRI signal of a neuron, the user chooses option 3 for the cell shape. The user specifies the name of the neuron to be simulated in line 2.

Line	Variable name	Example	Explanation
1	cell_shape	3	1 = spheres; 2 = cylinders; 3 = neuron;
2	fname	'msh_files/pyramidal/ 02b_pyramidal1aACC'	file name of neuron mesh
3	ncell	1	number of cells
4	Rmin	na	min Radius
5	Rmax	na	max Radius
6	dmin	na	min (%) distance between cells
7	dmax	na	max (%) distance between cells
8	para_deform	na	$[\alpha \quad \beta]$; α defines the amount of bend; β defines the amount of twist
9	Hcyl	na	height of cylinders

Table 9: Input file containing cells parameters. "na" means not applicable to neuron simulation.

7.2.2 Read simulation domain parameters

The user provides an input file for the simulation domain parameters, in the format described in Table 15.

Line	Variable name	Example	Explanation
1	Rratio	na	
2	include_ECS	na	
3	ECS_gap	na	
4	dcoeff_IN	na	
5	dcoeff_OUT	0.002	diffusion coefficient in OUT cmpt
6	dcoeff_ECS	na	
7	ic_IN	na	
8	ic_OUT	1	initial spin density in OUT cmpt
9	ic_ECS	na	
10	kappa_IN_OUT	na	
11	kappa_OUT_ECS	na	
12	Htetgen	-1	Requested tetgen mesh size; -1 = Use tetgen default;
13	tetgen_cmd	'SRC/TETGEN/ tetGen/win64/ tetgen'	path to tetgen_cmd

Table 10: Input file of simulation domain parameters. "na" means not applicable to neuron simulation.

7.2.3 Read simulation experiment parameters

The user provides an input file for the simulation experiment parameters, in the format described in Table 16.

Line	Variable name	Example	Explanation
1	ngdir	300	number of gradient direction; if $ngdir > 1$, the gradient directions are distributed uniformly on a sphere; if $ngdir = 1$, take the gradient direction from the line below;
2	gdir	1.0 0.0 0.0	gradient direction; No need to normalize;
3	nexperi	1	number of sequences to simulate;
4	sdeltavec	2500 10000	small delta;
5	bdeltavec	5000 43000	big delta;
6	seqvec	1 1 1	diffusion sequence of experiment; 1 = PGSE; 2 = OGSEsin; 3 = OGSEcos;
7	npervec	0 0 0	number of period of OGSE;
8	solve_hadc	na	0 = do not solve HADC; Otherwise solve HADC;
9	rtol_deff, atol_deff	na	$[r_{tol} \quad a_{tol}]$; relative and absolute tolerance for HADC ODE solver;
10	solve_btpde	1	0: do not solve BTPDE; Otherwise solve BTPDE;
11	rtol_bt, atol_bt	1e-2 1e-4	$[r_{tol} \quad a_{tol}]$; relative and absolute tolerance for BTPDE ODE solver;
12	nb	4	number of b-values;
13	blimit	0	0=specify bvec; 1=specify[bmin,bmax]; 2 = specify[gmin,gmax];
14	const_q	0	0: use input bvalues for all experiments; 1: take input bvalues for the first experiment and use the same q for the remaining experiments
15	b-values	0 1000 2000 3000	b-values or [bmin, bmax] or [gmin, gmax]; depending on line 13;

Table 11: Input file for simulation experiment parameters.

7.3 Neuron Module examples

Below we display figures from the functions PLOT_FEMESH, PLOT_PDESOLUTION, PLOT_SIGNAL, PLOT_HARDI_PTS. The geometrical configuration is the spindle neuron *03b_spindle6aFI*. The intrinsic diffusion coefficient is set to $\mathcal{D}_0 = 2 \times 10^{-3}\,\mathrm{mm}^2/\mathrm{s}$, with nonpermeable membrane. We simulated 2 diffusion-encoding sequences:
Experiment 1: f_1 is PGSE ($\delta = 10.6\mathrm{ms}, \Delta = 13\mathrm{ms}$),
Experiment 2: f_2 is PGSE ($\delta = 10.6\mathrm{ms}, \Delta = 73\mathrm{ms}$).

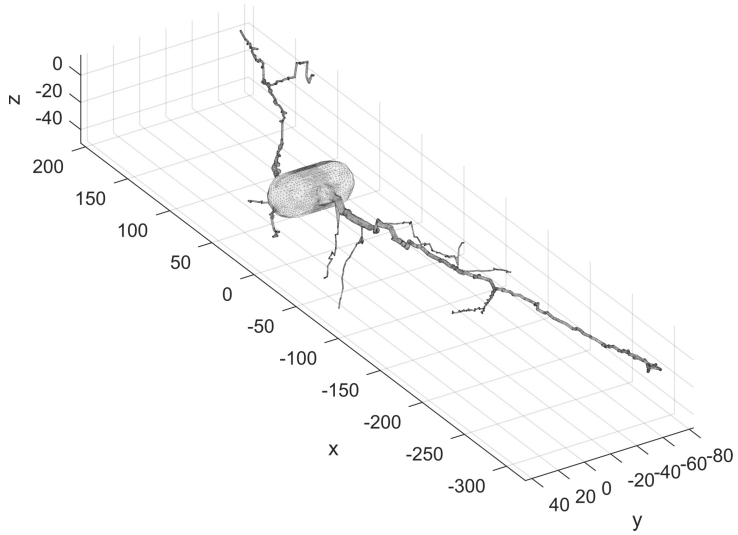

Figure 9: The finite elements mesh of the neuron *03b_spindle6aFI*. The unit is μm. (Using the command PLOT_FEMESH)

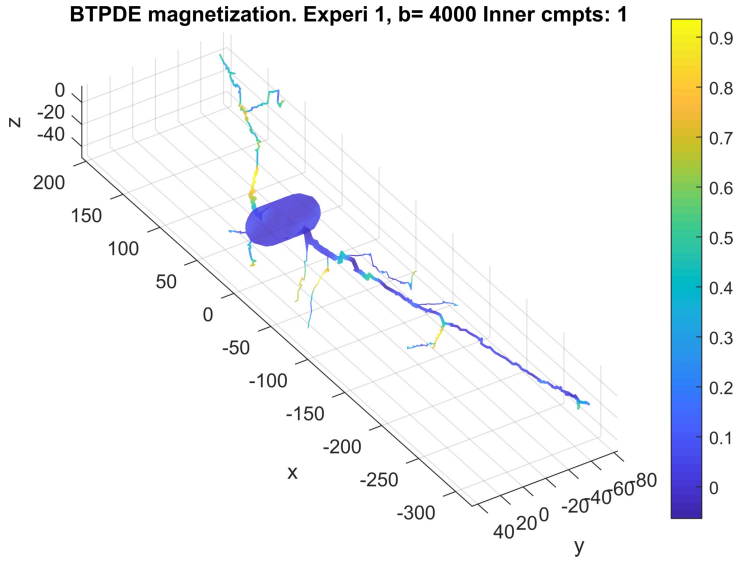

Figure 10: The PDE solution (magnetization) on the neuron *03b_spindle6aFI* in the diffusion-encoding direction (1, 1, 1). The diffusion-encoding sequence is PGSE ($\delta = 10.6$ms, $\Delta = 13$ms). The b-value is $b = 4000\,\mathrm{s/mm}^2$. (Using the command PLOT_PDESOLUTION)

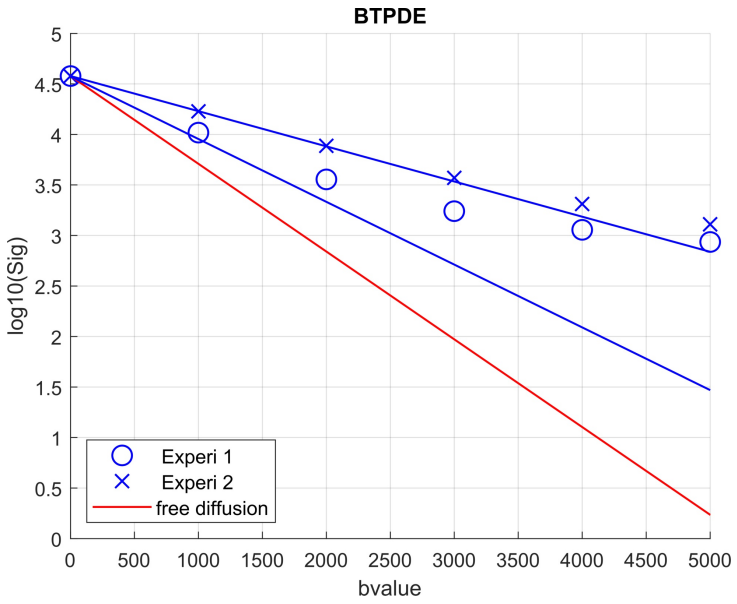

Figure 11: The (non-normalized) simulated signal of the neuron *03b_spindle6aFI* in the diffusion-encoding direction (1, 1, 1). For Experi 1, the diffusion-encoding sequence is PGSE ($\delta = 10.6$ms, $\Delta = 13$ms). For Experi 2, the diffusion-encoding sequence is PGSE ($\delta = 10.6$ms, $\Delta = 73$ms). (Using the command PLOT_SIGNAL)

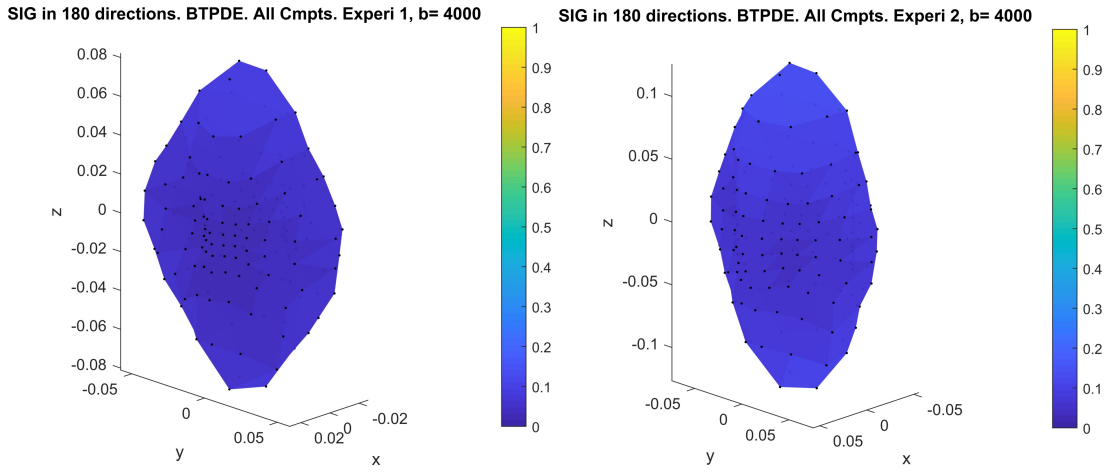

Figure 12: The simulated signal in 180 diffusion-encoding directions for the neuron *03b_spindle6aFI*. Left: The signal in 180 directions with PGSE ($\delta = 10.6$ms, $\Delta = 13$ms) and b=4000 s/mm^2. Right: The signal in 180 directions with PGSE ($\delta = 10.6$ms, $\Delta = 73$ms) and b=4000 s/mm^2. (Using the command PLOT_HARDI_PTS)

7.4 Finite elements meshes of neurons

Table 12 lists the names and the finite element mesh sizes of the group of 36 pyramidal neurons and the group of 29 spindle neurons. Table 13 shows the morphological characteristics for the neurons. The neuron models and the measurement data are from [13] and [14].

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 12: Names and sizes of all the neuron finite elements meshes generated by Tetgen with default settings ($Htetgen = -1$). The number of FE elements (not shown) is approximately four times the number of FE nodes.

Neuron ID	Brain region	Average diameter (μm)	Overall height (μm)	Soma volume (μm^3)	Total volume (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
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 13: The morphological characteristics of the neurons.

As an illustration, in Figure 13 we show the finite elements meshes of 4 spindle neurons and in Figure 14 we show the finite elements meshes of 4 pyramidal neurons.

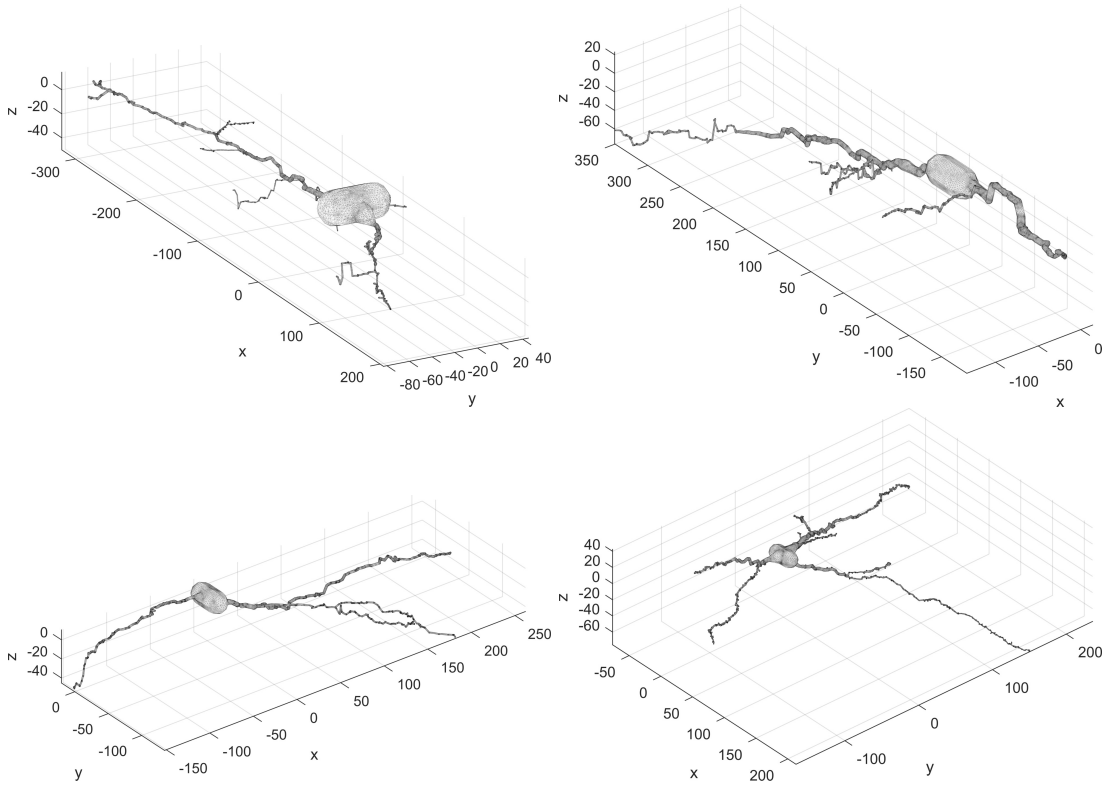

Figure 13: The finite elements meshes of four spindle neurons. The unit is μm. Top left: *03a_spindle6aFI*. Top right: *28o_spindle20aFI*. Bottom left: *09o_spindle8aFI*. Bottom right: *25o_spindle17aFI*.

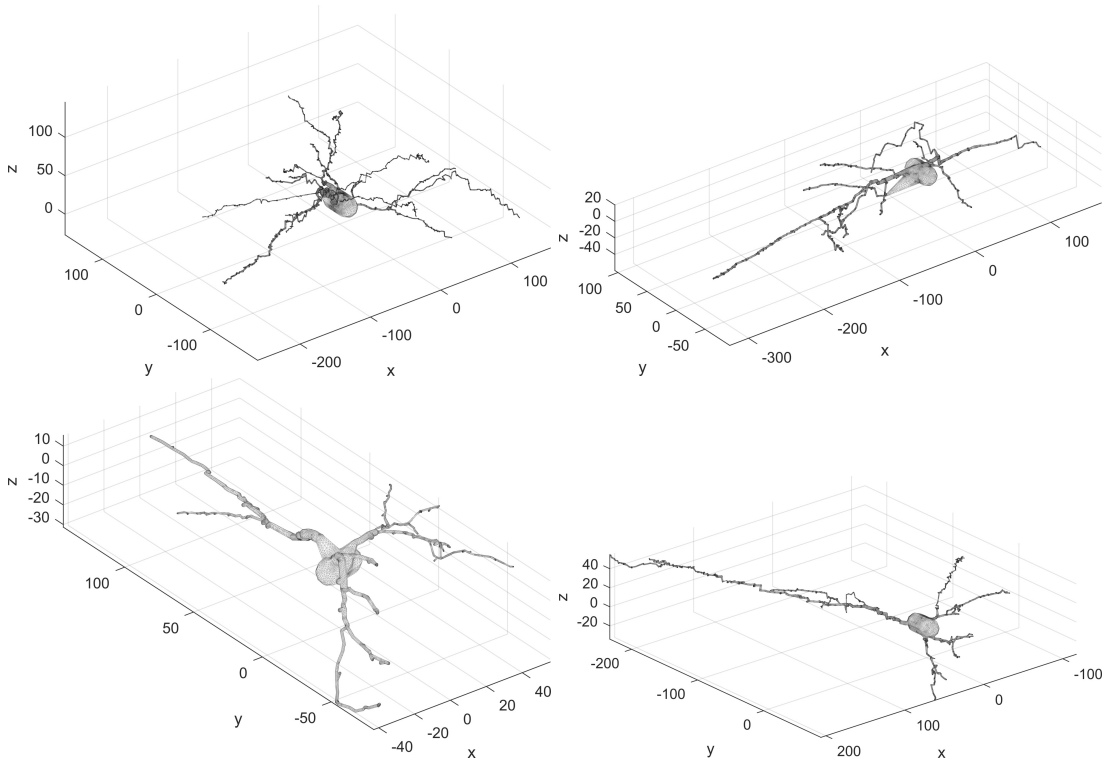

Figure 14: The finite elements meshes of four pyramidal neurons. The unit is μm. Top left: *02a_pyramidal2aFI*. Top right: *03b_pyramidal9aFI*. Bottom left: *03b_pyramidal2aACC*. Bottom right: *10a_pyramidal15aACC*.

7.5 Commented driver for typical simulations

```
%%%% The DISTRIBUTE folder contains a commented general purpose driver
%%%% called driver_spindoctor_neuronmodule_commented.m.
%%%% It is highly recommended to read this driver to understand the
    workflow
%%%% of SpinDoctor and the Neuron Module.
%%%% driver_spindoctor_neuronmodule_commented.m does not use saved
    simulation data,
%%%% all simulations are run from scratch.
%%%% The user is advised to read the latest version
%%%% from \url{https://github.com/jingrebeccali/SpinDoctor}

clear all;
close all;
restoredefaultpath;

%%%% User needs to choose to see some of the typical plots or not.
DO_PLOTS = true;

%%%% User needs to choose whether to output the solution of the BTPDE,
%%%% in the case only one diffusion direction is simulated.
%%%% If the finite elements mesh is large, it is
%%%% recommended not to output the magnetization.
%%%% If the flag is set to false, the function PLOT_PDESOLUTION cannot
    be
%%%% called.
%%%% For multiple diffusion directions (HARDI), the magnetization is
    never
%%%% outputted and the flag below is not taken into account.
OUTPUT_MAGNETIZATION = true;

%%%% User needs to define the directory where the 3 user provided input
     files are kept.
user_inputfiles_dir = 'params_files/
    params_spindoctor_neuronmodule_commented';

%%%% User needs to define the names of the 3 user provided input files
%%%% found in user_inputfiles_dir
fname_params_cells = 'params_cells_neuron.in';
fname_params_simul_domain  = 'params_simul_domain_neuron.in';
fname_params_simul_experi = 'params_simul_experi_neuron.in';

%%%% We add the path to the directory of the 3 user provided input
    files.
eval(['addpath ', user_inputfiles_dir]);

%%%% We add the path to the top level source code directory.
addpath SRC
%%%% We add the path to the deeper level source code directories.
%%%% We note the functions that need to be called in a typical
    simulation
%%%% situation are contained in the top level directory 'SRC'.
%%%% The less often used functions are kept in the deeper levels.
addpath SRC/PDE SRC/DMRI SRC/FEM SRC/GEOM SRC/TETGEN

%%%% We read the params_cells input file and create the geometrical
    configuration
```

```
%%%% In the first line of the params_cells input file, we expect a number
%%%% between 1 and 3;
%%%% 1 = ellipsoids, 2 = cylinders, 3 = neuron from msh file;
[params_cells,fname_cells] = create_geom(fname_params_cells);

%%%% We read the params_simul_domain input file;
[params_domain_geom,params_domain_pde,params_domain_femesh] ...
    = read_params_simul_domain(fname_params_simul_domain);

%%%% We set up the PDE model in the geometrical compartments.
[DIFF_cmpts,kappa_bdys,IC_cmpts,OUT_cmpts_index,ECS_cmpts_index,
    IN_cmpts_index,Ncmpt,Nboundary] ...
    = PREPARE_PDE(params_cells.ncell,params_cells.cell_shape,
        params_domain_geom,params_domain_pde);

%%%% We make a directory to store the finite elements mesh for this simulation;
save_meshdir_path = [fname_cells,'_dir'];
tf = isfolder(save_meshdir_path);
if (~tf)
    call_cmd = ['mkdir ',save_meshdir_path];
    disp(call_cmd);
    eval([call_cmd]);
end
%%%% We use an existing finite elements mesh or we create a new finite
%%%% elements mesh.
%%%% The name of the finite elements mesh is stored in the string fname_tetgen_femesh
ns = regexp(fname_cells,'/');
save_mesh_name = [fname_cells(ns(end)+1:end),'Htetgen',num2str(
    params_domain_femesh.Htetgen),'msh'];
fname_tetgen = [save_meshdir_path,'/',save_mesh_name];
fname_tmp = [fname_tetgen,'.1'];
tf = isfile([fname_tmp,'.node']);
if (tf)
    fname_tetgen_femesh = fname_tmp;
else
    [fname_tetgen_femesh] = ...
        create_femesh_fromcells(params_cells,fname_cells,
            params_domain_geom,params_domain_femesh,fname_tetgen);
end


%%%% We do not deform the finite elements mesh if in the Neuron module.
%%%% We do deform the finite elements mesh if in SpinDoctor White Matter
%%%% mode.
%%%% The finite elements mesh is stored in mymesh
if (params_cells.cell_shape == 3)
    params_cells.para_deform = [0,0]';
    [mymesh,cmpts_bdys_mat] = read_tetgen(fname_tetgen_femesh,
        params_cells.para_deform,Ncmpt,Nboundary);
else
    [mymesh,cmpts_bdys_mat] = read_tetgen(fname_tetgen_femesh,
        params_cells.para_deform,Ncmpt,Nboundary);
end

```

```
%%%% We plot the finite elements mesh
if (DO_PLOTS)
    PLOT_FEMESH(mymesh,OUT_cmpts_index,ECS_cmpts_index,IN_cmpts_index);
end

%%%% We read the params_simul_experi input file;
[experi_common,experi_hadc,experi_btpde] ...
    = read_params_simul_experi(fname_params_simul_experi);

%%%% We get the volume and the surface area quantities from the mesh
[VOL_cmpts,SA_cmpts,SAu_cmpts,VOL_allcmpts,VF_cmpts,SoV_cmpts] ...
    = GET_VOL_SA(mymesh,experi_common.gdir);

nexperi = length(experi_common.sdeltavec);

%%%% We run the simulation for one diffusion-encoding direction
if (experi_common.ngdir_total == 1)
    if (~isempty(experi_btpde))
        nb = size(experi_btpde.bvalues,2);
        %%%% We solve the BTPDE
        [SOL,SIG_BTPDE_cmpts,SIG_BTPDE_allcmpts,difftime,ctime_btpde]
           ...
            = BTPDE(experi_btpde,mymesh,DIFF_cmpts,kappa_bdys,IC_cmpts,
               OUTPUT_MAGNETIZATION);
        %%%% We fit the ADC of the signal
        [ADC_BTPDE_cmpts,ADC_BTPDE_allcmpts,ADC_allcmpts_S0] ...
            = FIT_SIGNAL(SIG_BTPDE_cmpts,SIG_BTPDE_allcmpts,
               experi_btpde.bvalues);
        %%%% We obtain the signal of free diffusion
        [Sig_free,ADC_free_allcmpts] = ADCFREE(experi_btpde.bvalues,
           DIFF_cmpts,VOL_cmpts,IC_cmpts);
        %%%% We plot the BTPDE signal, the S0*exp(-ADC*b) curve, and
           the
        %%%% free diffusion curves together.
        if (DO_PLOTS)
            PLOT_SIGNAL(experi_btpde.bvalues,SIG_BTPDE_allcmpts,
               Sig_free,ADC_allcmpts_S0,ADC_BTPDE_allcmpts,'BTPDE');
        end
        %%%% We plot the magnetization solution of the BTPDE if the
           user
        %%%% has set the flag OUTPUT_MAGNETIZATION to true
        if (DO_PLOTS & OUTPUT_MAGNETIZATION)
            for iexperi = 1:nexperi
                for ib = 1:nb
                    bv = experi_btpde.bvalues(iexperi,ib);
                    title_str = ['BTPDE magnetization. ','Experi ',...
                        num2str(iexperi),', b= ',num2str(bv)];
                    PLOT_PDESOLUTION(mymesh,SOL{iexperi}{ib},
                       OUT_cmpts_index,ECS_cmpts_index,IN_cmpts_index,
                       title_str);
                end
            end
        end
    end
    if (~isempty(experi_hadc))
        %%%% We solve the HADC model
        [ADC_HADC_cmpts,ADC_HADC_allcmpts,ctime_btpde] ...
            = HADC(experi_hadc,mymesh,DIFF_cmpts,IC_cmpts);
```

```
        %%%% We find the short time approximation (STA) of the
            effective
        %%%% diffusion.
        [ADC_STA_cmpts,ADC_STA_allcmpts] = STA(experi_common,DIFF_cmpts
            ,VOL_cmpts,SAu_cmpts,IC_cmpts);
        %%%% We plot the HADC and the STA
        if (DO_PLOTS)
            PLOT_ADC(ADC_HADC_cmpts,ADC_HADC_allcmpts,DIFF_cmpts,'HADC'
                );
            PLOT_ADC(ADC_STA_cmpts,ADC_STA_allcmpts,DIFF_cmpts,'STA');
        end
    end
end
%%%% We run the simulation for many diffusion-encoding directions
    uniformly
%%%% distributed in the unit sphere in 3 dimensions.
if (experi_common.ngdir_total > 1)

    ngdir_total = experi_common.ngdir_total;
    %%%% We obtain the multiple diffusion-encoding directions uniformly
    %%%% distributed in the unit sphere in 3 dimensions.
    [points_gdir,graddir_index,negii] = HARDI_PTS(ngdir_total);


    if (~isempty(experi_btpde))
        %%%% We solve the BTPDE in many diffusion directions.
        [SIG_BTPDE_cmpts_hardi,SIG_BTPDE_allcmpts_hardi,
            ctime_btpde_hardi] ...
            = SIG_BTPDE_HARDI(experi_btpde,mymesh,DIFF_cmpts,kappa_bdys
                ,IC_cmpts,...
            points_gdir,graddir_index,negii);
        %%%% We plot the normalized signal in many diffusion directions
            .
        if (DO_PLOTS)
            S0 = sum((IC_cmpts.*VOL_cmpts));
            nb = size(experi_btpde.bvalues,2);
            for iexperi = 1:nexperi
                for ib = 1:nb
                    bv = experi_btpde.bvalues(iexperi,ib);
                    title_str = ['BTPDE. All Cmpts. ','Experi ',...
                        num2str(iexperi),', b= ',num2str(bv)];
                    PLOT_HARDI_PT(points_gdir,...
                        squeeze(real(SIG_BTPDE_allcmpts_hardi(:,iexperi
                            ,ib)))/S0,title_str);
                end
            end
        end
    end
    %%%%% We solve the HADC model
    if (~isempty(experi_hadc))
        %%%% We solve the HADC model in many diffusion directions.
        [ADC_HADC_cmpts_hardi,ADC_HADC_allcmpts_hardi,ctime_hadc_hardi]
            ...
            = HADC_HARDI(experi_hadc,mymesh,DIFF_cmpts,IC_cmpts,...
            points_gdir,graddir_index,negii);
        %%%% We plot the normalized ADC (ADC/D0) in many diffusion
            directions.
        if(DO_PLOTS)
```

```
193             ADC0 = sum((DIFF_cmpts.*VF_cmpts));
194             for iexperi = 1:nexperi
195                 title_str = ['HADC. All Cmpts. ','Experi ',num2str(
                        iexperi)];
196                 PLOT_HARDI_PT(points_gdir,...
197                     squeeze(ADC_HADC_allcmpts_hardi(:,iexperi)/ADC0),
                            title_str);
198             end
199         end
200     end
201 end
```

8 Matrix Formalism Module for realistic neurons

The Matrix Formalism Module follows the same workflow as SpinDoctor and builds upon the functionalities of SpinDoctor. The Matrix Formalism Module uses the same set of three input files as SpinDoctor. SpinDoctor allows the easy construction of multiple compartment models of the brain white matter, with the possibility of coupling water diffusion between the geometrical compartments by permeable membranes. As this time, we have not implemented the Matrix Formalism 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 Matrix Formalism 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 Matrix Formalism Module. However, we have kept the exactly same input file formats as SpinDoctor in anticipation of the future development of the Matrix Formalism Module for permeable membranes. In particular, the various compartments in SpinDoctor have designations as IN, OUT, and ECS, and in the Matrix Formalism Module, the geometry defined by the finite element mesh is designated as the OUT compartment.

8.1 Matrix Formalism signal representation

It is known, though perhaps not as well known as it deserves to be, that using the Matrix Formalism [15, 16], the diffusion MRI signal has the following representation for the PGSE sequence. Let $\phi_n(\boldsymbol{x})$ and λ_n, $n = 1, \cdots$, be the L^2-normalized eigenfunctions and eigenvalues associated to the Laplace operator with homogeneous Neumann boundary conditions (the surface of the neurons are supposed impermeable):

$$\begin{aligned} \nabla \mathcal{D}_0 \left(\nabla \phi_n(\boldsymbol{x})\right) &= \lambda_n \phi_n(\boldsymbol{x}), \quad && \boldsymbol{x} \in \Omega, \\ \mathcal{D}_0 \nabla \phi_n(\boldsymbol{x}) \cdot \boldsymbol{\nu}(\boldsymbol{x}) &= 0, && \boldsymbol{x} \in \Gamma. \end{aligned}$$

We assume the non-negative eigenvalues are ordered in non-decreasing order:

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

so that $\lambda_1 = 0$ (this means the first Laplace eigenfunction is the constant function, we assume the neuron is a connected domain). Let L be the diagonal matrix containing the first N_{eig} Laplace eigenvalues:

$$L = \mathrm{diag}[\lambda_1, \lambda_2, \cdots, \lambda_{N_{eig}}] \in \mathbb{R}^{N_{eig} \times N_{eig}}. \tag{16}$$

Let W be the $N_{eig} \times N_{eig}$ matrix whose entries are:

$$W(\boldsymbol{g}) = g_x A^x + g_y A^y + g_z A^z \tag{17}$$

where

$$A^i = \left[a^i_{mn}\right], \quad i = \{x, y, z\}, \quad 1 \leq n, m \leq N_{eig}, \tag{18}$$

are three symmetric $N_{eig} \times N_{eig}$ matrices whose entries are the first order moments in the coordinate directions of the product of pairs of eigenfunctions:

$$a^x_{mn} = \int_\Omega x\phi_m(\boldsymbol{x})\phi_n(\boldsymbol{x})d\boldsymbol{x},$$
$$a^y_{mn} = \int_\Omega y\phi_m(\boldsymbol{x})\phi_n(\boldsymbol{x})d\boldsymbol{x},$$
$$a^z_{mn} = \int_\Omega z\phi_m(\boldsymbol{x})\phi_n(\boldsymbol{x})d\boldsymbol{x}.$$

We define the complex-valued matrix $K(\boldsymbol{g})$ and diagonalize it:

$$K(\boldsymbol{g}) \equiv L + \sqrt{-1}\,(g_x A^x + g_y A^y + g_z A^z) = V\Sigma V^{-1}, \tag{19}$$

where V has the eigenvectors in the columns and Σ has the eigenvalues on the diagonal. Then the following matrix

$$H(\boldsymbol{g}, f) = Ve^{-\delta\Sigma}V^{-1}e^{-(\Delta-\delta)L}\left(V^{-1}\right)^* e^{-\delta\Sigma^*}V^*, \tag{20}$$

gives the Matrix Formalism signal representation of the solution to the Bloch-Torrey PDE. For a constant initial density, the Matrix Formalism signal representation is the entry in the first row and first column of $H(\boldsymbol{g}, f)$:

$$S^{\mathrm{MF}}(\boldsymbol{g}, f) = H_{11}(\boldsymbol{g}, f). \tag{21}$$

We note that in Eq. 20 the matrices in the exponent (Σ and L and Σ^*) are diagonal and in this case, the matrix exponential is also diagonal. The notation * denotes the matrix complex conjugate transpose.

From the Matrix Formalism signal, the analytical expression for its ADC is the following:

$$\frac{ADC}{\mathcal{D}_0} = \sum_{n=1}^{N_{eig}} \frac{(\boldsymbol{u_g}\cdot\boldsymbol{a}_{1n})^2\lambda_n \int_0^{TE} F(t)\left(\int_0^t e^{-\mathcal{D}_0\lambda_n(t-s)}f(s)ds\right)dt}{\int_0^{TE} F^2(t)dt},$$

where the coefficients $\boldsymbol{a}_{1n} = \left[a^x_{1n}, a^y_{1n}, a^z_{1n}\right]^T$ are

$$a^x_{1n} = \frac{1}{\sqrt{|\Omega|}}\int_\Omega x\phi_n(\boldsymbol{x})d\boldsymbol{x},$$
$$a^y_{1n} = \frac{1}{\sqrt{|\Omega|}}\int_\Omega y\phi_n(\boldsymbol{x})d\boldsymbol{x}, \tag{22}$$
$$a^z_{1n} = \frac{1}{\sqrt{|\Omega|}}\int_\Omega z\phi_n(\boldsymbol{x})d\boldsymbol{x}.$$

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:

$$\frac{ADC(\boldsymbol{u_g}, f)}{\mathcal{D}_0} = \boldsymbol{u_g}^T\frac{D^{\mathrm{MF}}(f)}{\mathcal{D}_0}\boldsymbol{u_g}, \tag{23}$$

where the Matrix Formalism effective diffusion tensor is seen to be:

$$\frac{D^{\mathrm{MF}}(f)}{\mathcal{D}_0} = \sum_{n=1}^{N_{eig}} J(\lambda_n, f)\boldsymbol{a}_{1n}\boldsymbol{a}^T_{1n}, \tag{24}$$

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

$$J(\lambda_n, f) = \frac{\lambda_n\int_0^{TE} F(t)\left(\int_0^t e^{-\mathcal{D}_0\lambda_n(t-s)}f(s)ds\right)dt}{\int_0^{TE} F^2(t)dt}. \tag{25}$$

We also allow the possibility of computing the Matrix Formalism Gaussian Approximation (MFGA) signal, given as

$$S^{\mathrm{MFGA}}(\boldsymbol{g}, f) = \exp\left(-\boldsymbol{u_g}^T D^{\mathrm{MF}}(f)\boldsymbol{u_g}\, b\right). \tag{26}$$

8.1.1 Finite elements discretization of the Laplace operator

In a recent work, we have taken the morphological reconstructions of some realistic neurons from NeuroMorpho.Org [17] and processed them to create high quality finite elements meshes. The procedure is described in [18]. Specifically, there are 65 realistic neurons whose finite elements meshes we have made available to the public, with the mesh size ranges from having 15163 nodes to 622553 nodes. These are the finite elements meshes we use in the procedure described below to calculate the Matrix Formalism signal. To be clear, each neuron finite elements mesh consists of

1. a list of N_v nodes in three dimensions;
2. a list of N_e tetrahedral elements ($4 \times N_e$ indices referencing the nodes);

In SpinDoctor, the finite elements space is the space of 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 elements nodes:

$$\{\varphi_k(\boldsymbol{x})\}, \quad k = 1, \ldots, N_v.$$

In fact, any function in the finite elements space can be written as a linear combination of the above basis functions

$$\sum_{k=1}^{N_v} c_k \varphi_k(\boldsymbol{x}).$$

To discretize the Laplace operator with zero Neumann boundary conditions, two matrices are needed: $\boldsymbol{M} \in \mathbb{R}^{N_v \times N_v}$ and $\boldsymbol{S} \in \mathbb{R}^{N_v \times N_v}$, known in the FEM literature as the mass and stiffness matrices, respectively, which are defined as follows:

$$\boldsymbol{M}_{jk} = \int_\Omega \varphi_k(\boldsymbol{x}) \varphi_j(\boldsymbol{x})\, d\boldsymbol{x}, \qquad \boldsymbol{S}_{jk} = \int_\Omega \mathcal{D}_0\, \nabla\varphi_k(\boldsymbol{x}) \cdot \nabla\varphi_j(\boldsymbol{x})\, d\boldsymbol{x}, \quad 1 \le j, k \le N_v.$$

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 elements matrices in the assembly of $\boldsymbol{S}$ and $\boldsymbol{M}$ are evaluated at once and stored in a full matrix of size $4 \times 4 \times N_e$, where N_e denotes the number of tetrahedral elements.

The finite elements discretization described above changes the continuous Laplace operator eigenvalue problem to the following discrete *matrix* eigenvalue problem:

$$\begin{aligned} &\text{find } \{\lambda_n, p_n\},\ 1 \le n \le N_v, \ \text{ such that} \\ &\qquad \lambda_n \boldsymbol{M} p_n = -\boldsymbol{S} p_n,\ p_n \in \mathbb{R}^{N_v}, \end{aligned} \tag{27}$$

where p_n is the eigenvector (of N_v entries) associated to the eigenvalue λ_n. 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

$$\phi_n(\boldsymbol{x}) = \sum_{j=1}^{N_v} p_n^j \varphi_j(\boldsymbol{x}),$$

where the entries of the eigenvector p_n are the coefficients of the finite elements basis functions. To obtain the Matrix Formalism signal representation, we calculated the first moments in the three coordinate directions of the product of pairs of eigenfunctions. They can be written in the following form, that involves the first moments of the finite elements basis function pairs, shown in parentheses:

$$\begin{aligned} a^x_{mn} &= \sum_{j=1}^{N_v}\sum_{k=1}^{N_v} p_n^j p_m^k \left(\int_\Omega x\ \varphi_j(\boldsymbol{x})\varphi_k(\boldsymbol{x}) d\boldsymbol{x} \right), \\ a^y_{mn} &= \sum_{j=1}^{N_v}\sum_{k=1}^{N_v} p_n^j p_m^k \left(\int_\Omega y\ \varphi_j(\boldsymbol{x})\varphi_k(\boldsymbol{x}) d\boldsymbol{x} \right), \quad 1 \le m, n \le N_{eig}. \\ a^z_{mn} &= \sum_{j=1}^{N_v}\sum_{k=1}^{N_v} p_n^j p_m^k \left(\int_\Omega z\ \varphi_j(\boldsymbol{x})\varphi_k(\boldsymbol{x}) d\boldsymbol{x} \right). \end{aligned} \tag{28}$$

8.1.2 Eigenfunction length scale

The analytical eigenvalues of a line segment of length H are

$$\lambda_{\{1,2,\cdots\}} = \{0, \gamma_l\}, \gamma_l = \mathcal{D}_0 \left(\frac{\pi}{H/k}\right)^2, k = 1, 2, \cdots \tag{29}$$

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 (from the line segment eigenvalue formula):

$$l_s(\lambda) = \frac{\pi}{\sqrt{\lambda/\mathcal{D}_0}}, \tag{30}$$

and charaterize the computed eigenmode by $l_s(\lambda_n)$ instead of λ_n. 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 $J(\lambda_n, f)(\boldsymbol{u_g} \cdot \boldsymbol{a}_{1n})^2$. Thus, we call $\boldsymbol{a}_{1n} = \left[a^x_{1n}, a^y_{1n}, a^z_{1n}\right]^T$ the "diffusion direction" of the nth eigenmode. We remind that the three components of $\boldsymbol{a}_{1n}$ are the first moments in the 3 principle axes directions of the associated eigenfunction.

8.1.3 Eigenvalues interval and minimum length scale

We do not want to compute the entire set of eigenvalues and eigenvectors

$$\lambda_n \boldsymbol{M} p_n = -\boldsymbol{S} p_n, \ 1 \leq n \leq N_v,$$

of the matrix eigenvalue problem in Eq. 27, because the size of $\boldsymbol{M}$ and $\boldsymbol{S}$ is determined by the finite elements discretization (it is equal to N_v, the number of finite elements nodes). We remind that for the 65 realistic neurons whose finite elements meshes are available in the Neuron Module of SpinDoctor, the mesh size ranges from having 15163 nodes to 622553 nodes. This means most of the rapidly oscillating eigenmodes in the matrix eigenvalue problem are linked to the finite elements discretization, and not 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 compute the eigenvalues. We set the interval to be $[0, (\pi/l_s^{min})^2 \mathcal{D}_0]$, where l_s^{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_v. This restricted eigenvalue interval for the matrix eigenvalue problem was implemented by called the "pdeeig" function in the MATLAB PDE Toolbox, after defining a PDE model whose PDE is the Laplace equation with the diffusion coefficient $\mathcal{D}_0$.

The user needs to choose the minimum length scale of interest. When there are too many eigenvalues in the requested eigenvalue interval, MATLAB takes a long time to converge, so it is better to break the requested eigenvalue interval into many subintervals such that there are no more than 10 or so eigenvalues in each interval. Keeping this in mind, the user should specify how many subintervals to break the requested eigenvalue interval into.

8.2 User provided input files

In SpinDoctor, there are three input files in which the user specifies the parameters of the desired simulations. They are:

1. *params_cells.in*: contains the cells parameters
2. *params_simul_domain.in*: contains the simulation domain parameters
3. *params_simul_experi.in*: contains the simulation experiment parameters

We list the input files the user must provide in order to use the Matrix Formalism Module, noting where relevant, the input parameters that are not applicable (marked by "na") to the current version of the Matrix Formalism Module.

8.2.1 Read cells parameters

The user provides an input file for the cells parameters, in the format described in Table 14. To simulate the diffusion MRI signal of a neuron, the user chooses option 3 for the cell shape. The user specifies the name of the neuron to be simulated in line 2.

Line	Variable name	Example	Explanation
1	cell_shape	3	1 = spheres; 2 = cylinders; 3 = neuron;
2	fname	'msh_files/pyramidal/ 02b_pyramidal1aACC'	file name of neuron mesh
3	ncell	1	number of cells
4	Rmin	na	min Radius
5	Rmax	na	max Radius
6	dmin	na	min (%) distance between cells
7	dmax	na	max (%) distance between cells
8	para_deform	na	$[\alpha \quad \beta]$; α defines the amount of bend; β defines the amount of twist
9	Hcyl	na	height of cylinders

Table 14: Input file containing cells parameters. "na" means not applicable to neuron simulation.

8.2.2 Read simulation domain parameters

The user provides an input file for the simulation domain parameters, in the format described in Table 15.

Line	Variable name	Example	Explanation
1	Rratio	na	
2	include_ECS	na	
3	ECS_gap	na	
4	dcoeff_IN	na	
5	dcoeff_OUT	0.002	diffusion coefficient in OUT cmpt
6	dcoeff_ECS	na	
7	ic_IN	na	
8	ic_OUT	1	initial spin density in OUT cmpt
9	ic_ECS	na	
10	kappa_IN_OUT	na	
11	kappa_OUT_ECS	na	
12	Htetgen	-1	Requested tetgen mesh size; -1 = Use tetgen default;
13	tetgen_cmd	'SRC/TETGEN/ tetGen/win64/ tetgen'	path to tetgen_cmd

Table 15: Input file of simulation domain parameters. "na" means not applicable to neuron simulation.

8.2.3 Read simulation experiment parameters

The user provides an input file for the simulation experiment parameters, in the format described in Table 16.

Line	Variable name	Example	Explanation
1	ngdir	300	number of gradient direction; if $ngdir > 1$, the gradient directions are distributed uniformly on a sphere; if $ngdir = 1$, take the gradient direction from the line below;
2	gdir	1.0 0.0 0.0	gradient direction; No need to normalize;
3	nexperi	2	number of sequences to simulate;
4	sdeltavec	2500 10000	small delta;
5	bdeltavec	5000 43000	big delta;
6	seqvec	1 1	diffusion sequence of experiment; 1 = PGSE; 2 = OGSEsin; 3 = OGSEcos;
7	npervec	0 0	number of period of OGSE;
8	solve_hadc	na	0 = do not solve HADC; Otherwise solve HADC;
9	rtol_deff, atol_deff	na	$[r_{tol} \quad a_{tol}]$; relative and absolute tolerance for HADC ODE solver;
10	solve_btpde	1	0: do not solve BTPDE; Otherwise solve BTPDE;
11	rtol_bt, atol_bt	1e-2 1e-4	$[r_{tol} \quad a_{tol}]$; relative and absolute tolerance for BTPDE ODE solver;
12	nb	4	number of b-values;
13	blimit	0	0=specify bvec; 1=specify[bmin,bmax]; 2 = specify[gmin,gmax];
14	const_q	0	0: use input bvalues for all experiments; 1: take input bvalues for the first experiment and use the same q for the remaining experiments
15	b-values	0 1000 2000 3000	b-values or [bmin, bmax] or [gmin, gmax]; depending on line 13;

Table 16: Input file for simulation experiment parameters.

Quantities relevant to the Matrix Formalism Module

In Table 17 we list quantities relevant to the Matrix Formalism Module. The braces in the "Size" column denote MATLAB cell data structure and the brackets denote MATLAB matrix data structure.

Variable name	Size	Explanation
experiment		a structure that contains experimental parameters.
mymesh		a structure that contains the finite elements mesh.
Ncmpt		Fixed to be 1 (the number of compartments)
neig		the number of eigenvalues
nnodes		the number of finite element nodes
nexperi		the number of experiments (sequences) to be simulated
nb		the number of b-values
ngdir		the number of diffusion-encoding directions
DIFF_cmpts	[Ncmpt]	the intrinsic diffusion coefficient in the compartments.
VOL_cmpts	[Ncmpt]	the volume of the compartments.
IC_cmpts	[Ncmpt]	the initial spin density of the compartments.
EigLim_cmpts	[Ncmpt]	the upper limit of the eigenvalues interval to be passed to the MATLAB pdeeig function.
EIG_value_cmpts	{Ncmpt} [neig]	the eigenvalues in the compartments.
EIG_func_cmpts	{Ncmpt} [nnodes,neig]	the eigenfunctions in the compartments.
EIG_proj_cmpts	{Ncmpt} [3×neig×neig]	the first moments of the products of the eigenfunctions in the compartments.
DTENSOR_cmpts	[Ncmpt × nexperi × 3 × 3]	the effective diffusion tensor in the compartments.
DTENSOR_allcmpts	[nexperi × 3 × 3]	the effective diffusion tensor summed over all compartments.
SIG_MF_cmpts	[Ncmpt × nexperi × nb]	the MF signal in the compartments.
SIG_MF_allcmpts	[nexperi × nb]	the MF signal summed over all compartments.
SIG_MFGA_cmpts	[Ncmpt × nexperi × nb]	the MFGA signal in the compartments.
SIG_MFGA_allcmpts	[nexperi × nb]	the MFGA signal summed over all compartments.
points_gdir	[ngdir× 3]	uniformly distributed gradient directions on a sphere
SIG_MF_cmpts_dir	{Ncmpt}[ngdir × nexperi× nb]	MF signal in the compartments in the gradient directions.
SIG_MF_allcmpts_dir	[ngdir × nexperi × nb]	MF signal summed over all compartments in the gradient directions.
SIG_MFGA_cmpts_dir	{Ncmpt}[ngdir × nexperi× nb]	MFGA signal in the compartments in the gradient directions.
SIG_MFGA_allcmpts_dir	[ngdir × nexperi × nb]	MFGA signal summed over all compartments in the gradient directions.

Table 17: Some important Matrix Formalism Module output quantities.

8.3 Important functions of the Matrix Formalism Module

In Table 18 we list important functions of the Matrix Formalism Module. For detailed information about them, including argument lists, please read the online documentation.

Function name	Purpose
read_params_simul_domain	Reads the params_simul_domain input file.
read_params_simul_experi	Reads the params_simul_experi input file.
read_tetgen	Reads the finite elements mesh.
PREPARE_PDE	Set up the PDE model in the geometrical compartments.
GET_VOL_SA	Gets the volume and the surface area quantities from the finite elements mesh.
BTPDE	Computes the BTPDE signal in one diffusion-encoding direction.
HADC	Computes the HADC in one diffusion-encoding direction.
STA	Computes the short time approximation in one diffusion-encoding direction.
ADCFREE	Computes the free diffusion ADC.
FIT_SIGNAL	Fits the S_0 and the ADC of the signal.
HARDI_PTS	Provides gradient directions uniformly distributed in unit 3D sphere.
SIG_BTPDE_HARDI	Computes the BTPDE signal in multiple diffusion-encoding directions.
HADC_HARDI	Computes the HADC in multiple diffusion-encoding directions.
PLOT_FEMESH	Displays the finite elements mesh.
PLOT_SIGNAL	Displays the simulated signal in one diffusion-encoding direction.
PLOT_ADC	Displays the simulated ADC in one diffusion-encoding direction.
PLOT_HARDI_PT	Displays simulation results in multiple diffusion-encoding directions.
LAPLACE_EIG	Computes the Laplace eigenvalues, eigenfunctions and the first order moments of the products of the eigenfunctions.
MF_EIG_TO_LENGTH	Converts the computed eigenvalues into a length scale
MF_JN	Computes the quantity $J(\lambda_n, f)$
MF_DTENSOR	Computes the effective diffusion tensor
PLOT_DTENSOR	Displays the effective diffusion tensor
PLOT_PDESOLUTION	Plots the PDE solution on the finite elements mesh
PLOT_EIGENFUNCTION	Plots the eigenfunction on the finite elements mesh
SIG_MF	Computes the Matrix Formalism signal in one diffusion-encoding direction
SIG_MFGA	Computes the Matrix Formalism Gaussian Approximation signal in one diffusion-encoding direction.
SIG_MF_HARDI	Computes the Matrix Formalism signal in multiple diffusion-encoding directions.
SIG_MFGA_HARDI	Computes the Matrix Formalism Gaussian Approximation signal in multiple diffusion-encoding directions.

Table 18: Some important functions of the Matrix Formalism Module.

8.4 Matrix Formalism Module 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*. The intrinsic diffusion coefficient is set to $\mathcal{D}_0 = 2 \times 10^{-3}\,\mathrm{mm}^2/\mathrm{s}$, with an impermeable membrane. We simulated 2 diffusion-encoding sequences:
Experiment 1: f_1 is PGSE ($\delta = 10.6\mathrm{ms}, \Delta = 13\mathrm{ms}$),
Experiment 2: f_2 is PGSE ($\delta = 10.6\mathrm{ms}, \Delta = 73\mathrm{ms}$).

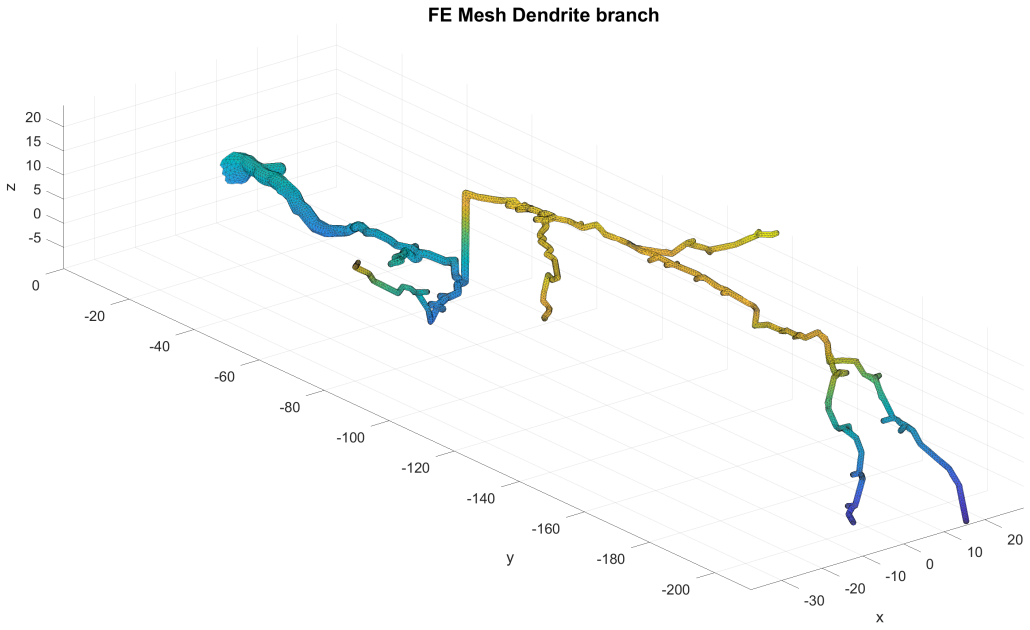

Figure 15: The finite elements mesh of the spindle neuron dendrite branch *03b_spindle6aACC_dendrites_2*. (Using the command PLOT_FEMESH).

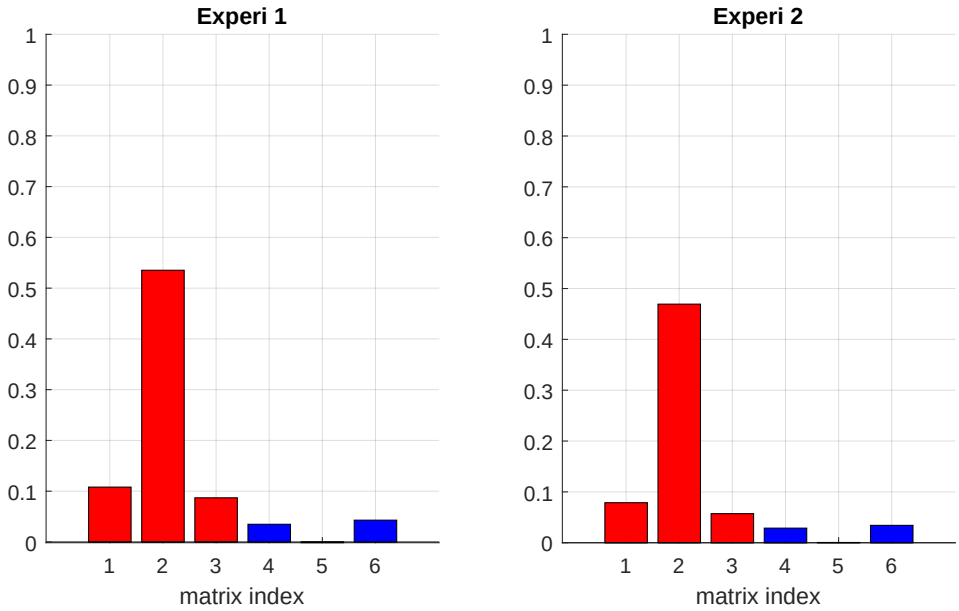

Figure 16: The 6 entries of the normalized diffusion tensor $D^{\mathrm{MF}}(f)/\mathcal{D}_0$ for the dendrite branch. Indices 1 to 6 are in the order of $\{D_{11}^{\mathrm{MF}}(f), D_{22}^{\mathrm{MF}}(f), D_{33}^{\mathrm{MF}}(f), D_{12}^{\mathrm{MF}}(f), D_{13}^{\mathrm{MF}}(f), D_{23}^{\mathrm{MF}}(f)\}/\mathcal{D}_0$. The diagonal entries of $D^{\mathrm{MF}}(f)/\mathcal{D}_0$ 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_DTENSOR).

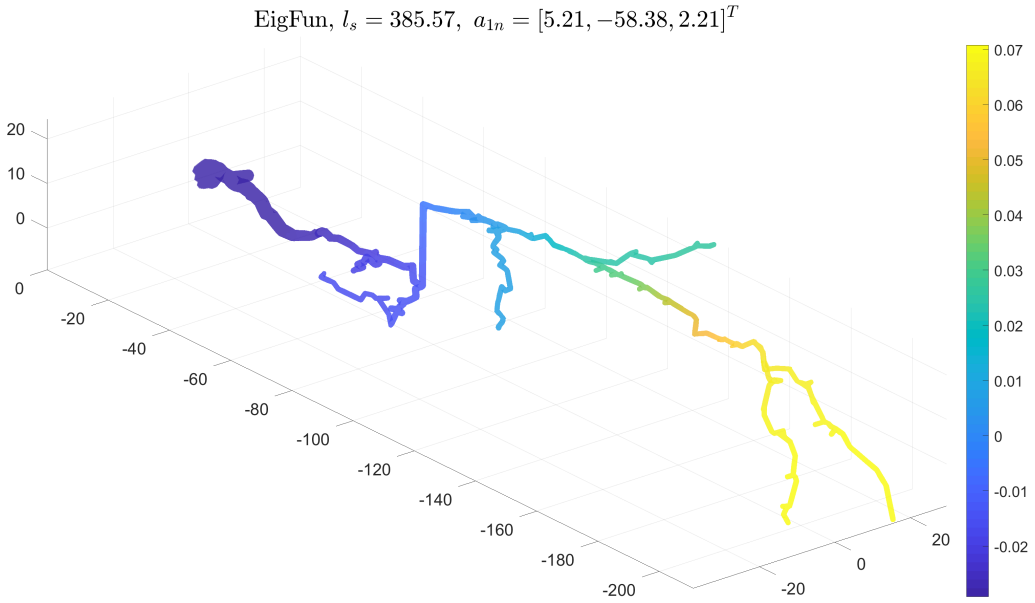

Figure 17: The eigenfunction with the associated length scale $l_s = 383.57\mu$m and the first moments vector $a_{1n} = [5.21, -58.38, 2.21]^T$. (Using the command PLOT_PDESOLUTION).

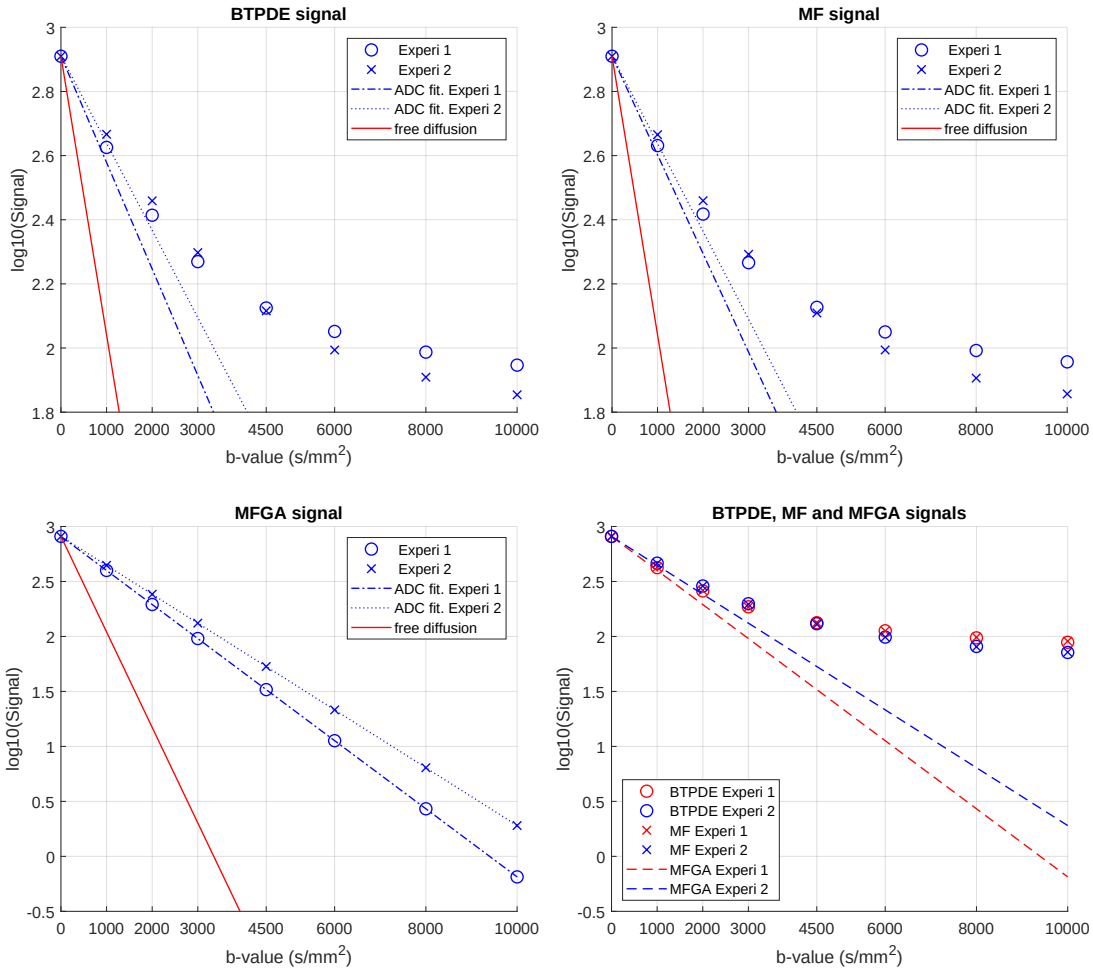

Figure 18: Top left: the BTPDE signals. Top right: the MF signals. Bottom left: the MFGA signals. The markers indicate the values of the simulated signals, the blue lines indicate the ADC fit of those signals, the red line is the signal of free diffusion at the intrinsic diffusion coefficient. Bottom right: the BTPDE, the MF, the MFGA signals plotted together. Experiment 1: f_1 is PGSE ($\delta = 10.6\text{ms}, \Delta = 13\text{ms}$), Experiment 2: f_2 is PGSE ($\delta = 10.6\text{ms}, \Delta = 73\text{ms}$). Diffusion-encoding direction $\boldsymbol{u_g} = [1, 1, 0]/\sqrt{2}$. (Using the command PLOT_SIGNAL).

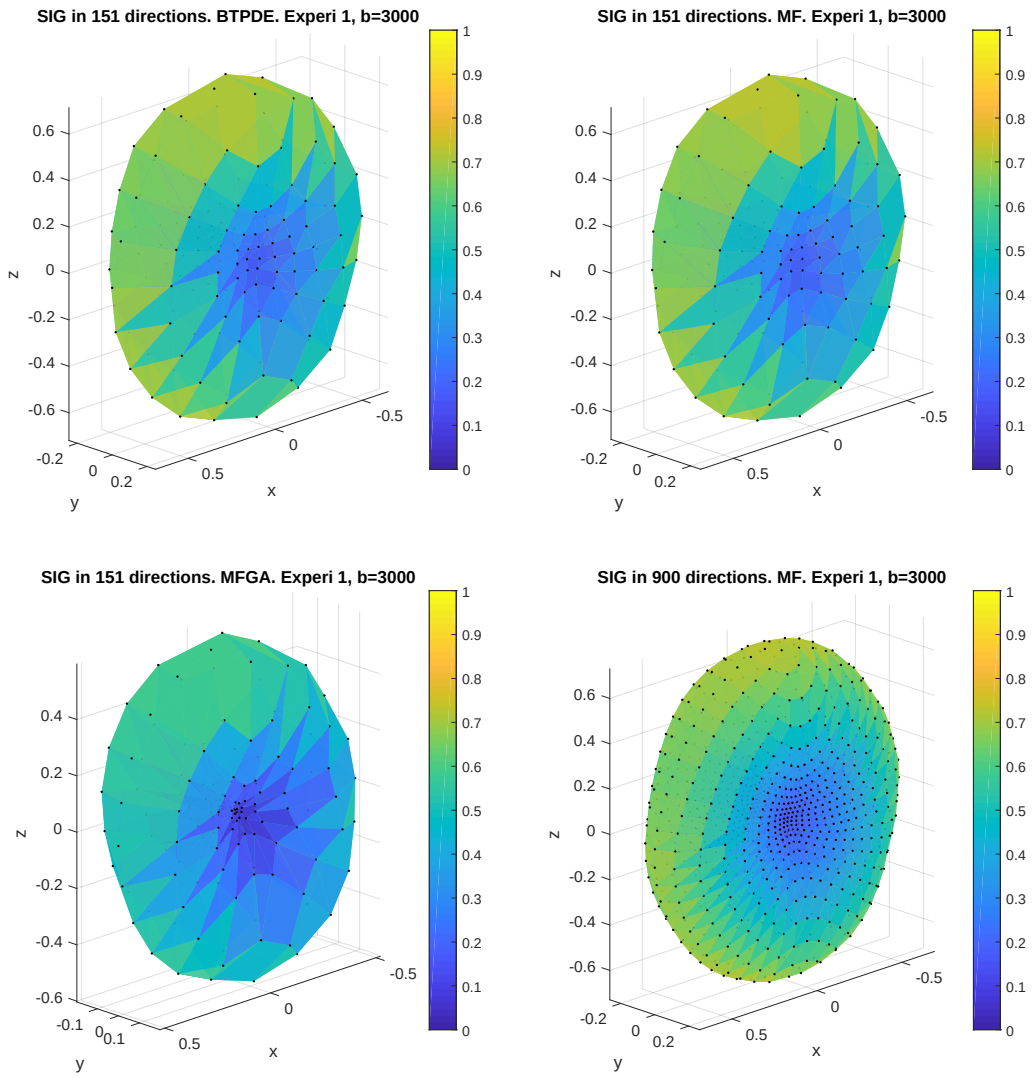

Figure 19: Top left: the BTPDE signals, S^{BTPDE}/S_0, in 151 diffusion-encoding directions. Top right: the MF signals, S^{MF}/S_0, in 151 diffusion-encoding directions. Bottom left: the MFGA signals, S^{MFGA}/S_0, in 151 diffusion-encoding directions. Bottom right: the MF signals, S^{MF}/S_0, in 900 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_PT).

8.5 Commented drivers for typical simulations

8.5.1 Driver to simulate one diffusion direction

```
%%%% The DISTRIBUTE folder contains a commented general purpose driver
%%%% called driver_spindoctor_MF_1direction_commented.m.
%%%% It is highly recommended to read this driver to understand the
    workflow
%%%% of SpinDoctor and the Matrix Formalism Module.
%%%% driver_spindoctor_MF_commented.m does not use saved simulation
    data,
%%%% all simulations are run from scratch.
%%%% The user is advised to read the latest version
%%%% from \url{https://github.com/jingrebeccali/SpinDoctor}
clear all;
close all;
```

```
restoredefaultpath;

%%%% User needs to choose to see some of the typical plots or not.
DO_PLOTS = true;
%%%% User needs to choose to the minimum length scale of interest
%%%% This controls the upper limit of eigenvalue interval to solve for
%%%% Laplace eigenvalues.
LSCALE_MIN = 4;
%%%% When there are too many eigenvalues in the requested eigenvalue
%%%% interval, Matlab takes a long time to converge.  So it is better to
%%%% break the requested eigenvalue interval into many subintervals so that
%%%% there are no more than 10 or so eigenvalues in each interval.
%%%% The User should specify how many subintervals to break the requested
%%%% eigenvalue interval into.  Values between 1 and 100 can be tried here.
NUM_EIGINTERVALS = 30;

%%%% User needs to define the directory where the 3 user provided input files are kept.
user_inputfiles_dir = 'params_files/params_spindoctor_MF_1direction_commented';

%%%% User needs to define the names of the 3 user provided input files
%%%% found in user_inputfiles_dir
fname_params_cells = 'params_cells_neuron.in';
fname_params_simul_domain  = 'params_simul_domain_neuron.in';
fname_params_simul_experi = 'params_simul_experi_neuron.in';

%%%% We add the path to the directory of the 3 user provided input files.
eval(['addpath ', user_inputfiles_dir]);

%%%% We add the path to the top level source code directory.
addpath SRC
%%%% We add the path to the deeper level source code directories.
%%%% We note the functions that need to be called in a typical simulation
%%%% situation are contained in the top level directory 'SRC'.
%%%% The less often used functions are kept in the deeper levels.
addpath SRC/PDE SRC/DMRI SRC/FEM SRC/GEOM SRC/TETGEN

%%%% We read the params_cells input file and create the geometrical configuration
%%%% In the first line of the params_cells input file, we expect a number
%%%% between 1 and 3;
%%%% 1 = ellipsoids, 2 = cylinders, 3 = neuron from msh file;
[params_cells,fname_cells] = create_geom(fname_params_cells);

%%%% We read the params_simul_domain input file;
[params_domain_geom,params_domain_pde,params_domain_femesh] ...
    = read_params_simul_domain(fname_params_simul_domain);

%%%% We set up the PDE model in the geometrical compartments.
```

```
[DIFF_cmpts,kappa_bdys,IC_cmpts,OUT_cmpts_index,ECS_cmpts_index,
    IN_cmpts_index,Ncmpt,Nboundary] ...
    = PREPARE_PDE(params_cells.ncell,params_cells.cell_shape,
        params_domain_geom,params_domain_pde);

%%%% We make a directory to store the finite elements mesh for this
    simulation;
save_meshdir_path = [fname_cells,'_dir'];
tf = isfolder(save_meshdir_path);
if (~tf)
    call_cmd = ['mkdir ',save_meshdir_path];
    disp(call_cmd);
    eval([call_cmd]);
end

%%%% We use an existing finite elements mesh or we create a new finite
%%%% elements mesh.
%%%% The name of the finite elements mesh is stored in the string
    fname_tetgen_femesh
ns = regexp(fname_cells,'/');
save_mesh_name = [fname_cells(ns(end)+1:end),'Htetgen',num2str(
    params_domain_femesh.Htetgen),'msh'];
fname_tetgen = [save_meshdir_path,'/',save_mesh_name];
fname_tmp = [fname_tetgen,'.1'];
tf = isfile([fname_tmp,'.node']);
if (tf)
    fname_tetgen_femesh = fname_tmp;
else
    [fname_tetgen_femesh] = ...
        create_femesh_fromcells(params_cells,fname_cells,
            params_domain_geom,params_domain_femesh,fname_tetgen);
end

%%%% We do not deform the finite elements mesh if in the Neuron module.
%%%% We do deform the finite elements mesh if in SpinDoctor White
    Matter
%%%% mode.
%%%% The finite elements mesh is stored in mymesh
if (params_cells.cell_shape == 3)
    params_cells.para_deform = [0,0]';
    [mymesh,cmpts_bdys_mat] = read_tetgen(fname_tetgen_femesh,
        params_cells.para_deform,Ncmpt,Nboundary);
else
    [mymesh,cmpts_bdys_mat] = read_tetgen(fname_tetgen_femesh,
        params_cells.para_deform,Ncmpt,Nboundary);
end

%%%% We plot the finite elements mesh
if (DO_PLOTS)
    PLOT_FEMESH(mymesh,OUT_cmpts_index,ECS_cmpts_index,IN_cmpts_index);
end

%%%% We read the params_simul_experi input file;
[experi_common,experi_hadc,experi_btpde] ...
    = read_params_simul_experi(fname_params_simul_experi);

%%%% We get the volume and the surface area quantities from the mesh
[VOL_cmpts,SA_cmpts,SAu_cmpts,VOL_allcmpts,VF_cmpts,SoV_cmpts] ...
```

```
    = GET_VOL_SA(mymesh,experi_common.gdir);

%%%% Instead of using 0 as the minimum of the eigenvalue interval we use
%%%% -infinity to make sure we get 0 as an output eigenvalue.
EigLim_min(1:Ncmpt) = -Inf;
%%%% The upper bound of the eigenvalue interval is determined by
%%%% LSCALE_MIN, which is the minimum length scale of interest.
%%%% This minimum length gives the upper bound of the eigenvalue interval
%%%% by using the formula that links the eigenvalues of the line segment
%%%% and the wavelength of the eigenfunctions on the line segment.
EigLim(1:Ncmpt) = DIFF_cmpts.*(pi^2./(LSCALE_MIN.^2));

%%%% Breaking the requested eigenvalue interval into a number of
%%%% subintervals so that Matlab matrix eigenvalue problem can converge
%%%% quickly.  The optimal choice
%%%% is when there are not more than 10 or 20 eigenvalues in each
%%%% subinterval.
EigIntervals(1:Ncmpt) = NUM_EIGINTERVALS;

for icmpt = 1:Ncmpt
    [EIG_value_cmpts{icmpt},EIG_proj_cmpts{icmpt},EIG_func_cmpts{icmpt},ctime_laplace(1,icmpt)]...
        = LAPLACE_EIG(mymesh.Pts_cmpt_reorder{icmpt},mymesh.Ele_cmpt_reorder{icmpt},...
        DIFF_cmpts(icmpt),VOL_cmpts(icmpt),EigLim(icmpt),EigLim_min(icmpt),EigIntervals(icmpt));
end

%%%% Converting eigenvalues into a length scale according to the
%%%% eigenvalues of the line segment and their connection to the wavelength
%%% of the Laplace eigenfunctions on the line segment.
[EIG_length_cmpts] = MF_EIG_TO_LENGTH(EIG_value_cmpts,DIFF_cmpts);

experi_mf = experi_btpde;
nb = size(experi_mf.bvalues,2);
nexperi = size(experi_mf.bvalues,1);

%%%% Computing the JN value that relates the eigenmodes to their
%%%% contribution to the Matrix Formalism signal for a diffusion-encoding
%%%% sequence.  Note, only PGSE is implemented at this point.
[MF_JN_cmpts] = MF_JN(EIG_value_cmpts,DIFF_cmpts,experi_mf);

% We compute the Matrix Formalism effective diffusion tensor
[DTENSOR_cmpts,DTENSOR_allcmpts] = MF_DTENSOR(VOL_cmpts,...
    IC_cmpts,EIG_value_cmpts,EIG_proj_cmpts,DIFF_cmpts,MF_JN_cmpts);

if (DO_PLOTS ~= 0)
    %%%% We plot the Matrix Formalism effective diffusion tensor
    PLOT_DTENSOR(DTENSOR_cmpts,DTENSOR_allcmpts,DIFF_cmpts);
    cmpts_use = 1;
    %%%% We plot the eigenfunction index by nindex
    for nindex = 2
        diffdir = EIG_proj_cmpts{cmpts_use}(:,nindex);
```

```
        title_str = ['EigFun, l_s = ', num2str(EIG_length_cmpts{1}(
            nindex))...
                ', a_{1n} = [',num2str(diffdir'),']^T'];
        PLOT_EIGENFUNCTION(mymesh,EIG_func_cmpts,OUT_cmpts_index,
            ECS_cmpts_index,IN_cmpts_index,...
             nindex,title_str);
    end
end

%%%% We run the simulation for one diffusion-encoding direction
if (experi_common.ngdir_total == 1)
    if (~isempty(experi_btpde))
        %%%% We solve the BTPDE
        [SOL,SIG_BTPDE_cmpts,SIG_BTPDE_allcmpts,difftime,ctime_btpde]
            ...
             = BTPDE(experi_btpde,mymesh,DIFF_cmpts,kappa_bdys,IC_cmpts,
                false);
        %%%% We fit the ADC of the signal
        [ADC_BTPDE_cmpts,ADC_BTPDE_allcmpts,ADC_allcmpts_S0] ...
             = FIT_SIGNAL(SIG_BTPDE_cmpts,SIG_BTPDE_allcmpts,
                experi_btpde.bvalues);
        %%%% We obtain the signal of free diffusion
        [Sig_free,ADC_free_allcmpts] = ADCFREE(experi_btpde.bvalues,
            DIFF_cmpts,VOL_cmpts,IC_cmpts);

        %%%% We compute the Matrix Formalism signal
        [SIG_MF_cmpts,SIG_MF_allcmpts,ctime_mf] ...
             = SIG_MF(experi_mf,VOL_cmpts,IC_cmpts,EIG_value_cmpts,
                EIG_proj_cmpts);
        %%%% We fit the ADC of the signal
        [ADC_MF_cmpts,ADC_MF_allcmpts,ADC_MF_allcmpts_S0] ...
             = FIT_SIGNAL(SIG_MF_cmpts,SIG_MF_allcmpts,experi_mf.bvalues
                );

        %%%% We compute the Matrix Formalism Gaussian Approximation
            signal
        [SIG_MFGA_cmpts,SIG_MFGA_allcmpts] ...
             = SIG_MFGA(experi_mf,VOL_cmpts,IC_cmpts,DTENSOR_cmpts);
        %%%% We fit the ADC of the signal
        [ADC_MFGA_cmpts,ADC_MFGA_allcmpts,ADC_MFGA_allcmpts_S0] ...
             = FIT_SIGNAL(SIG_MFGA_cmpts,SIG_MFGA_allcmpts,experi_mf.
                bvalues);

        % We plot the BTPDE, MF and MFGA signals
        if (DO_PLOTS ~= 0)
             PLOT_SIGNAL(experi_btpde.bvalues,SIG_BTPDE_allcmpts,
                Sig_free,ADC_allcmpts_S0,ADC_BTPDE_allcmpts,'BTPDE
                signal');
             PLOT_SIGNAL(experi_mf.bvalues,SIG_MF_allcmpts,Sig_free,
                ADC_MF_allcmpts_S0,ADC_MF_allcmpts,'MF signal');
             PLOT_SIGNAL(experi_mf.bvalues,SIG_MFGA_allcmpts,Sig_free,
                ADC_MFGA_allcmpts_S0,ADC_MFGA_allcmpts,'MFGA signal');
             PLOT_SIGNAL_BTPDE_MF(experi_btpde,SIG_BTPDE_allcmpts,
                SIG_MF_allcmpts,SIG_MFGA_allcmpts);
        end
    end
end
```

8.5.2 Driver for HARDI simulation

```
1
2  %%%% The DISTRIBUTE folder contains a commented general purpose driver
3  %%%% called driver_spindoctor_MF_1direction_commented.m.
4  %%%% It is highly recommended to read this driver to understand the
       workflow
5  %%%% of SpinDoctor and the Matrix Formalism Module.
6  %%%% driver_spindoctor_MF_commented.m does not use saved simulation
       data,
7  %%%% all simulations are run from scratch.
8  %%%% The user is advised to read the latest version
9  %%%% from \url{https://github.com/jingrebeccali/SpinDoctor}
10 clear all;
11 close all;
12 restoredefaultpath;
13
14 %%%% User needs to choose to see some of the typical plots or not.
15 DO_PLOTS = true;
16 %%%% User needs to choose to the minimum length scale of interest
17 %%%% This controls the upper limit of eigenvalue interval to solve for
18 %%%% Laplace eigenvalues.
19 LSCALE_MIN = 4;
20 %%%% When there are too many eigenvalues in the requested eigenvalue
21 %%%% interval, Matlab takes a long time to converge.  So it is better
       to
22 %%%% break the requested eigenvalue interval into many subintervals so
       that
23 %%%% there are no more than 10 or so eigenvalues in each interval.
24 %%%% The User should specify how many subintervals to break the
       requested
25 %%%% eigenvalue interval into.  Values between 1 and 100 can be tried
       here.
26 NUM_EIGINTERVALS = 30;
27
28 %%%% User needs to define the directory where the 3 user provided input
        files are kept.
29 user_inputfiles_dir = 'params_files/
       params_spindoctor_MF_hardi_commented';
30
31 %%%% User needs to define the names of the 3 user provided input files
32 %%%% found in user_inputfiles_dir
33 fname_params_cells = 'params_cells_neuron.in';
34 fname_params_simul_domain  = 'params_simul_domain_neuron.in';
35 fname_params_simul_experi = 'params_simul_experi_neuron.in';
36
37 %%%% We add the path to the directory of the 3 user provided input
       files.
38 eval(['addpath ', user_inputfiles_dir]);
39
40 %%%% We add the path to the top level source code directory.
41 addpath SRC
42 %%%% We add the path to the deeper level source code directories.
43 %%%% We note the functions that need to be called in a typical
       simulation
44 %%%% situation are contained in the top level directory 'SRC'.
45 %%%% The less often used functions are kept in the deeper levels.
46 addpath SRC/PDE SRC/DMRI SRC/FEM SRC/GEOM SRC/TETGEN
```

```
47
48 %%%% We read the params_cells input file and create the geometrical
       configuration
49 %%%% In the first line of the params_cells input file, we expect a
       number
50 %%%% between 1 and 3;
51 %%%% 1 = ellipsoids, 2 = cylinders, 3 = neuron from msh file;
52 [params_cells,fname_cells] = create_geom(fname_params_cells);
53
54 %%%% We read the params_simul_domain input file;
55 [params_domain_geom,params_domain_pde,params_domain_femesh] ...
56     = read_params_simul_domain(fname_params_simul_domain);
57
58 %%%% We set up the PDE model in the geometrical compartments.
59 [DIFF_cmpts,kappa_bdys,IC_cmpts,OUT_cmpts_index,ECS_cmpts_index,
       IN_cmpts_index,Ncmpt,Nboundary] ...
60     = PREPARE_PDE(params_cells.ncell,params_cells.cell_shape,
           params_domain_geom,params_domain_pde);
61
62 %%%% We make a directory to store the finite elements mesh for this
       simulation;
63 save_meshdir_path = [fname_cells,'_dir'];
64 tf = isfolder(save_meshdir_path);
65 if (~tf)
66     call_cmd = ['mkdir ',save_meshdir_path];
67     disp(call_cmd);
68     eval([call_cmd]);
69 end
70
71 %%%% We use an existing finite elements mesh or we create a new finite
72 %%%% elements mesh.
73 %%%% The name of the finite elements mesh is stored in the string
       fname_tetgen_femesh
74 ns = regexp(fname_cells,'/');
75 save_mesh_name = [fname_cells(ns(end)+1:end),'Htetgen',num2str(
       params_domain_femesh.Htetgen),'msh'];
76 fname_tetgen = [save_meshdir_path,'/',save_mesh_name];
77 fname_tmp = [fname_tetgen,'.1'];
78 tf = isfile([fname_tmp,'.node']);
79 if (tf)
80     fname_tetgen_femesh = fname_tmp;
81 else
82     [fname_tetgen_femesh] = ...
83         create_femesh_fromcells(params_cells,fname_cells,
               params_domain_geom,params_domain_femesh,fname_tetgen);
84 end
85
86 %%%% We do not deform the finite elements mesh if in the Neuron module.
87 %%%% We do deform the finite elements mesh if in SpinDoctor White
       Matter
88 %%%% mode.
89 %%%% The finite elements mesh is stored in mymesh
90 if (params_cells.cell_shape == 3)
91     params_cells.para_deform = [0,0]';
92     [mymesh,cmpts_bdys_mat] = read_tetgen(fname_tetgen_femesh,
           params_cells.para_deform,Ncmpt,Nboundary);
93 else
```

```
94      [mymesh,cmpts_bdys_mat] = read_tetgen(fname_tetgen_femesh,
            params_cells.para_deform,Ncmpt,Nboundary);
95  end
96
97  %%%% We plot the finite elements mesh
98  if (DO_PLOTS)
99      PLOT_FEMESH(mymesh,OUT_cmpts_index,ECS_cmpts_index,IN_cmpts_index);
100 end
101
102 %%%% We read the params_simul_experi input file;
103 [experi_common,experi_hadc,experi_btpde] ...
104     = read_params_simul_experi(fname_params_simul_experi);
105
106 %%%% We get the volume and the surface area quantities from the mesh
107 [VOL_cmpts,SA_cmpts,SAu_cmpts,VOL_allcmpts,VF_cmpts,SoV_cmpts] ...
108     = GET_VOL_SA(mymesh,experi_common.gdir);
109
110 %%%% Instead of using 0 as the minimum of the eigenvalue interval we
        use
111 %%%% -infinity to make sure we get 0 as an output eigenvalue.
112 EigLim_min(1:Ncmpt) = -Inf;
113 %%%% The upper bound of the eigenvalue interval is determined by
114 %%%% LSCALE_MIN, which is the minimum length scale of interest.
115 %%%% This minimum length gives the upper bound of the eigenvalue
        interval
116 %%%% by using the formula that links the eigenvalues of the line
        segment
117 %%%% and the wavelength of the eigenfunctions on the line segment.
118 EigLim(1:Ncmpt) = DIFF_cmpts.*(pi^2./(LSCALE_MIN.^2));
119
120 %%%% Breaking the requested eigenvalue interval into a number of
121 %%%% subintervals so that Matlab matrix eigenvalue problem can converge
122 %%%% quickly.  The optimal choice
123 %%%% is when there are not more than 10 or 20 eigenvalues in each
124 %%%% subinterval.
125 EigIntervals(1:Ncmpt) = NUM_EIGINTERVALS;
126
127 for icmpt = 1:Ncmpt
128     [EIG_value_cmpts{icmpt},EIG_proj_cmpts{icmpt},EIG_func_cmpts{icmpt
            },ctime_laplace(1,icmpt)]...
129         = LAPLACE_EIG(mymesh.Pts_cmpt_reorder{icmpt},mymesh.
                Ele_cmpt_reorder{icmpt},...
130         DIFF_cmpts(icmpt),VOL_cmpts(icmpt),EigLim(icmpt),EigLim_min(
                icmpt),EigIntervals(icmpt));
131 end
132
133 %%%% Converting eigenvalues into a length scale according to the
134 %%%% eigenvalues of the line segment and their connection to the
        wavelength
135 %%% of the Laplace eigenfunctions on the line segment.
136 [EIG_length_cmpts] = MF_EIG_TO_LENGTH(EIG_value_cmpts,DIFF_cmpts);
137
138 experi_mf = experi_btpde;
139 nb = size(experi_mf.bvalues,2);
140 nexperi = size(experi_mf.bvalues,1);
141
142 %%%% Computing the JN value that relates the eigenmodes to their
```

```
%%%% contribution to the Matrix Formalism signal for a diffusion-encoding
%%%% sequence.  Note, only PGSE is implemented at this point.
[MF_JN_cmpts] = MF_JN(EIG_value_cmpts,DIFF_cmpts,experi_mf);

% We compute the Matrix Formalism effective diffusion tensor
[DTENSOR_cmpts,DTENSOR_allcmpts] = MF_DTENSOR(VOL_cmpts,...
    IC_cmpts,EIG_value_cmpts,EIG_proj_cmpts,DIFF_cmpts,MF_JN_cmpts);

if (DO_PLOTS ~= 0)
    %%%% We plot the Matrix Formalism effective diffusion tensor
    PLOT_DTENSOR(DTENSOR_cmpts,DTENSOR_allcmpts,DIFF_cmpts);
    cmpts_use = 1;
    %%%% We plot the eigenfunction index by nindex
    for nindex = 2
        diffdir = EIG_proj_cmpts{cmpts_use}(:,nindex);
        title_str = ['EigFun, l_s = ', num2str(EIG_length_cmpts{1}(
            nindex))...
            ', a_{1n} = [',num2str(diffdir'),']^T'];
        PLOT_EIGENFUNCTION(mymesh,EIG_func_cmpts,OUT_cmpts_index,
            ECS_cmpts_index,IN_cmpts_index,...
            nindex,title_str);
    end
end

%%%% We run the simulation for many diffusion-encoding directions
    uniformly
%%%% distributed in the unit sphere in 3 dimensions.
if (experi_common.ngdir_total > 1)

    ngdir_total = experi_common.ngdir_total;
    %%%% We obtain the multiple diffusion-encoding directions uniformly
    %%%% distributed in the unit sphere in 3 dimensions.
    [points_gdir,graddir_index,negii] = HARDI_PTS(ngdir_total);

    if (~isempty(experi_btpde))
        %%%% We solve the BTPDE in many diffusion directions.
        [SIG_BTPDE_cmpts_hardi,SIG_BTPDE_allcmpts_hardi,
            ctime_btpde_hardi] ...
            = SIG_BTPDE_HARDI(experi_btpde,mymesh,DIFF_cmpts,kappa_bdys
                ,IC_cmpts,...
            points_gdir,graddir_index,negii);
    end

    experi_mf = experi_btpde;
    nb = size(experi_mf.bvalues,2);
    nexperi = size(experi_mf.bvalues,1);

    %%%% We compute the Matrix Formalism signal
    [SIG_MF_cmpts_hardi,SIG_MF_allcmpts_hardi,ctime_mf_hardi] ...
        = SIG_MF_HARDI(experi_mf,VOL_cmpts,IC_cmpts,EIG_value_cmpts,
            EIG_proj_cmpts,...
        points_gdir,graddir_index,negii);

    %%%% We compute the Matrix Formalism Gaussian Approximation signal
    [SIG_MFGA_cmpts_hardi,SIG_MFGA_allcmpts_hardi,ctime_mfga_hardi] ...
        = SIG_MFGA_HARDI(experi_mf,VOL_cmpts,IC_cmpts,DTENSOR_cmpts,...
        points_gdir,graddir_index,negii);
```

```
194
195     if (DO_PLOTS ~= 0)
196         %%%% We plot the normalized signal in many diffusion directions
                .
197         cmpts_use = 1;
198         experi_use = unique([1,nexperi]);
199         bvec_use = unique([1,nb]);
200         for iexperi = experi_use
201             for ib = bvec_use
202                 S0 = sum((IC_cmpts.*VOL_cmpts));
203                 bv = experi_btpde.bvalues(iexperi,ib);
204                 title_str = ['BTPDE. Experi ',...
205                     num2str(iexperi),', b=',num2str(bv)];
206                 PLOT_HARDI_PT(points_gdir,...
207                     squeeze(real(SIG_BTPDE_allcmpts_hardi(:,iexperi,ib)
                            ))/S0,title_str);
208
209                 bv = experi_mf.bvalues(iexperi,ib);
210                 title_str = ['MF. Experi ',...
211                     num2str(iexperi),', b=',num2str(bv)];
212                 PLOT_HARDI_PT(points_gdir,...
213                     squeeze(real(SIG_MF_allcmpts_hardi(:,iexperi,ib)))/
                            S0,title_str);
214
215                 bv = experi_mf.bvalues(iexperi,ib);
216                 title_str = ['MFGA. Experi ',...
217                     num2str(iexperi),', b=',num2str(bv)];
218                 PLOT_HARDI_PT(points_gdir,...
219                     squeeze(real(SIG_MFGA_allcmpts_hardi(:,iexperi,ib))
                            )/S0,title_str);
220             end
221         end
222     end
223
224 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] 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.

[6] 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

[7] 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.

[8] 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

[9] 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).

[10] 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.

[11] 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.

[12] D. V. Nguyen, J.-R. Li, D. Grebenkov, D. Le 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.
URL http://www.sciencedirect.com/science/article/pii/S0021999114000308

[13] 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

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

[15] 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

[16] 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

[17] 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

[18] V.-D. Nguyen, C. Fang, D. Wassermann, J.-R. Li, Diffusion MRI simulation of realistic neurons with SpinDoctor and the Neuron Module, arXiv:1910.07916. Submitted.