Journal of Magnetic Resonance 309 (2019) 106611

Contents lists available at ScienceDirect

Journal of Magnetic Resonance

journal homepage: www.elsevier.com/locate/jmr

Portable simulation framework for diffusion MRI

Van-Dang Nguyen [a,*], Massimiliano Leoni [a], Tamara Dancheva [b,a], Johan Jansson [a], Johan Hoffman [a], Demian Wassermann [c], Jing-Rebecca Li [d]

[a] *Division of Computational Science and Technology, KTH Royal Institute of Technology, Sweden*
[b] *Basque Center for Applied Mathematics (BCAM), Bilbao, Spain*
[c] *Parietal, INRIA, Paris, France*
[d] *INRIA Saclay-Equipe DEFI, CMAP, Ecole Polytechnique Route de Saclay, 91128 Palaiseau Cedex, France*

ARTICLE INFO

Article history:
Received 1 August 2019
Revised 8 September 2019
Accepted 19 September 2019
Available online 23 September 2019

Keywords:
Cloud computing
Diffusion MRI
Bloch-Torrey equation
Interface conditions
Pseudo-periodic boundary conditions
FEniCS

ABSTRACT

The numerical simulation of the diffusion MRI signal arising from complex tissue micro-structures is helpful for understanding and interpreting imaging data as well as for designing and optimizing MRI sequences. The discretization of the Bloch-Torrey equation by finite elements is a more recently developed approach for this purpose, in contrast to random walk simulations, which has a longer history. While finite element discretization is more difficult to implement than random walk simulations, the approach benefits from a long history of theoretical and numerical developments by the mathematical and engineering communities. In particular, software packages for the automated solutions of partial differential equations using finite element discretization, such as FEniCS, are undergoing active support and development. However, because diffusion MRI simulation is a relatively new application area, there is still a gap between the simulation needs of the MRI community and the available tools provided by finite element software packages. In this paper, we address two potential difficulties in using FEniCS for diffusion MRI simulation. First, we simplified software installation by the use of FEniCS containers that are completely portable across multiple platforms. Second, we provide a portable simulation framework based on Python and whose code is open source. This simulation framework can be seamlessly integrated with cloud computing resources such as Google Colaboratory notebooks working on a web browser or with Google Cloud Platform with MPI parallelization. We show examples illustrating the accuracy, the computational times, and parallel computing capabilities. The framework contributes to reproducible science and open-source software in computational diffusion MRI with the hope that it will help to speed up method developments and stimulate research collaborations.

1. Introduction

The numerical simulation of the diffusion MRI signal arising from complex tissue micro-structures is helpful for understanding and interpreting imaging data as well as for designing and optimizing MRI sequences. It can be classified into two main groups. The first group is referred to as Monte-Carlo simulations in the literature and previous works include [1–5]. Software packages include the UCL Camino Diffusion MRI Toolkit [6], which has been widely used in the field. The second group of simulations relies on solving the Bloch-Torrey PDE in a geometrical domain, either using finite difference methods (FDM) [7–10], typically on a Cartesian grid, or finite element methods (FEM), typically on a tetrahedral grid. Previous works on FEM include [11] for the short gradient pulse limit of some simple geometries, [12] for the multi-compartment Bloch-Torrey equation with general gradient pulses, and [13] with the flow and relaxation terms added. In [14], a simplified 1D manifold Bloch-Torrey equation was solved to study the diffusion MRI signal from neuronal dendrite trees. FEM in a high-performance computing framework was proposed in [15,16] for diffusion MRI simulations on supercomputers. An efficient simulation method for thin media was proposed in [17]. A comparison of the Monte-Carlo approach with the FEM approach for the short pulse limit was performed in [11], where FEM simulations were evaluated to be more accurate and faster than the equivalent modeling with Monte-Carlo simulations. Recently, SpinDoctor [18], a Matlab-based diffusion MRI simulation toolbox that discretizes the Bloch-Torrey equation using finite elements, was released and shown to be hundreds of times faster than Monte-Carlo based

* Corresponding author.
E-mail addresses: vdnguyen@kth.se (V.-D. Nguyen), jjan@kth.se (J. Jansson), jhoffman@kth.se (J. Hoffman), demian.wassermann@inria.fr (D. Wassermann), jingrebecca.li@inria.fr (J.-R. Li).

https://doi.org/10.1016/j.jmr.2019.106611
1090-7807/© 2019 Elsevier Inc. All rights reserved.

simulations at the same level of accuracy for other diffusion sequences.

The discretization of the Bloch-Torrey equation by finite elements is a more recently developed approach for the purpose of dMRI simulations, in contrast to random walk simulations, which have a longer history. While finite element discretization is more difficult to implement than random walk simulations, the approach benefits from long-established theoretical and numerical developments by the mathematical and engineering communities. In particular, software packages for the automated solutions of partial differential equations using finite element discretization, such as FEniCS [19,20], are subject to active support and development. However, because diffusion MRI simulation is a relatively new application area, there is still a gap between the simulation needs of the MRI community and the available tools built on top of these finite element software packages.

The deployment of FEniCS containers [21] opens a new direction to improve productivity and sharing in the scientific computing community. In particular, it can dramatically improve the accessibility and usability of high-performance computing (HPC) systems. In this paper, we address two potential difficulties in using FEniCS for diffusion MRI simulation. First, we simplified software installation by the use of FEniCS containers that are completely portable across multiple platforms. Second, we provide a simulation framework written in Python and whose code is open source. This simulation framework can be seamlessly integrated with cloud computing resources such as Google Colaboratory notebooks (working on a web browser) or with Google Cloud Platform with MPI parallelization.

One of the advantages of the simulation framework we propose here over the Matlab-based SpinDoctor [18] is that the Python code is free, whereas SpinDoctor requires the purchase of the software Matlab. Many researchers are now adopting Python since it is a free, cross-platform, general-purpose and high-level programming language. Plenty of Python scientific packages are available with extensive documentation such as SciPy for fundamentals of scientific computing, NumPy for large and multi-dimensional arrays and matrices, SymPy for symbolic computation, IPython for the enhanced interactive console, Pandas for data structures & analysis, Matplotlib for comprehensive 2D plotting. In addition, parallel computing for finite elements is relatively easy to implement within FEniCS, thus, this framework has advantages over SpinDoctor for very large scale problems.

The disadvantage of this simulation framework compared to SpinDoctor is the current lack of high-order adaptive time-stepping methods in Python tailored to the kind of ODEs systems coming from finite element discretization, whereas such time-stepping methods are available in Matlab. Thus, in contrast to SpinDoctor where an adaptive, variable order, time-stepping method is used, the time-stepping method in the proposed framework is the θ-method, with a fixed time step size. The θ-method is second-order accurate if θ is chosen to be $\frac{1}{2}$.

The simulation framework we propose meets the following needs:

1. the specification of an intrinsic diffusion tensor and a T_2-relaxation coefficient in each geometrical compartment;
2. the specification of a permeability coefficient on the interface between the geometrical compartments;
3. the periodic extension of the computational domain (assumed a box);
4. the specification of general diffusion-encoding gradient pulse sequences;
5. the simulation of thin-layer and thin-tube media using a discretization on manifolds.

Since this framework is based on FEniCS, packaged as an image, it inherits all functionalities of FEniCS related to mesh generation, mesh adaptivity, finite element matrices construction, linear system solve, solution post-processing and display, as well as the underlying FEniCS computational optimization related to the above tasks. Finally, the framework is conceived with cloud computing and high performance computing in mind, thus, it.

1. supports Cloud Computing with Google Colaboratory and Google Cloud Platform;
2. allows for MPI parallelization.

The paper is organized as follows. In Section 2 we recall the diffusion MRI simulation model based on the Bloch-Torrey equation. Then, we propose a portable simulation framework in Section 3 for which the numerical validation and the comparison are carried out in Section 4. Several simulations examples are shown in Section 5. We share some perspectives about the proposed framework in Section 6. The paper is finalised with a conclusion in Section 7.

2. Theory

The evolution of the complex transverse magnetization $U(\boldsymbol{x},t)$ over time t can be described by the Bloch-Torrey equation [22]. For simplicity we consider a medium composed of two compartments, $\Omega = \Omega_0 \cup \Omega_1$, each of which may be disconnected (see Fig. 1a). The equation takes the following form

$$\frac{\partial U(\boldsymbol{x},t)}{\partial t} = -i\gamma f(t)\boldsymbol{g}\cdot\boldsymbol{x}U(\boldsymbol{x},t) - \frac{U(\boldsymbol{x},t)}{T_2(\boldsymbol{x})} + \nabla\cdot(\boldsymbol{D}(\boldsymbol{x})\nabla U(\boldsymbol{x},t)), \qquad (1)$$

where i is the complex unit ($i^2 = -1$), $\gamma = 2.67513\times10^8$ rad s^{-1} T^{-1} is the gyromagnetic ratio of the water proton, and $\boldsymbol{g}$ is the diffusion gradient including gradient strength $g = \|\boldsymbol{g}\|$ and gradient direction

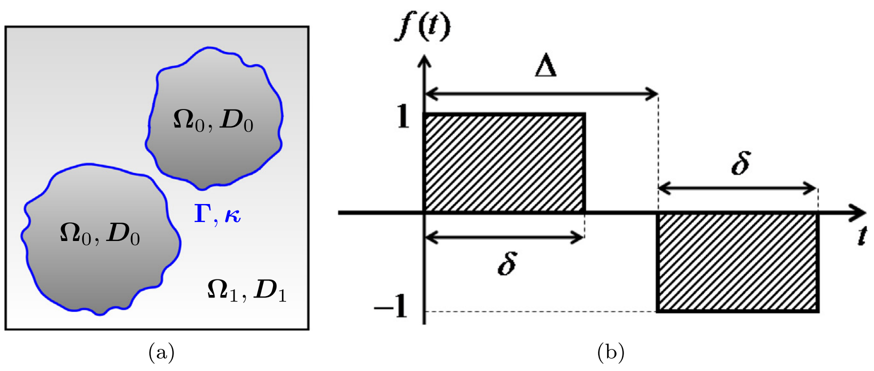


Fig. 1. A composed domain $\Omega = \Omega_0 \cup \Omega_1$ (a), and a PGSE sequence (b).

$\boldsymbol{q} = \frac{\mathbf{g}}{\|\mathbf{g}\|}$. In the general case, $\boldsymbol{D}(\boldsymbol{x})$ is the diffusion tensor, a symmetric positive definite 3×3 matrix. T_2 relaxation is the process by which the transverse magnetization decays or dephases.

On the interface Γ between different compartments the magnetization is allowed to be discontinuous via the use of a permeability coefficient κ [23]

$$\begin{aligned} &[\![\boldsymbol{D}\nabla U \cdot \boldsymbol{n}^0]\!] = 0, \\ &\{\boldsymbol{D}\nabla U \cdot \boldsymbol{n}^0\} = -\kappa [\![U]\!], \end{aligned} \tag{2}$$

for $\boldsymbol{x} \in \Gamma = \partial\Omega_0 \cap \partial\Omega_1$ and $\boldsymbol{n}^k$ is a normal vector to the interface Γ and pointing outward the volume Ω_k. Here $\{\cdot\}$ and $[\![\cdot]\!]$ denote the average and the jump operators defined on the interface Γ, i.e.

$$\{a\} = \frac{a_0 + a_1}{2}, \quad [\![a]\!] = a_0 - a_1.$$

The temporal profile $f(t)$ can vary for different applications and the most commonly used diffusion-encoding sequence in diffusion MRI literature is called the Pulsed-Gradient Spin Echo (PGSE) sequence [24]. For this sequence, one can write $f(t)$ in the following way (see also Fig. 1b):

$$f(t) = \begin{cases} 1, & 0 \leqslant t \leqslant \delta, \\ -1, & \Delta < t \leqslant \Delta + \delta, \\ 0, & \text{otherwise}. \end{cases} \tag{3}$$

The quantity δ is the duration of the diffusion-encoding gradient pulse and Δ is the time delay between the start of the two pulses. Beyond the PGSE, the Oscillating Gradient Spin Echo (OGSE) [25], nonstandard diffusion sequences such as double diffusion encoding [26–29] and multidimensional diffusion encoding [30] can be modelled.

Concerning the boundary conditions (BCs) on the exterior boundaries $\partial\Omega$, there are two options that are very often employed. One is placing the spins to be simulated sufficiently away from $\partial\Omega$ and impose simple BCs on $\partial\Omega$ such as homogeneous Neumann conditions. This supposes that the spins would have a low probability of having arrived at $\partial\Omega$ during the diffusion experiment. Another option is to place the spins anywhere desired, but to assume that Ω is repeated periodically in all space directions to fill $\mathbb{R}^d$, for example, $\Omega = \prod_{k=1}^{d} [a_k, b_k]$. So, one can mimic the phenomenon where the water molecules can enter and exit the computational domain. Under this assumption of periodic continuation of the geometry, the magnetization satisfies pseudo-periodic BCs on $\partial\Omega$ [8]

$$\begin{aligned} &U_m = U_s e^{i\,\theta_k(t)}, \\ &\boldsymbol{D}_m \nabla U_m \cdot \boldsymbol{n} = \boldsymbol{D}_s \nabla U_s \cdot \boldsymbol{n}\, e^{i\,\theta_k(t)}, \end{aligned} \tag{4}$$

where

$$U_m = U(\boldsymbol{x}, t)|_{x_k = a_k}, \quad U_s = U(\boldsymbol{x}, t)|_{x_k = b_k},$$

$$\nabla U_m \cdot \boldsymbol{n} = \nabla U(\boldsymbol{x}, t) \cdot \boldsymbol{n}|_{x_k = a_k}, \quad \nabla U_s \cdot \boldsymbol{n} = \nabla U(\boldsymbol{x}, t) \cdot \boldsymbol{n}|_{x_k = b_k},$$

and

$$\theta_k(t) := \gamma\, g_k (b_k - a_k) \mathcal{F}(t), k = 1, \cdots, d, \quad \mathcal{F}(t) = \int_0^t f(s)\, ds.$$

Here we use 'm' and 's' to indicate master and slave components of the pseudo-periodic BCs. The master-slave method corresponds to the implementation of the periodic BCs [31].

The MRI signal S is the total transverse magnetization $U(\boldsymbol{x}, t)$ over Ω measured at the echo time T

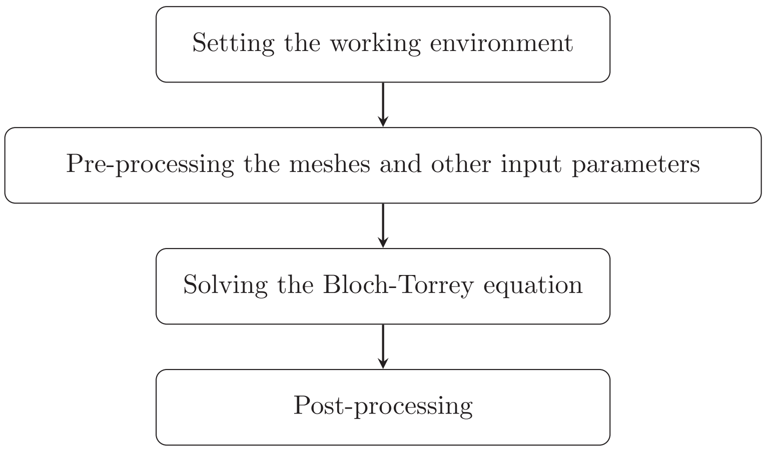
$$S = \int_{\boldsymbol{x} \in \Omega} U(\boldsymbol{x}, T)\, d\boldsymbol{x}. \tag{5}$$

The signal is usually plotted against the gradient strength $g = \|\mathbf{g}\|$ or a quantity called the b-value which is defined as

$$b = \gamma^2 \|\mathbf{g}\|^2 \int_0^T \mathcal{F}(s)^2\, ds. \tag{6}$$

3. Method

For software portability, we consider two container technologies which are Docker [32] and Singularity [33]. They allow for bundling the whole collection of software packages that a user needs in a single file, that can be shared and used by collaborators. This would make a huge impact in scientific applications, where reproducibility is a core concern [34]. In particular, this enables us to develop software that other users can easily test. A software update reduces to a matter of downloading the newest version of a single file and different versions can coexist next to each other for easy consistency checks. We choose Docker for the IPython notebooks and Singularity for the deployment on HPC infrastructure. They follow the same workflow as the following.


Setting the working environment
Pre-processing the meshes and other input parameters
Solving the Bloch-Torrey equation
Post-processing

3.1. Diffusion MRI simulation library

The solution of the Bloch-Torrey equation and other functionalities related to diffusion MRI simulations have been packaged into the Python library `DmriFemLib`, saved in GitHub.https://github.com/van-dang/DMRI-FEM-Cloud/blob/master/DmriFemLib.py.

Due to considerations related to the way FEniCS envisions the PDE solution workflow, and the fact that the PDE from the diffusion MRI simulation problem has some important differences from the typical PDEs for which FEniCS was designed, we made the following choices regarding the implementation of the numerical method that are different than the choices made in the Matlab-based toolbox SpinDoctor. These choices are:

1. the permeable interface conditions are imposed by the use of the partition of unity finite element method (PUFEM) [35,16];
2. the pseudo-periodic BCs coming from the periodic extension of the computational box are imposed on either side of the box face by a PDE transformation;
3. in case of a non-periodic mesh, the necessary pseudo-periodic BCs are imposed by using an artificially permeability coefficient on the box face whose magnitude is inversely proportional to the finite element mesh size [15,16];
4. the implicit Crank-Nicolson method is chosen as the time-stepping method [13]. It is especially important to ensure the stability with the use of the artificial permeability coefficient;

3.2. Mesh generation

Dealing with meshes is one of the most challenging problems in FEM and we inherit what has been done in Python and FEniCS regarding this issue. For simple geometries, one can internally use some built-in meshes. Meshing a box $\Omega = [0, 10]^3$ with given resolutions `nx`, `ny`, `nz` is simply done by the following commands.

```
nx, ny, nz = 10, 10, 10
mesh = BoxMesh(Point(0.0, 0.0, 0.0), Point(10.0, 10.0, 10.0), nx, ny, nz)
```

For more complicated geometries, it is recommended to use `mshr` [36], the mesh generation component of FEniCS, to generate simplicial DOLFIN meshes in 2D and 3D from geometries described by Constructive Solid Geometry or from surface files, utilizing CGAL and Tetgen as mesh generation backends. The commands below are used to generate a two-layered disk:

```
from mshr import *
R1, R2 = 5, 10; origin = Point(0.,0.);
circle = Circle(origin, R1, segments=32)
domain = Circle(origin, R2, segments=32)
domain.set_subdomain(1, circle)
mesh = generate_mesh(domain, 15) # 15 is the resolution
```

More generally, our framework accepts meshes in the DOLFIN XML format [37]. In this paper, the meshes were generated with GMSH [38], Salomé [39], and ANSA [40]. The GMSH script (`.geo`) and Salomé script (`.py`) are available at https://github.com/van-dang/DMRI-FEM-Cloud/tree/mesh and they are distributed through examples discussed later in the paper. GMSH can be embedded in our framework.

```
import os
# define mesh_name ...
os.system('gmsh -3 '+mesh_name+'.geo -o '+mesh_name+'.msh')
```

All the formats need to be converted to DOLFIN XML format by the use of either `dolfin-convert` available with FEniCS or MESHIO [41]. To convert a mesh from `.msh` to `.xml` in a Colaboratory notebook, we just simply call.

```
import os
os.system("dolfin-convert mesh.msh mesh.xml")
```

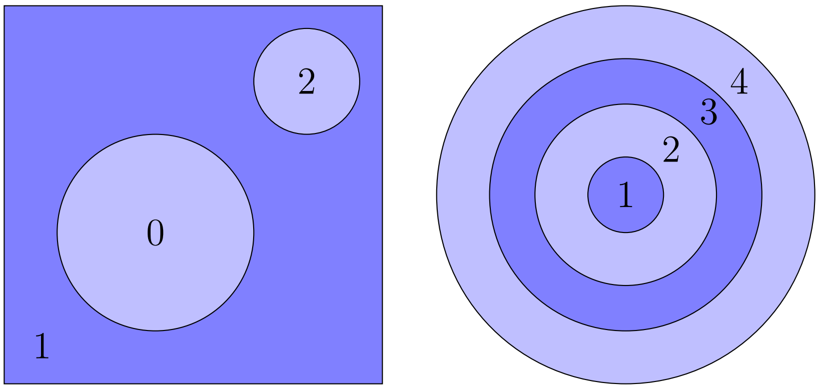


Fig. 2. For multi-compartment domains, the compartments need to be sorted into two groups, `oddgroup` (blue) marked with odd numbers and `evengroup` (light-blue) marked with even numbers which are referred to as the `partition_marker` such that in each group, the compartments should be completely disconnected. It is, therefore, enough to use a `phase` function with two values 0 and 1 to impose the permeability. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

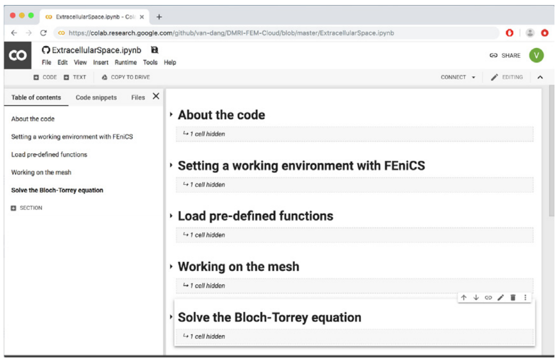

ExtracellularSpace.ipynb - Col
https://colab.research.google.com/github/van-dang/DMRI-FEM-Cloud/blob/master/ExtracellularSpace.ipynb
ExtracellularSpace.ipynb
File Edit View Insert Runtime Tools Help
SHARE
CODE TEXT COPY TO DRIVE
CONNECT
EDITING
Table of contents Code snippets Files
About the code
Setting a working environment with FEniCS
Load pre-defined functions
Working on the mesh
Solve the Bloch-Torrey equation
SECTION
About the code
1 cell hidden
Setting a working environment with FEniCS
1 cell hidden
Load pre-defined functions
1 cell hidden
Working on the mesh
1 cell hidden
Solve the Bloch-Torrey equation
1 cell hidden

Fig. 3. A typical Google Colaboratory notebook for diffusion MRI simulation.

To impose the interface conditions between compartments, we use a `phase` function which is Φ_h in Eq. (A.6). This function initially supports two-compartment domains since it has only two values 0 and 1. To apply for a multi-compartment domain, the compartments need to be sorted into two groups `oddgroup` and `evengroup`. The permeability is imposed at the intermediate interfaces between the two groups. In each group, however, there is no interface between two compartments or in other words, they are completely disconnected (see Fig. 2).

We defined a routine called `CreatePhaseFunc` to create the `phase` function and the `partition_marker`. If the sub-meshes corresponding to compartments are given, these two functions can be created as follows.

```
# Download the meshes
mesh = Mesh("multi_layer_torus.xml")
cmpt_mesh = Mesh("multi_layer_torus_compt1.xml")
evengroup = []
oddgroup = [cmpt_mesh]
phase, partion_list, partition_marker = CreatePhaseFunc(mesh, evengroup,
    ↪ oddgroup, None)
```

For a multi-compartment domain, a `partition_marker` is used to assign each compartment to an identity that allows for defining nonuniform initial conditions, discontinuous diffusion tensors and discontinuous T_2-relaxation. It is a `MeshFunction` in FEniCS defined as the following.

```
partition_marker = MeshFunction("size_t", mesh, mesh.topology().dim())
for cell in cells(mesh):
    partition_marker[cell.index()] = <an identity>;
```

The `partition_marker` can be generated and saved to a DOLFIN XML file, for instance `partition_marker.xml`. The file structure below shows that the elements (cells) with indices of 0 and 1 are assigned with partition marker 3 and 4 respectively.

```
<?xml version="1.0"?>
<dolfin xmlns:dolfin="http://fenicsproject.org">
  <mesh_function>
    <mesh_value_collection type="uint" dim="3" size="6614">
      <value cell_index="0" local_entity="0" value="3" />
      <value cell_index="1" local_entity="0" value="4" />
      ...
    </mesh_value_collection>
  </mesh_function>
</dolfin>
```

In case the partition markers are defined with GMSH by the use of "physical groups", the file can be simply generated by calling our built-in routine `GetPartitionMarkers`

```
GetPartitionMarkers("mesh.msh", "partition_marker.xml")
```

With a given `partition_marker.xml`, the phase function is generated by.

```
File("partition_marker.xml")>>partition_marker
phase, partition_list = CreatePhaseFunc(mesh, [], [] partition_marker)
```

It requires extra care to generate periodic meshes to use the strong imposition of the pseudo-periodic BCs (see A.1). GMSH supports this by `Periodic` mapping which is equivalent to `Projection` routine in Salomé. As part of the framework, we developed the scripts to generate periodic meshes with cells are available in GMSH https://github.com/van-dang/DMRI-FEM-Cloud/blob/mesh/CirclesInSquare.geo and in Salomé https://github.com/van-dang/DMRI-FEM-Cloud/blob/mesh/SpheresInBox.py.

3.3. The main workspace

The workflow is carried out in the main workspace which is either web-based Jupyter notebooks or a script-based interface. The library `DmriFemLib` and other functionalities need to be loaded here.

```
import os
os.system("wget https://raw.githubusercontent.com/van-dang/
    ↪ DMRI-FEM-Cloud/master/DmriFemLib.py")
from DmriFemLib import *
```

3.3.1. *Python notebooks*

Google Colaboratory [42] is a free Jupyter notebook environment that requires no setup and runs entirely in the cloud. It can connect to either a hosted runtime provided by Google Cloud or a local runtime. The hosted runtime allows us to access free resources for up to 12 h at a time and the current one, used to obtain the results presented in this paper, has the following configuration:

- Operating system: Ubuntu 18.04.2 LTS
- Processors: 2 x Intel(R) Xeon(R) CPU @ 2.30 GHz
- RAM: 13 GB

In order to check the configuration you can run the following commands.

```
!cat /proc/meminfo #check RAM memory
!lscpu #check processor
!cat /etc/lsb-release #check distribution
```

Fig. 3 shows a typical structure of our Google Colaboratory notebooks where the simulations can run directly since the setup of the FEniCS environment is done within the notebook.

The installation of FEniCS is quite straightforward in the hosted runtime. The command lines are just the same as the installation on Ubuntu.

```
!apt-get install software-properties-common
!add-apt-repository ppa:fenics-packages/fenics
!sudo apt-get update
!apt-get install --no-install-recommends fenics
from dolfin import *; from mshr import *
```

For longer executions, it is more convenient to connect to the local runtime. To this end, one can execute the following command lines to create a local runtime to which the notebook can connect. This command creates a Docker container from the latest stable FEniCS version at the time of writing, given with the `fenics_tag` variable. Inside this container, we install a Jupyter extension developed by Google Colaboratory's developers and then run a Jupyter notebook from within the container on port `8888`.

```
fenics_tag=2019.1.0.r3 # version of FEniCS image
docker run --name notebook-local -w /home/fenics -v
    ↪ $(pwd):/home/fenics/shared -ti -d -p 127.0.0.1:8888:8888
    ↪ quay.io/fenicsproject/stable:${fenics_tag} "sudo pip install
    ↪ jupyter_http_over_ws; sudo apt-get install -y gmsh; jupyter
    ↪ serverextension enable --py jupyter_http_over_ws;
    ↪ jupyter-notebook --ip=0.0.0.0
    ↪ --NotebookApp.allow_origin='https://colab.research.google.com'
    ↪ --NotebookApp.port_retries=0 --NotebookApp.allow_root=True
    ↪ --NotebookApp.disable_check_xsrf=True --NotebookApp.token=''
    ↪ --NotebookApp.password='' --port=8888"
```

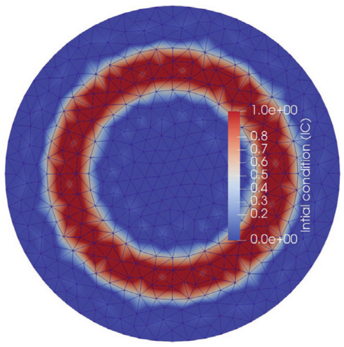
3.3.2. *Script-based interface*

For the script-based interface with parallel executions, the workspace is available at https://github.com/van-dang/DMRI-FEM-Cloud/blob/master/GCloudDmriSolver.py.

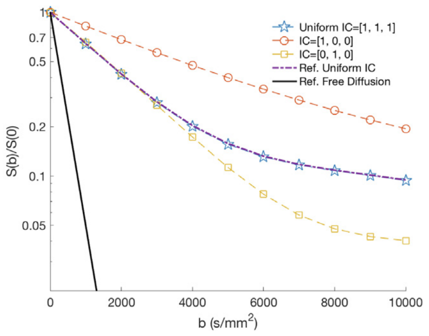
Users can pre-process the inputs for one- and multi-compartment domain by respectively using the functions implemented at https://github.com/van-dang/DMRI-FEM-Cloud/blob/master/PreprocessingOneCompt.py https://github.com/van-dang/DMRI-FEM-Cloud/blob/master/PreprocessingMultiCompt.py.

This workspace can work with Docker image by launching the following command from a Mac or Linux terminal.

```
docker run -ti -p 127.0.0.1:8000:8000 -v $(pwd):/home/fenics/shared -w
    ↪ /home/fenics/shared quay.io/fenicsproject/stable:current
```



(a) Initial conditions [0, 1, 0]



(b) Signals

Fig. 4. Simulations of diffusion inside a three-layered disk of $[5, 7.5, 10]$ μm with different settings of the initial conditions for a PGSE sequence with $\Delta = 43100$ μs and $\delta = 10600$ μs. (a) show the initial conditions with [0, 1, 0]. The signals are strongly dependent on how the initial conditions are set up (b). The accuracy of the simulations is verified by comparing with the reference signal for the uniform distribution of the initial conditions.

However, in the HPC context, Singularity is preferable to Docker due to the security, the accessibility, the portability, and the scheduling issues [33]. Fortunately, it is straightforward to build a Singularity image from a Docker image and for our framework, the command lines are as follows.

```
wget https://raw.githubusercontent.com/van-dang/DMRI-FEM-Cloud/
    ↪ singularity_images/Singularity_recipe_FEniCS_DMRI
sudo singularity build -w writable_fenics_dmri.simg
    ↪ Singularity_recipe_FEniCS_DMRI
```

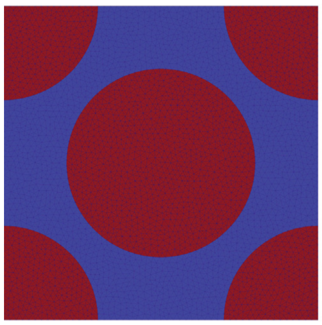
3.3.3. Code structure

Although the interfaces are different between the web-based and the script-based workspaces, they have similar structures with three main classes.

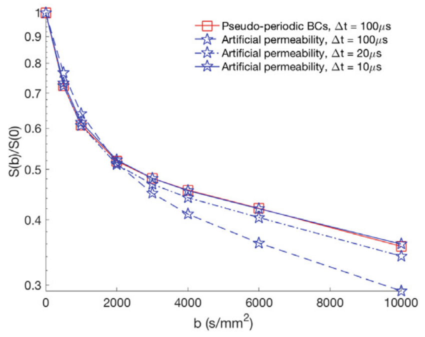
- `MRI_parameters` manages the diffusion pulses such as sequence type, *b*-values, *g*-value and the conversion between them.
- `MRI_domain` manages the finite element meshes, function spaces, domain sizes, diffusion tensors, permeability, and boundary markers.
- `MRI_simulation` manages the initial conditions, time-stepping sizes, linear solvers, solutions, and post-processing.

In `MRI_domain`, the boolean variable `IsDomainMultiple` is used to switch between the single-compartment and the multi-compartment domains. Both strong and weak imposition of the pseudo-periodic BCs have some advantages and disadvantages. The strong imposition works efficiently with periodic meshes with higher accuracy compared to the weak imposition. In some cases, it is, however, not practical to generate periodic meshes. We allow for both options by the use of a boolean variable `IsDomainPeriodic`. When it is `True`, Eq. (A.1) is solved, otherwise, Eq. (1) is solved.

In what follows, we show how to define an arbitrary diffusion sequence and how to use `partition_marker` to define some input parameters on heterogeneous domains.

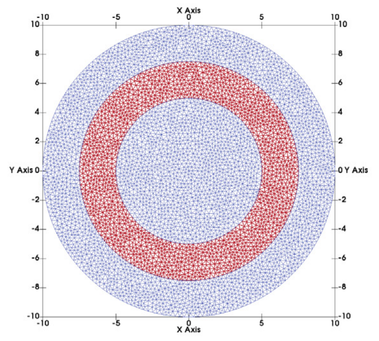


(a) A square box with circular cells

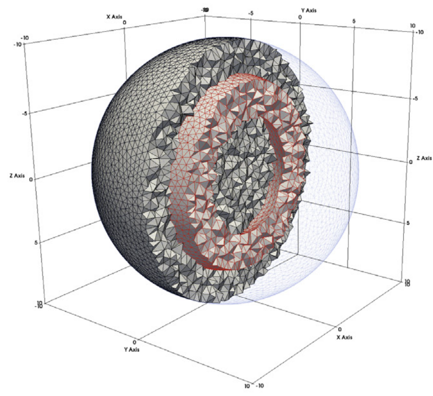


(b) Normalized signals

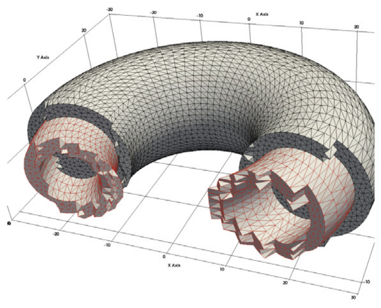
Fig. 5. The artificial permeable method approaches the pseudo-periodic BCs. The time-step size needs to be small to achieve the same accuracy. It is, however, important to recall that the artificial permeable method is useful for non-periodic computational boxes.



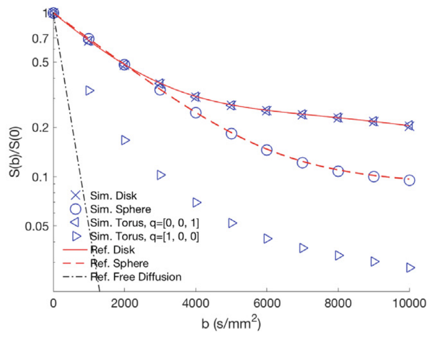
(a) Three-layered disk



(b) Three-layered sphere

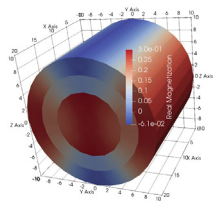


(c) Three-layered torus

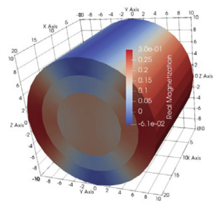


(d) Signals

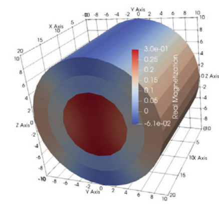
Fig. 6. Three-layered structures and their signals for a PGSE with $\Delta = 43100\ \mu s, \delta = 10600\ \mu s$. The time step size is $\Delta t = 200\ \mu s$.



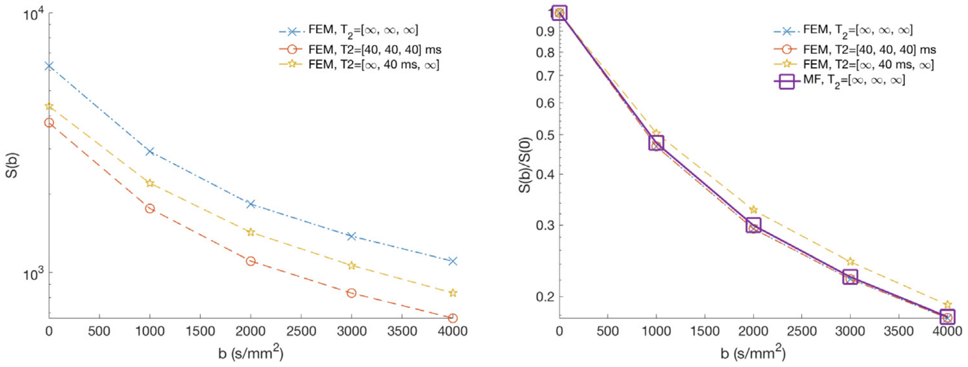
(a) $T_2 = [\infty, \infty, \infty]$



(b) $T_2 = [40\,\text{ms}, \infty, \infty]$

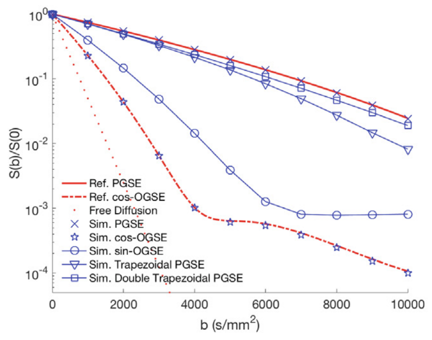


(c) $T_2 = [\infty, 40, 40]\,\text{ms}$

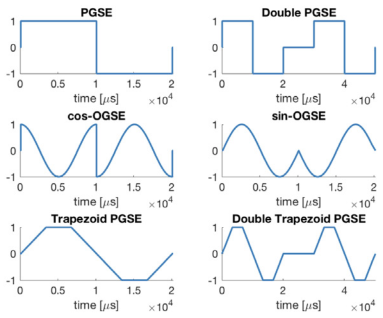


(d) Signals

Fig. 7. T_2-effects of diffusion inside a three-layered cylinder for a PGSE with $\Delta = \delta = 10$ ms, permeability $\kappa = 10^{-5}$ m/s to the magnetization at $b = 4000\ \text{s/mm}^2$ (a, b, c), and to the signals for b between 0 and 4000 s/mm^2 (d).



(a) Signals



(b) Temporal profiles

Fig. 8. Simulated signals inside a disk of radius 5 μm (a) for different temporal profiles: PGSE, Double PGSE, cos-OGSE, sin-OGSE, Trapezoidal PGSE, and Double Trapezoidal PGSE with $\delta = \Delta = 10000$ μs (b). The simulated signals match very well the reference signals for the PGSE and cos-OGSE. The signals with OGSE sequences decay faster compared to the others.

3.3.4. General diffusion-encoding sequence

The framework allows for arbitrary temporal profiles $f(t)$. During the simulation, we need to compute its integral $\mathcal{F}(t) = \int_0^t f(s)\,ds$ and convert between b-value and the gradient strength g-value following Eq. (6). `SymPy` is used to compute the symbolic integration. So, users only need to provide the expression of $f(s)$ to the function member `fs_sym` of the class `MRI_parameters()`. $\mathcal{F}(t)$ and the conversion between b-value, g-value are automatically done. For example, a cos-OGSE sequence

$$f(s) = \begin{cases} \cos(\omega s), & \text{if } s \leqslant \delta, \\ -\cos(\omega(s-\tau)), & \text{if } \tau < s \leqslant \delta + \tau, \\ 0, & \text{otherwise}, \end{cases} \tag{7}$$

with $\omega = \frac{2n\pi}{\delta}, \tau = \frac{\delta+\Delta}{2}$ can be simply defined as the following

```
import sympy as sp
mp = MRI_parameters()
...
mp.delta, mp.Delta = 10000, 10000
mp.T = mp.delta+mp.Delta
mri_para.nperiod = 2
omega = 2.0*mri_para.nperiod*pi/mri_para.delta
tau = mp.T/2.
mri_para.fs_sym = sp.Piecewise(
    (  sp.cos(omega*mri_para.s) ,     mri_para.s <= mri_para.delta ),
    (  0.,                            mri_para.s <= tau ),
    (  -sp.cos(omega*(mri_para.s-tau)), mri_para.s <= mri_para.delta +
       ↪ tau ),
    (  0., True )
)
...
mp.Apply() # F(t) and the conversion between b and q are done here
```

Table 1
Timing in minutes of neuron simulations on Google Colaboratory for a PGSE sequence with $\Delta = 43100$ μs, $\delta = 10600$ μs and different time step sizes Δt.

Neuron	Mesh size	$\Delta t = 50$ μs	$\Delta t = 100$ μs
04b_pyramidal7aACC	615146 vertices	119	64
25o_spindle17aFI	51792 vertices	21	10
03b_pyramidal2aACC	27811 vertices	6	3

3.3.5. Initial conditions

By default, we initialize the spins to be one everywhere. However, it is possible to define discontinuous initial conditions which are illustrated in the following code snippet.

```
mri_simu = MRI_simulation()
mri_para = MRI_parameters()
mymesh = Mesh(...)
mri_domain = MRI_domain(mymesh, mri_para)
...
IC_array = [0, 1, 0];
dofmap_DG = mri_domain.V_DG.dofmap()
disc_ic = Function(mri_domain.V_DG);
for cell in cells(mymesh):
    cmk = partition_marker[cell.index()]
    cell_dof = dofmap_DG.cell_dofs(cell.index())
    disc_ic.vector()[cell_dof] = IC_array[cmk];
disc_ic=project(disc_ic, mri_domain.V)
...
mri_simu.solve(mri_domain, mri_para, linsolver, disc_ic)
```

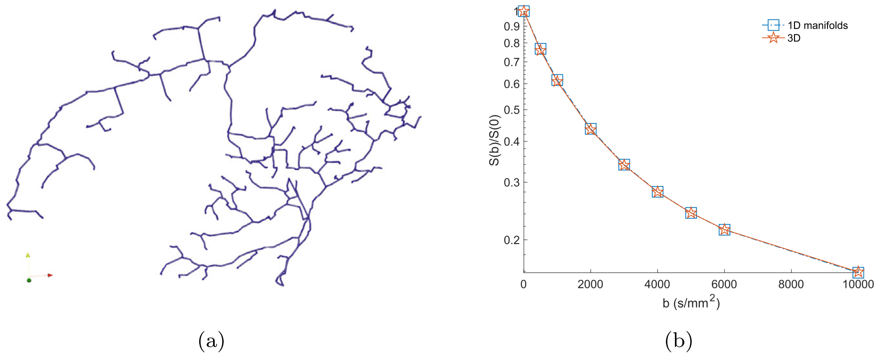


Fig. 9. A comparison between signals inside a neuron from the drosophila melanogaster for a standard 3D mesh and the corresponding 1D manifolds. For $\Delta t = 200$ µs, it costs only 3 s for 1D manifolds but 380 s for 3D to compute the signal for one b-values with the same accuracy.

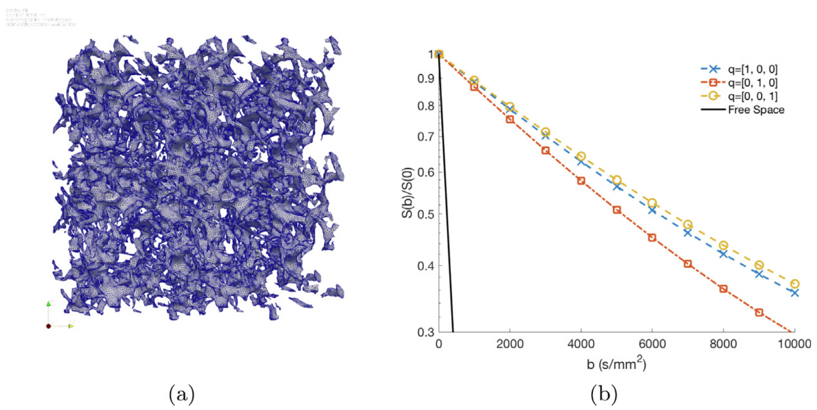


Fig. 10. The mesh of an extracellular space with 462420 vertices and 926058 tetrahedrons (a). It was reconstructed from the medical segmentation published at http://synapseweb.clm.utexas.edu/2013kinney (see also [46]). Two directions in the xz-plane give quite similar signals showing that the domain is quite isotropic in these directions and they both are distinguishable to the signals in the y-direction (b).

3.3.6. Diffusion coefficients and tensors

We allow for a general definition of the diffusion tensor with $d \times d$ components

$$\boldsymbol{D}(\boldsymbol{x}) = \left[d_{jk}(\boldsymbol{x})\right]_{j=1..d,k=1..d}, \tag{8}$$

where $d_{jk}(\boldsymbol{x})$ is cell-based piecewise continuous. We loop through all elements (cells) and the value can be determined by the coordinates of the cell midpoint `p = cell.midpoint()` or a given `partition_marker`.

```
partition_marker = MeshFunction("size_t", mesh, mesh.topology().dim())
# define partition markers
...
D0_array=[3e-3, 1e-3, 3e-3]
# Variable diffusion tensor
V_DG=mri_domain.V_DG; dofmap_DG = V_DG.dofmap();
d00 = Function(V_DG); d01 = Function(V_DG); d02 = Function(V_DG)
d10 = Function(V_DG); d11 = Function(V_DG); d12 = Function(V_DG)
d20 = Function(V_DG); d21 = Function(V_DG); d22 = Function(V_DG)
for cell in cells(mymesh):
    p = cell.midpoint() # the coordinate of the cell center.
    cmk = partition_marker[cell.index()]
    cell_dof = dofmap_DG.cell_dofs(cell.index())
    d00.vector()[cell_dof] = D0_array[cmk];
    d11.vector()[cell_dof] = D0_array[cmk];
    d22.vector()[cell_dof] = D0_array[cmk];
mri_domain.ImposeDiffusionTensor(d00,d01,d02,d10,d11,d12,d20,d21,d22)
```

3.3.7. T_2-relaxation coefficient

By default T_2-relaxation is set to be `1e16`. However, users can define it similarly to the diffusion entry. The following code shows how to define T_2 for a three-compartment domain.

```
T2_array=[4e16, 4e4, 4e4]
dofmap_DG = mri_domain.V_DG.dofmap()
T2 = Function(mri_domain.V_DG);
for cell in cells(mymesh):
    cmk = partition_marker[cell.index()]
    cell_dof = dofmap_DG.cell_dofs(cell.index())
    T2.vector()[cell_dof] = T2_array[cmk];
mri_para.T2 = T2
```

3.4. Solution visualization and post-processing

After solving the Bloch-Torrey equation, the solutions are saved, visualized and the signals are computed following Eq. (5) in a routine called `PostProcessing`. Matplotlib [43] is used for simple visualizations. For more advanced features, Paraview [44] can be externally used for the saved solutions.

4. Numerical validation and comparison

Unless stated otherwise, the simulations were performed for a PGSE with $\Delta = 43100\ \mu s, \delta = 10600\ \mu s$, b-values between 0 and $10000\ s/mm^2$, and the diffusion coefficient of $D = 3 \times 10^{-3}\ mm^2/s$. The membrane between the compartments, if any, is permeable with the permeability of $\kappa = 10^{-5}$ m/s. The simulated signals are compared to the reference ones computed by the matrix formalism (MF) method [45].

We provide a complete simulation method of diffusion inside the multilayered structures such as concentric disks, cylinders, spheres, and torus with the mesh generation software GMSH [38].

First, we study diffusion inside a three-layered disk of $[5, 7.5, 10]$ μm with different settings of the initial conditions. Fig. 4a shows the setting `IC_array=[0, 1, 0]` as discussed in Section 3.3.5. The solver is available at https://colab.research.google.com/github/van-dang/DMRI-FEM-Cloud/blob/master/DiscontinuousInitialCondition.ipynb.

The time step size of $\Delta t = 200$ μs is used. The signals are shown in Fig. 4b where we show that the signals are strongly dependent on how we set up the initial conditions. The accuracy of the simulations is verified by comparing with the reference signal for the uniform distribution of the initial conditions.

To reduce the computational domain, one can assume that the domain is periodically repeated and our framework provides two options for this purpose. With a periodic mesh, it is strongly recommended to solve Eq. (A.1) with periodic boundary conditions (Eq. A.2). However, it is usually challenging to generate such a periodic mesh. So, we also provide an approximation using an artificial permeability coefficient [15,16] (see also in B) for non-periodic meshes. To illustrate this capacity, we consider a square

$\Omega = [-5\ \mu\text{m}, 5\ \mu\text{m}]^2$ including some permeable periodic cells with the permeability $\kappa = 10^{-5}$ m/s (see Fig. 5a). The signals were computed for a PGSE with $\Delta = 13000\ \mu\text{s}, \delta = 10000\ \mu\text{s}, \boldsymbol{q} = \frac{[1,1,0]}{\sqrt{2}}$ (see Fig. 5b). We see that the artificial-permeability method approaches the pseudo-periodic BCs. To achieve the same accuracy, the artificial-permeability method needs $\Delta t = 10\ \mu\text{s}$ which is ten times as smaller as the periodic BCs. The solver is available at https://colab.research.google.com/github/van-dang/DMRI-FEM-Cloud/blob/master/PeriodicDomains.ipynb.

We now consider discontinuous diffusion tensors mentioned in Section 3.3.6 to study diffusion in three-layered structures including a disk, a sphere and a torus whose radii are 5, 7.5 and 10 μm respectively (see Fig. 6a, b, and c). For the torus, the radius from the center of the hole to the center of the torus tube is $R = 20\ \mu\text{m}$. The simulated signals match very well to the reference signals with the time step size of $\Delta t = 200\ \mu\text{m}$ (see Fig. 6d). The Python source code is available at https://colab.research.google.com/github/van-dang/DMRI-FEM-Cloud/blob/master/MultilayeredStructures.ipynb.

The three-layered cylinder is again used to illustrate the effect of T_2-relaxation to the magnetization and the signal attenuation. The gradient direction is $\boldsymbol{q} = [0, 1, 0]$ which is perpendicular to the cylinder axis. As expected, the transverse magnetization decays faster for smaller T_2 (Fig. 7a, b, and c). The signals $S(b)$ are quite different when T_2 varies (Fig. 7d). Here we also show that our signals approximate accurately the reference ones calculated by the matrix formalism (solid curve in the figure) [45]. In short, T_2-relaxation can be used as one of the sources of the image contrast. The Python source code is available at https://colab.research.google.com/github/van-dang/DMRI-FEM-Cloud/blob/master/T2_Relaxation.ipynb.

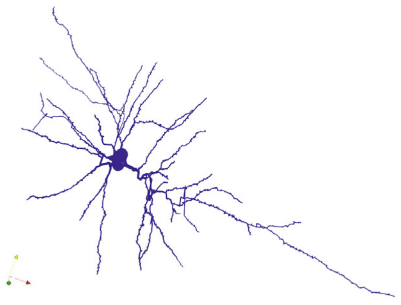
Now, we use the solver to compare the signals inside a disk of radius $5\mu m$ for some temporal profiles: PGSE, Double PGSE, cos-OGSE, sin-OGSE, Trapezoidal PGSE, and Double Trapezoidal PGSE with $\delta = \Delta = 10000\ \mu\text{s}$ (see Fig. 8b). The solver is available at https://colab.research.google.com/github/van-dang/DMRI-FEM-Cloud/blob/master/ArbitraryTimeSequence.ipynb.

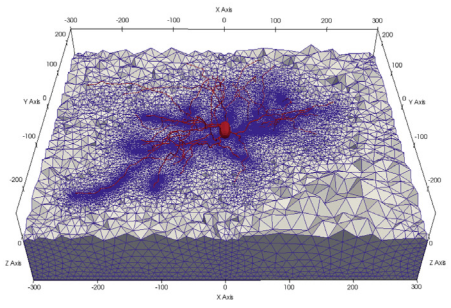
The simulated signals match very well the reference signals for the PGSE and cos-OGSE. The signals with OGSE sequences decay faster compared to the others (see Fig. 8a).

5. Simulation examples

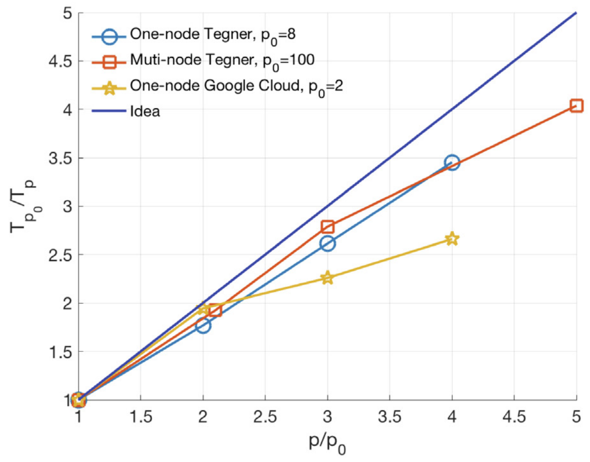
5.1. Realistic neurons

We consider a population of 36 pyramidal and 29 spindle neurons. They are distributed in the anterior frontal insula (aFI) and the anterior cingulate cortex (ACC) of the neocortex of the human brain. They share some morphological similarities such as having a single soma and dendrites branching on opposite sides. This population consists of 20 neurons for each type in aFI, and 9 spindles, 16 pyramidals in ACC. We have published these volume meshes at https://github.com/van-dang/RealNeuronMeshes.


(a) Neuron `04b_pyramidal7aACC`



(b) The mouse neuron embedded in a box



(c) Strong parallel scaling of one-node and multi-node simulations

Fig. 11. The simulation performance is verified for a single computational node on the neuron `04b_pyramidal7aACC` and for multiples nodes on the mouse neuron embedded in a box presented in Section 7.6 in [16]. The strong scaling on Tegner is good both on one node (32 CPUs) and multi-node (scaling up to 500 CPUs). For a small number of cores (2, 4, 6, 8) on Google Cloud, the scaling is less good. It is because the workload and data partition per process are greatly exceeding the ideal work and data per process ratio with the mesh size of 2.5 M tetrahedrons.

The solver is available at https://colab.research.google.com/github/van-dang/DMRI-FEM-Cloud/blob/master/RealNeurons.ipynb.

Table 1 shows the computational time in minutes of neuron simulations on Google Colaboratory for a PGSE sequence with $\Delta = 43100\ \mu s, \delta = 10600\ \mu s, b = 4000\,s/mm^2$ and two different time step sizes $\Delta t = 50, 100\ \mu s$. The relative difference in signals between them is within 4%. With $\Delta = 100\ \mu s$ it costs about an hour for the largest neuron with 615146 vertices whereas it costs only 3 min with a small neuron with 27811 vertices.

In addition to the standard approach using volume elements, we also allow for simulating on manifolds following the method developed in [17]. Fig. 9 shows a comparison between signals inside a neuron from the drosophila melanogaster for a standard 3D mesh and the corresponding 1D manifolds. For $\Delta t = 200\mu s$, it costs only 3 s for 1D manifolds but 380 s for 3D to compute the signal for one b-values with the same accuracy. For more details, it is recommended to look at the solver available at https://colab.research.google.com/github/van-dang/DMRI-FEM-Cloud/blob/master/Manifolds.ipynb.

5.2. Extracellular space

It is challenging to perform simulations on extracellular space (ECS) due to the geometrical complexity. The thickness of ECS is tiny compared to the computational domain. It is extremely time-consuming to use Monte-Carlo approaches. If the reflection condition is applied, the particle undergoes multiple reflections until no further surface intersections are detected, and if the rejection method is applied, the time step sizes need to be very small to be accurate.

In this section, we show that it is efficient to use our framework. We tested with the ECS extracted from the medical segmentation published at http://synapseweb.clm.utexas.edu/2013kinney (see also [46]). The volume mesh is shown in Fig. 10a with 462420 vertices and 926058 tetrahedrons The processed meshes are available in the following link https://github.com/van-dang/DMRI-FEM-Cloud/raw/mesh/2E_ExtraCellular_group_10um_vol.xml.zip.

The Google Colab-based solver is available in the following link https://colab.research.google.com/github/van-dang/DMRI-FEM-Cloud/blob/master/ExtracellularSpace.ipynb.

The timing per b-value is about 30 min on Google Colaboratory for the time discretization $\Delta t = 1$ ms. With half of this time-step size, i.e., $\Delta t = 500\ \mu s$, it takes about an hour for $b = 10000\ s/mm^2$ on Google Colaboratory and the difference in the signals compared to $\Delta t = 1000\ \mu s$ is only 1%. The signals for three principle gradient directions are shown in Fig. 10b. Two directions in the xz-plane give quite similar signals showing that the domain is quite isotropic in these directions and they both are distinguishable from the signals in the y-direction.

5.3. Parallelization

Now we verify the simulation performance with the Singularity image on a 12-month free trial of Google Cloud Platform (https://cloud.google.com) and Tegner (PDC - KTH). The script-based interface is used. It shares the core functionalities with the Python Notebook interface and supports all the functionalities discussed in the paper except the artificial permeability implementation which is still in development. So, the mesh needs to be periodic to have the pseudo-periodic BCs in the parallel execution.

First, we show the simulation performance on one computational node on the neuron `04b_pyramidal7aACC` (Fig. 11a) with the mesh size of about 0.6 M vertices (2.5 M tetrahedrons) and the time discretization of $\Delta t = 200\ \mu s$. A PGSE with $\Delta = 43100\ \mu s, \delta = 10600\ \mu s$ is used. The commands to execute the simulation with the FEniCS image are follows.

```
singularity exec -B $PWD writable_fenics_dmri.simg python3
    ↪ PreprocessingOneCompt.py -o onecompt_files.h5
ListNumProcs="2 4 6  8"    # one-node in Google Cloud
ListNumProcs="5 10 15 20" # one-node in Tegner
for p in ${ListNumProcs}
do
   singularity exec -B $PWD writable_fenics_dmri.simg mpirun -n ${p}
       ↪ python3 GCloudDmriSolver.py -f onecompt_files.h5 -M 0 -b 1000
       ↪ -d 10600 -D 43100 -k 200 -K 3e-3 -gdir 1 0 0
done
```

On Tegner with 20 processors, it costs about 7 min per one b-value whereas on Google Cloud with 8 processors, it costs about 30 min.

Then, we verify with 25 computational nodes on the sample presented in Section 7.6 [16]. The sample consists of a pyramidal neuron of an adult female mouse [47] embedded in the center of a computational domain $\Omega = [-300, 300] \times [-250, 250] \times [-100, 100]\ \mu m^3$ (Fig. 11b). We assume that there is a permeable membrane with $\kappa = 10^{-5}$ m/s between the neuron and the extracellular space. The whole mesh (box + neuron) has about 1.5 M vertices (8.5 M tetrahedrons). The neuron itself consists of 131996 vertices and 431326 tetrahedrons. The whole mesh and sub-mesh (neuron) are available for download at https://github.com/van-dang/DMRI-FEM-Cloud/raw/mesh/volume_box_N_18_7_3_5L_fine.xml.zip https://github.com/van-dang/DMRI-FEM-Cloud/raw/mesh/volume_N_18_7_3_5L_fine.xml.zip.

Below are the commands to execute the simulation with the FEniCS image.

```
singularity exec -B $PWD writable_fenics_dmri.simg python3
    ↪ PreprocessingMultiCompt.py -o multcompt_files.h5
ListNumProcs="100 210 300 500" # 25 nodes in Tegner
for p in ${ListNumProcs}
do
    mpirun -n $p singularity exec -B $PWD writable_fenics_dmri.simg
        ↪ python3 GCloudDmriSolver.py -f multcompt_files.h5 -M 1 -b
        ↪ 1000 -p 1e-5 -d 10600 -D 43100 -k 200 -gdir 0 1 0
```

The simulation with 500 processors costs about 20 min per b-value with $\Delta t = 200\ \mu s$.

The strong parallel scaling is shown in Fig. 11c. Ideally, one expects that the timing is reduced by half when the number of the processors is doubled. Assume that we start with p_0 processors and measure the timing T_{p_0}. Then, we increase the number of the processors to p ($p \geqslant p_0$) and measure the timing T_p. In general, the ideal scaling (the blue curve in Fig. 11c) is

$$\frac{T_{p_0}}{T_p} = \frac{p}{p_0}.$$

We now compare between the realistic scaling of our framework and this ideal linear scaling for $p_0 = 8, 100$ on Tegner and $p_0 = 2$ on Google Cloud. The strong scaling on Tegner is good both on one node (32 CPUs) and multi-node (scaling up to 500 CPUs). For a small number of cores (2, 4, 6, 8) on Google Cloud, the scaling is less good. It is because the workload and data partition per process are greatly exceeding the ideal work and data per process ratio with the mesh size of 2.5 M tetrahedrons.

6. Discussion

The proposed framework can be viewed as the Python version of the Matlab-based SpinDoctor. As verified in [18] for SpinDoctor, this present framework is supposed to be faster and more accurate than Monte-Carlo simulation packages such as Camino. More importantly, our approach benefits from a long history of theoretical and numerical developments by the mathematical and engineering communities. It enhances software reliability which is one of the core concerns in medical applications. Additionally, this framework inherits all of the PDE solution functionalities of FEniCS. Thus, extensions and generalizations of the present dMRI simulation problem, including the coupling with flow, the simulation on deforming domains like the heart, or the coupling of simulations in manifolds with simulations in 3D domains would be straightforward.

We also focused on advanced software engineering features such as portability and parallelization. As other cloud-based software developments, this framework brings reproducible science and open-source software to computational diffusion MRI. It speeds up the method development process since the results are easy to confirm and new methods can be easily developed on top of the existing methods. New algorithms written as Google Colaboratory notebooks can quickly circulate in the MRI community and this allows for active collaboration between research groups. With the parallelization on supercomputers, the simulations lasting weeks can be now reduced to hours or minutes. It enables us to develop, in the near future, real-time simulations of diffusion MRI in which the computer simulation runs at the same rate as the actual physical system.

Since SpinDoctor couples the finite element discretization with optimized adaptive ODE solvers, it is more efficient than our framework in terms of time discretization. The analogous ODE solvers written in Python can be found in the SciPy Library [48] but they are not ready to use within our framework: they do not yet efficiently support the mass matrix and the sparse Jacobian matrix. The lumped mass matrix approach can be used to fix the first issue but more investigations are needed to resolve the latter issue.

Generating finite element meshes from medical segmentation is very challenging. Complicated surface meshes currently need to be processed outside the framework to obtain a good quality finite element mesh. Streamlining this process is an interesting direction of future investigation and it may be well worthwhile to develop algorithms to automate this process.

7. Conclusions

We proposed a portable simulation framework for computational diffusion MRI that works efficiently with cloud technology. The framework can be seamlessly integrated with cloud computing resources such as Google Colaboratory notebooks working on a web browser or with Google Cloud Platform with MPI parallelization. Many simulation needs of the field were addressed by the use of advanced finite element methods for both single- and multi-compartment diffusion domains, with or without permeable membrane and periodic boundaries. We showed the accuracy, the computational times, and parallel computing capabilities through a set of examples, while mentioning straightforward future extensions. The framework contributes to reproducible science and open-source software in computational diffusion MRI. We hope that it will help to speed up method developments and stimulate research collaborations.

Acknowledgement

This research has been supported by the Swedish Energy Agency, Sweden with the project ID P40435-1; MSO4SC with the Grant No. 731063; the Basque Excellence Research Center (BERC 20142017) program by the Basque Government; the Spanish Ministry of Economy and Competitiveness MINECO: BCAM Severo Ochoa accreditation SEV-2013-0323; the ICERMAR ELKARTEK project of the Basque Government; the projects of the Spanish Ministry of Economy and Competitiveness with reference MTM2013-40824-P and MTM2016-76016-R. The research was conducted on resources provided by the Swedish National Infrastructure for Computing (SNIC) at the Center of High- Performance Computing (PDC). We also would like to thank ANSA from Beta-CAE Systems S. A., who generously provided an academic license. The first author would like to thank Niyazi Cem Degirmenci for his enthusiastic supports.

Appendix A

The methods imposed in our framework are based on the partition of unity finite element method (PUFEM) to manage the interface conditions [16]. Both weak and strong implementation of the periodicity with some advantages and disadvantages are included. The θ-method is used for the time discretization.

A.1. Strong implementation of the pseudo-periodic BCs

The complex-valued and time-dependent term in the pseudo-periodic boundaries make it too difficult to implement in a standard FEM software package. So, one can transform the pseudo-periodic BCs to the periodic ones. Following [10,12], one can choose to transform the magnetization to a new unknown $u(\boldsymbol{x},t)$:

$$u(\boldsymbol{x},t) = U(\boldsymbol{x},t)\,e^{i\,\gamma\,\mathcal{F}(t)\,\boldsymbol{g}\cdot\boldsymbol{x}}.$$

The Bloch-Torrey PDE (1) is then transformed to [12]

$$\frac{\partial u}{\partial t} = -i\gamma\mathcal{F}(\boldsymbol{g}\cdot\boldsymbol{D}\nabla u + \nabla u\cdot\boldsymbol{D}\boldsymbol{g}) - (\gamma\mathcal{F})^2\boldsymbol{g}\cdot\boldsymbol{D}\boldsymbol{g}u - \frac{u}{T_2} + \nabla\cdot(\boldsymbol{D}\nabla u), \tag{A.1}$$

with periodic BCs

$$\begin{aligned} &u_m = u_s,\\ &\boldsymbol{D}_m\nabla u_m\cdot\boldsymbol{n} = \boldsymbol{D}_s\nabla u_s\cdot\boldsymbol{n}\end{aligned} \tag{A.2}$$

The homogeneous Neumann boundary condition of U leads to

$$\boldsymbol{D}\nabla u\cdot\boldsymbol{n} = i\gamma\mathcal{F}u\boldsymbol{D}\boldsymbol{g}\cdot\boldsymbol{n}.$$

The interface conditions (Eq. (2)) are changed to

$$\begin{aligned} &[\![\boldsymbol{D}\nabla u\cdot\boldsymbol{n}_0]\!] = 2i\gamma\mathcal{F}\{u\boldsymbol{D}\boldsymbol{g}\cdot\boldsymbol{n}_0\},\\ &\{\boldsymbol{D}\nabla u\cdot\boldsymbol{n}_0\} = -\kappa[\![u]\!] + \tfrac{i\gamma\mathcal{F}}{2}[\![u\boldsymbol{D}\boldsymbol{g}\cdot\boldsymbol{n}_0]\!]\end{aligned} \tag{A.3}$$

Since the magnetization is discontinuous ($m_0 \neq m_1$ on the interface), Eq. (A.3) shows the flux is also discontinuous.

Following the same PUFEM approach proposed in [16], we obtain the following weak form

$$\left(\frac{\partial}{\partial t}u, v\right)_{\Omega_0\cup\Omega_1} = F(u,v,t),$$

where

$$\begin{aligned} F(u,v,t) = &-(i\gamma\mathcal{F}(\boldsymbol{g}\cdot\boldsymbol{D}\nabla u + \nabla u\cdot\boldsymbol{D}\boldsymbol{g}), v)_{\Omega_0\cup\Omega_1}\\ &-\left((\gamma\mathcal{F})^2\boldsymbol{g}\cdot\boldsymbol{D}\boldsymbol{g}u + \frac{u}{T_2}, v\right)_{\Omega_0\cup\Omega_1}\\ &-(\boldsymbol{D}\nabla u,\nabla v)_{\Omega_0\cup\Omega_1} +\\ &< -\kappa[\![u]\!] + \frac{i\gamma\mathcal{F}}{2}[\![u\boldsymbol{D}\boldsymbol{g}\cdot\boldsymbol{n}_0]\!], [\![v]\!] >_\Gamma +\\ &< 2i\gamma\mathcal{F}\{u\boldsymbol{D}\boldsymbol{g}\cdot\boldsymbol{n}_0\}, \{v\} >_\Gamma +\\ &< i\gamma\mathcal{F}u\boldsymbol{D}\boldsymbol{g}\cdot\boldsymbol{n}, v >_{\Gamma_0^N\cup\Gamma_1^N}.\end{aligned} \tag{A.4}$$

We consider a partition of the time domain $0 = t_0 < t_1 < \ldots < t_N = T$ associated with the time intervals $I_n = (t^{n-1}, t^n]$ of length $k^n = t^n - t^{n-1}$ and u^n be an approximation of $u(\boldsymbol{x},t)$ for a given a triangulation $\mathcal{T}^h$ at $t = t^n$.

The PUFEM with the time-stepping θ-method is stated as: Find $u_h^n = (u_{h,0}^n, u_{h,1}^n) \in \boldsymbol{V}_h$ such that

$$\left(\frac{u_h^n - u_h^{n-1}}{k^n}, v_h\right)_{\Omega_0\cup\Omega_1} = \theta F(u_h^n, v_h, t^n) + (1 - \theta)F(u_h^{n-1}, v_h, t^{n-1}), \tag{A.5}$$

for all $v_h = (v_{0,h}, v_{1,h}) \in \boldsymbol{V}_h$, where $(a,b)_{\Omega_{0,h}\cup\Omega_{1,h}} = ((1-\Phi_h)a_0, b_0)_{\Omega_h} + (\Phi_h a_1, b_1)_{\Omega_h}$, and Φ_h is an element-wise constant function:

$$\Phi_h = \begin{matrix} 1, & \text{in } \Omega_{1,h},\\ 0, & \text{in } \Omega_{0,h}.\end{matrix} \tag{A.6}$$

The bilinear and linear forms are defined by

$$\begin{aligned} &\boldsymbol{a}(u_h^n, v_h) = \left(\tfrac{u_h^n}{k^n}, v_h\right)_{\Omega_0\cup\Omega_1} - \theta F(u_h^n, v_h, t^n),\\ &\boldsymbol{L}(v_h) = \left(\tfrac{u_h^{n-1}}{k^n}, v_h\right)_{\Omega_{0,h}\cup\Omega_{1,h}} + (1-\theta)F(u_h^{n-1}, v_h, t^{n-1}).\end{aligned} \tag{A.7}$$

Appendix B. Weak implementation of the pseudo-periodic BCs

The pseudo-periodic BCs (Eq. (4)) can be implemented weakly through the use of an artificial permeability coefficient, κ^e [15,16]. The artificial permeability condition at the external boundaries take two equations for the master side and the slave side of the mesh. For the master side, it has the following form

$$\boldsymbol{D}_m\nabla U_m\cdot\boldsymbol{n}_m = \kappa^e\left(U_s e^{i\theta_{ms}} - U_m\right), \tag{B.1}$$

and for the slave-side it has the following form

$$\boldsymbol{D}_s\nabla u_s\cdot\boldsymbol{n}_s = \kappa^e\left(U_m e^{i\theta_{sm}} - U_s\right), \tag{B.2}$$

where $U_s = U(\boldsymbol{x}_s), U_m = U(\boldsymbol{x}_m), \theta_{ms} = -\theta_{sm} = \gamma\,\boldsymbol{g}\cdot(\boldsymbol{x}_s - \boldsymbol{x}_m)\,\mathcal{F}(t)$. When the master side is considered (Eq. B.1), $\boldsymbol{x}_m$ is the mesh point and $\boldsymbol{x}_s$ is the projection of $\boldsymbol{x}_m$ onto the slave side. Similarly, when the slave side is considered (Eq. B.2), $\boldsymbol{x}_s$ is the mesh point and $\boldsymbol{x}_m$ is the projection of $\boldsymbol{x}_s$ onto the master side. So, the points always align each other but they do not need to be the mesh grid at the same time. So, this method allows for non-matching meshes.

The artificial permeability coefficient κ^e can be chosen to be consistent with the Nitsche's method for the Dirichlet BCs [49] (see also a review in [50] and references therein), i.e $\kappa^e = \max\{\frac{\boldsymbol{D}}{h}\}$ where h is the element size.

To overcome the CFL constraints, the following operator splitting can be used to have an unconditionally stable scheme

$$\begin{aligned} &\boldsymbol{D}_m\nabla U_m\cdot\boldsymbol{n}_m \approx \kappa^e\left(U_s^{n-1}e^{i\theta_{ms}^n} - U_m^n\right),\\ &\boldsymbol{D}_s\nabla U_s\cdot\boldsymbol{n}_s \approx \kappa^e\left(U_m^{n-1}e^{i\theta_{sm}^n} - U_s^n\right),\end{aligned} \tag{B.3}$$

where U^n and U^{n-1} are the approximations at the current and previous time step respectively.

Without imposing the weak pseudo-periodic, the PUFEM with the time-stepping θ-method is stated as: Find $U^n = (U_0^n, U_1^n) \in \boldsymbol{V}^h$ such that

$$\left(\frac{U^n - U^{n-1}}{k^n}, v^h\right)_{\Omega_0\cup\Omega_1} = \theta F(U^n, v^h, t^n) + (1-\theta)F(U^{n-1}, v^h, t^{n-1}), \tag{B.4}$$

for all $v^h = (v_0^h, v_1^h) \in \boldsymbol{V}^h$, where

$$\begin{aligned} F(U,v,t) = &\left(-i\gamma f(t)\boldsymbol{g}\cdot\boldsymbol{x}U - \frac{U}{T_2}, v\right)_{\Omega_0\cup\Omega_1}\\ &-(\boldsymbol{D}\nabla U,\nabla v)_{\Omega_0\cup\Omega_1} - \kappa\langle[\![U]\!],[\![v]\!]\rangle,\end{aligned} \tag{B.5}$$

and $(a,b)_{\Omega_0^h \cup \Omega_1^h} = \left((1-\Phi^h)a_0, b_0\right)_{\Omega^h} + \left(\Phi^h a_1, b_1\right)_{\Omega^h}, \Phi^h$ is an element-wise constant function.

The bilinear and linear forms are defined by

$$\begin{aligned} \boldsymbol{a}(U^n, v^h) &= \left(\tfrac{U^n}{k^n}, v^h\right)_{\Omega_0 \cup \Omega_1} - \theta F(U^n, v^h, t^n), \\ \boldsymbol{L}(v^h) &= \left(\tfrac{U^{n-1}}{k^n}, v^h\right)_{\Omega_0 \cup \Omega_1} + (1-\theta) F\left(U^{n-1}, v^h, t^{n-1}\right). \end{aligned} \tag{B.6}$$

The linear system of equations corresponding to the bilinear and linear forms (Eq. (B.6)) is

$$\boldsymbol{A}\boldsymbol{U}^n = \boldsymbol{F}, \tag{B.7}$$

where

$$\boldsymbol{A} = \boldsymbol{M}(k^n)^{-1} - \theta\left(-\left(i\gamma f^n + \frac{1}{T_2}\right)\boldsymbol{J} - \boldsymbol{S} - \boldsymbol{I}\right). \tag{B.8}$$

Here $\boldsymbol{M}$ and $\boldsymbol{S}$ are referred to as the mass and stiffness matrices respectively, $\boldsymbol{J}$ and $\boldsymbol{I}$ are corresponding to the first and third terms on the right-hand side of F (Eq. (B.5)), i.e $(\boldsymbol{g} \cdot \boldsymbol{x} U, v)$ and $\kappa \langle [\![U]\!], [\![v]\!] \rangle$.

To impose the weak periodic BCs, we plug Eq. (B.3) to the linear and bilinear forms

$$\boldsymbol{a}^*(U_h^n, v_h) = a(U_h^n, v_h) + \theta\kappa^e\left(U_h^n, v_{h_{\Gamma_m^0 \cup \Gamma_m^1}} + \langle U_h^n, v_h \rangle_{\Gamma_s^0 \cup \Gamma_s^1}\right),$$

$$\boldsymbol{L}^*(v_h) = L(v_h) + (1-\theta)\kappa^e\left(U_{s,h}^{n-1} e^{i\theta_{ms}^n}, v_{h_{\Gamma_m^0 \cup \Gamma_m^1}} + U_{m,h}^{n-1} e^{i\theta_{sm}^n}, v_{h_{\Gamma_s^0 \cup \Gamma_s^1}}\right).$$

References

[1] B.D. Hughes, Random Walks and Random Environments, Clarendon Press, Oxford; New York, 1995.

[2] C.-H. Yeh, B. Schmitt, D. Le Bihan, J.-R. Li-Schlittgen, C.-P. Lin, C. Poupon, Diffusion microscopist simulator: a general monte carlo simulation system for diffusion magnetic resonance imaging, PLoS One 8 (8) (2013) e76626, https://doi.org/10.1371/journal.pone.0076626, pONE-D-13-18755[PII]. URL http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3794953/.

[3] M.G. Hall, D.C. Alexander, Convergence and parameter choice for monte-carlo simulations of diffusion mri, IEEE Trans. Med. Imag. 28 (9) (2009) 1354–1364, https://doi.org/10.1109/TMI.2009.2015756.

[4] M. Palombo, C. Ligneul, C. Najac, J. Le Douce, J. Flament, C. Escartin, P. Hantraye, E. Brouillet, G. Bonvento, J. Valette, New paradigm to assess brain cell morphology by diffusion-weighted mr spectroscopy in vivo, Proc. Natl. Acad. Sci. USA 113 (24) (2016) 6671–6676, https://doi.org/10.1073/pnas.1504327113, 201504327[PII] http://www.ncbi.nlm.nih.gov/pmc/articles/PMC4914152/.

[5] K.V. Nguyen, E.H. Garzon, J. Valette, Efficient gpu-based monte-carlo simulation of diffusion in real astrocytes reconstructed from confocal microscopy, J. Magn. Reson. https://doi.org/10.1016/j.jmr.2018.09.013. <http://www.sciencedirect.com/science/article/pii/S1090780718302386>.

[6] P.A. Cook, Y. Bai, S. Nedjati-Gilani, K.K. Seunarine, M.G. Hall, G.J.M. Parker, D.C. Alexander, Camino: Open-source diffusion-mri reconstruction and processing, 2006.

[7] S.N. Hwang, C.-L. Chin, F.W. Wehrli, D.B. Hackney, An image-based finite difference model for simulating restricted diffusion, Magn. Reson. Med. 50 (2) (2003) 373–382, https://doi.org/10.1002/mrm.10536.

[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). <http://view.ncbi.nlm.nih.gov/pubmed/17374905>.

[9] K.D. Harkins, J.-P. Galons, T.W. Secomb, T.P. Trouard, Assessment of the effects of cellular tissue properties on ADC measurements by numerical simulation of water diffusion, Magn. Reson. Med. 62 (6) (2009) 1414–1422, https://doi.org/10.1002/mrm.22155.

[10] G. Russell, K.D. Harkins, T.W. Secomb, J.-P. Galons, T.P. Trouard, A finite difference method with periodic boundary conditions for simulations of diffusion-weighted magnetic resonance experiments in tissue, Phys. Med. Biol. 57 (4) (2012) N35. http://stacks.iop.org/0031-9155/57/i=4/a=N35.

[11] B.F. Moroney, T. Stait-Gardner, B. Ghadirian, N.N. Yadav, W.S. Price, Numerical analysis of NMR diffusion measurements in the short gradient pulse limit, J. Magn. Reson. 234 (2013) 165–175. http://www.sciencedirect.com/science/article/pii/S1090780713001572.

[12] D.V. Nguyen, J.-R. Li, D. Grebenkov, D.L. Bihan, A finite elements method to solve the blochtorrey equation applied to diffusion magnetic resonance imaging, J. Comput. Phys. 263 (Supplement C) (2014) 283–302, https://doi.org/10.1016/j.jcp.2014.01.009, URL http://www.sciencedirect.com/science/article/pii/S0021999114000308.

[13] L. Beltrachini, Z.A. Taylor, A.F. Frangi, A parametric finite element solution of the generalised blochtorrey equation for arbitrary domains, J. Magn. Reson. 259 (Supplement C) (2015) 126–134, https://doi.org/10.1016/j.jmr.2015.08.008, URL http://www.sciencedirect.com/science/article/pii/S1090780715001743.

[14] D.V. Nguyen, J.R. Li, D.S. Grebenkov, D.L. Bihan, Modeling the diffusion magnetic resonance imaging signal inside neurons, J. Phys: Conf. Ser. 490 (1) (2014), 012013 http://stacks.iop.org/1742-6596/490/i=1/a=012013.

[15] V.D. Nguyen, A FEniCS-HPC framework for multi-compartment Bloch-Torrey models, Vol. 1, 2016, pp. 105–119, QC 20170509. <https://www.eccomas2016.org/>.

[16] V.-D. Nguyen, J. Jansson, J. Hoffman, J.-R. Li, A partition of unity finite element method for computational diffusion mri, J. Comput. Phys. 375 (2018) 271–290, https://doi.org/10.1016/j.jcp.2018.08.039. http://www.sciencedirect.com/science/article/pii/S0021999118305709.

[17] V.-D. Nguyen, J. Jansson, H.T.A. Tran, J. Hoffman, J.-R. Li, Diffusion mri simulation in thin-layer and thin-tube media using a discretization on manifolds, J. Magn. Reson. 299 (2019) 176–187, https://doi.org/10.1016/j.jmr.2019.01.002. http://www.sciencedirect.com/science/article/pii/S1090780719300023.

[18] 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), https://doi.org/10.1016/j.neuroimage.2019.116120, 116120 http://www.sciencedirect.com/science/article/pii/S1053811919307116.

[19] A. Logg, K.-A. Mardal, G.N. Wells, Automated solution of differential equations by the finite element method: the FEniCS book, Springer Verlag, 2012, xIII, 723 s.: ill.

[20] FEniCS, Fenics project, <http://www.fenicsproject.org>.

[21] J.S. Hale, L. Li, C.N. Richardson, G.N. Wells, Containers for portable, productive, and performant scientific computing, Comput. Sci. Eng. 19 (6) (2017) 40–50, https://doi.org/10.1109/MCSE.2017.2421459.

[22] H.C. Torrey, Bloch equations with diffusion terms, Phys. Rev. 104 (1956) 563–565, https://doi.org/10.1103/PhysRev. 104.563. https://link.aps.org/doi/10.1103/PhysRev.104.563.

[23] J.E. Tanner, Transient diffusion in a system partitioned by permeable barriers. application to nmr measurements with a pulsed field gradient, J. Chem. Phys. 69 (4) (1978) 1748–1754, https://doi.org/10.1063/1.436751.

[24] E.O. Stejskal, J.E. Tanner, Spin diffusion measurements: spin echoes in the presence of a time-dependent field gradient, J. Chem. Phys. 42 (1) (1965) 288–292, https://doi.org/10.1063/1.1695690.

[25] 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, https://doi.org/10.1002/mrm.10385.

[26] N. Shemesh, S.N. Jespersen, D.C. Alexander, Y. Cohen, I. Drobnjak, T.B. Dyrby, J. Finsterbusch, M.A. Koch, T. Kuder, F. Laun, M. Lawrenz, H. Lundell, P.P. Mitra, M. Nilsson, E. Zarslan, D. Topgaard, C.-F. Westin, Conventions and nomenclature for double diffusion encoding nmr and mri, Magn. Reson. Med. 75 (1) (2016) 82–87, https://doi.org/10.1002/mrm.25901. https://onlinelibrary.wiley.com/doi/pdf/10.1002/mrm.25901.

[27] B. Dhital, M. Reisert, E. Kellner, V.G. Kiselev, Intra-axonal diffusivity in brain white matter, NeuroImage 189 (2019) 543–550, https://doi.org/10.1016/j.neuroimage.2019.01.015. http://www.sciencedirect.com/science/article/pii/S1053811919300151.

[28] D.S. Novikov, E. Fieremans, S.N. Jespersen, V.G. Kiselev, Quantifying brain microstructure with diffusion MRI: theory and parameter estimation, NMR Biomed. 32 (4) (2019), https://doi.org/10.1002/nbm.3998, e3998 https://onlinelibrary.wiley.com/doi/abs/10.1002/nbm.3998.

[29] R.N. Henriques, S.N. Jespersen, N. Shemesh, Microscopic anisotropy misestimation in spherical-mean single diffusion encoding mri, Magn. Reson. Med. 81 (5) (2019) 3245–3261, https://doi.org/10.1002/mrm.27606. https://onlinelibrary.wiley.com/doi/pdf/10.1002/mrm.27606.

[30] D. Topgaard, Multidimensional diffusion mri, J. Magn. Reson. 275 (2017) 98–113, https://doi.org/10.1016/j.jmr.2016.12.007. http://www.sciencedirect.com/science/article/pii/S1090780716302701.

[31] Z. Yuan, J. Fish, Toward realization of computational homogenization in practice, Int. J. Numer. Meth. Eng. 73 (2008) 361–380, https://doi.org/10.1002/nme.2074.

[32] B. Bashari Rad, H. Bhatti, M. Ahmadi, An introduction to docker and analysis of its performance, IJCSNS Int. J. Comput. Sci. Network Secur. 173 (2017) 8.

[33] G.M. Kurtzer, V. Sochat, M.W. Bauer, Singularity: scientific containers for mobility of compute, PLOS ONE 12 (5) (2017) 1–20, https://doi.org/10.1371/journal.pone.0177459.

[34] E. National, Academies of Sciences, Medicine, Reproducibility and Replicability in Science, The National Academies Press, Washington, DC, 2019, doi: 10.17226/25303.

[35] J. Melenk, I. Babuka, The partition of unity finite element method: basic theory and applications, Comput. Methods Appl. Mech. Eng. 139 (1) (1996) 289–314, https://doi.org/10.1016/S0045-7825(96)01087-0. http://www.sciencedirect.com/science/article/pii/S0045782596010870.

[36] B. Kehlet, A. Logg, J. Ring, G.N. Wells, FEniCS project, https://bitbucket.org/fenics-project/mshr/, 2019.

[37] A. Logg, G.N. Wells, Dolfin: automated finite element computing, ACM Trans. Math. Softw. 37 (2) (2010) 20:1–20:28, https://doi.org/10.1145/1731022.1731030.

[38] C. Geuzaine, J.F. Remacle, Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities, Int. J. Numer. Methods Eng.

[39] A. Ribes, C. Caremoli, Salom platform component model for numerical simulation, in: 31st Annual International Computer Software and Applications Conference (COMPSAC 2007), Vol. 2, 2007, pp. 553–564. https://doi.org/10.1109/COMPSAC.2007.185.

[40] Beta cae systems, ansa pre-processor: The advanced cae pre-processing software for complete model build up., <https://www.beta-cae.com>.

[41] N. Schlömer, meshio, <https://github.com/nschloe/meshio>, 2019.

[42] G. Inc., Google colaboratory, <https://github.com/jupyter/colaboratory>, 2014.

[43] J.H. et al., Matplotlib.

[44] K.I. Sandia Corporation, Paraview.

[45] D.S. Grebenkov, Pulsed-gradient spin-echo monitoring of restricted diffusion in multilayered structures, J. Magn. Reson. 205 (2) (2010) 181–195, https://doi.org/10.1016/j.jmr.2010.04.017. http://www.sciencedirect.com/science/article/pii/S1090780710001199.

[46] J.P. Kinney, J. Spacek, T.M. Bartol, C.L. Bajaj, K.M. Harris, T.J. Sejnowski, Extracellular sheets and tunnels modulate glutamate diffusion in hippocampal neuropil, J. Comparat. Neurol. 521 (2) (2013) 448–464, https://doi.org/10.1002/cne.23181. https://onlinelibrary.wiley.com/doi/pdf/10.1002/cne.23181.

[47] L. Carim-Todd, K.G. Bath, G. Fulgenzi, S. Yanpallewar, D. Jing, C.A. Barrick, J. Becker, H. Buckley, S.G. Dorsey, F.S. Lee, L. Tessarollo, Endogenous truncated trkb.t1 receptor regulates neuronal complexity and trkb kinase receptor function in vivo, J. Neurosci. 29 (3) (2009) 678–685, https://doi.org/10.1523/JNEUROSCI.5060-08.2009. http://www.jneurosci.org/content/29/3/678.full.pdf.

[48] S. Developers, Scipy, <https://www.scipy.org>, 2001.

[49] J. Nitsche, Über ein variationsprinzip zur lösung von dirichlet-problemen bei verwendung von teilräumen, die keinen randbedingungen unterworfen sind, Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg 36 (1) (1971) 9–15, https://doi.org/10.1007/BF02995904.

[50] P. Hansbo, Nitsche's method for interface problems in computational mechanics, GAMM-Mitteilungen 28 (2) (2005) 183–206, https://doi.org/10.1002/gamm.201490018.