Recent advances in SpinDoctor: new functionalities, efficiency, and future extensions

Syver Døving Agdestein

Equipe IDEFIX, CMAP, Ecole Polytechnique

February 10, 2021

Table of contents

SpinDoctor – Contributors

Jing-Rebecca Li
Chengran Fang
Demian Wassermann
Van-Dang Nguyen
Try Nguyen Tran
Jan Valdman
Cong-Bang Trang
Khieu Van Nguyen
Duc Thach Son Vu
Hoang An Tran
Hoang Trong An Tran
Thi Minh Phuong Nguyen

SpinDoctor Toolbox

MATLAB toolbox including:

- Geometry generation and handling
- DMRI[1] simulation by solving BTDPE[2], HADC[3], STA[4], HARDI[5][1]
- Neuron Module[2]
- Matrix Formalism module[3]
- Different configurations and options
- Visualization

---

[1] Diffusion Magnetic Resonance Imaging
[2] Bloch-Torrey Partial Differential Equation
[3] Homogenized Apparent Diffusion Coefficient model
[4] Short Time Approximation
[5] High Angular Resolution Diffusion Imaging

Background: Bloch-Torrey PDE (general form)

SpinDoctor solves for the complex transverse water proton magnetization M:

$$\frac{\partial}{\partial t} M(\boldsymbol{x}, t) = -\underline{\mathrm{i}}\gamma f(t) \boldsymbol{g} \cdot \boldsymbol{x}\, M(\boldsymbol{x}, t) + \nabla \cdot (\sigma(\boldsymbol{x}) \nabla M(\boldsymbol{x}, t)) \quad (1)$$

Equation support:

$$(\boldsymbol{x}, t) \in \Omega \times [0, T_{\text{echo}}],$$

Initial spin density:

$$M(\boldsymbol{x}, 0) = \rho(\boldsymbol{x}), \quad \boldsymbol{x} \in \Omega$$

Boundary conditions (Neumann):

$$\sigma(\boldsymbol{x}) \nabla M(\boldsymbol{x}, t) \cdot \boldsymbol{n}(\boldsymbol{x}) = 0, \quad \boldsymbol{x} \in \partial\Omega$$

Model input: magnetic field gradient pulsed sequence

$$f : [0, T_{\text{echo}}] \to [-1, 1]$$

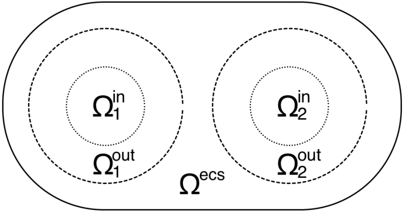
$$\boldsymbol{g} \in \mathbb{R}^3$$

Model output: acquired signal

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

Bloch-Torrey PDE: SpinDoctor assumptions I

Domain consists of N_{cell} cells with or without nuclei and ECS[6]



$$\Omega = \bigcup_{i=1}^{N_{\text{cell}}} \Omega_i^{\text{in}} \cup \Omega_i^{\text{out}} \cup \Omega^{\text{ecs}}$$

$$M(\boldsymbol{x}, t) = \begin{cases} M_i^{\text{in}}(\boldsymbol{x}, t), & \boldsymbol{x} \in \Omega_i^{\text{in}} \\ M_i^{\text{out}}(\boldsymbol{x}, t), & \boldsymbol{x} \in \Omega_i^{\text{out}} \\ M^{\text{ecs}}(\boldsymbol{x}, t), & \boldsymbol{x} \in \Omega^{\text{ecs}} \end{cases},$$

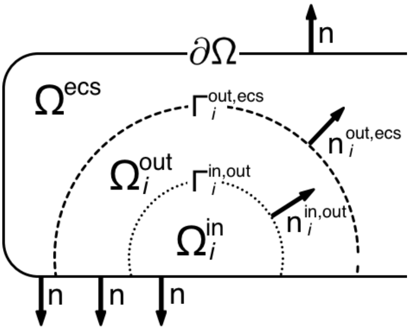
$$\rho(\boldsymbol{x}) = \begin{cases} \rho^{\text{in}}, & \boldsymbol{x} \in \Omega_i^{\text{in}} \\ \rho^{\text{out}}, & \boldsymbol{x} \in \Omega_i^{\text{out}} \\ \rho^{\text{ecs}}, & \boldsymbol{x} \in \Omega^{\text{ecs}} \end{cases}, \quad \sigma(\boldsymbol{x}) = \begin{cases} \sigma^{\text{in}}, & \boldsymbol{x} \in \Omega_i^{\text{in}} \\ \sigma^{\text{out}}, & \boldsymbol{x} \in \Omega_i^{\text{out}} \\ \sigma^{\text{ecs}}, & \boldsymbol{x} \in \Omega^{\text{ecs}} \end{cases},$$

Bloch-Torrey PDE: SpinDoctor assumptions II

Interface conditions

$$\Gamma_i^{\text{in,out}} = \Omega_i^{\text{in}} \cap \Omega_i^{\text{out}},$$

$$\Gamma_i^{\text{out,ecs}} = \Omega_i^{\text{out}} \cap \Omega^{\text{ecs}},$$



Flux continuity and interface permeability ($[\![a]\!]_i^j = a_j - a_i$):

$$
\begin{aligned}
[\![\sigma \nabla M]\!]_{i,\text{in}}^{i,\text{out}} \cdot \boldsymbol{n}_i^{\text{in,out}} &= 0, && \text{on } \Gamma_i^{\text{in,out}} \\
[\![\sigma \nabla M]\!]_{i,\text{out}}^{\text{ecs}} \cdot \boldsymbol{n}_i^{\text{out,ecs}} &= 0, && \text{on } \Gamma_i^{\text{out,ecs}} \\
\sigma^{\text{in}} \nabla M_i^{\text{in}} \cdot \boldsymbol{n}_i^{\text{in,out}} &= \kappa^{\text{in,out}} [\![M]\!]_{i,\text{in}}^{i,\text{out}}, && \text{on } \Gamma_i^{\text{in,out}} \\
\sigma^{\text{out}} \nabla M_i^{\text{out}} \cdot \boldsymbol{n}_i^{\text{out,ecs}} &= \kappa^{\text{out,ecs}} [\![M]\!]_{i,\text{out}}^{\text{ecs}}, && \text{on } \Gamma_i^{\text{out,ecs}}
\end{aligned}
$$

[6] Extra-Cellular Space

Gradient sequences I

SpinDoctor comes with support for the following gradient sequences:

- Pulsed Gradient Spin Echo (PGSE)[4]:

$$f(t) = \mathbb{1}_{[0,\delta]}(t) - \mathbb{1}_{[\Delta,\Delta+\delta]}(t);$$

- Oscillating Gradient Spin Echo (OGSE)[5, 6]:

$$f(t) = \cos\left(\frac{2\pi n}{\delta} t\right) \mathbb{1}_{[0,\delta]}(t) - \cos\left(\frac{2\pi n}{\delta}(t - \Delta)\right) \mathbb{1}_{[\Delta,\Delta+\delta]}(t);$$

- Other time profiles f, including double-PGSE, sin-OGSE and custom time profiles.

The SpinDoctor algorithms are optimized for the most commmon time profiles, and also support arbitrary time profiles, where the integral quantities are computed numerically.

Gradient sequences II

Important quantity: b-value (combined effect of strength and duration of pulses)

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

Free diffusion (with diffusivity σ): signal attenuation is given by

$$\mathrm{e}^{-\sigma b}$$

To account for deviation from free diffusion: replace $\sigma \to$ ADC ("Apparent" Diffusion Coefficient)

$$\mathrm{ADC}\left(f, \frac{\boldsymbol{g}}{\|\boldsymbol{g}\|}\right) = -\frac{\partial}{\partial b} \log \frac{S(f, \boldsymbol{g}(f, b))}{S(f, \mathbf{0})}$$

Matrix Formalism – Laplace eigenvalue decomposition

Find pairs (λ, ϕ) satisfying

$$-\nabla \cdot (\sigma(\boldsymbol{x}) \nabla \phi(\boldsymbol{x})) = \lambda \phi(\boldsymbol{x}), \quad \boldsymbol{x} \in \Omega, \tag{2}$$

with the same conditions on $\Gamma_i^{\text{in,out}}$, $\Gamma_i^{\text{out,ecs}}$ and $\partial\Omega$ as for the BTPDE.

For a line segment with length L and diffusivity σ:

$$\lambda_n = \left(\frac{\pi(n-1)}{L}\right)^2 \sigma, \quad n = 1, 2, \ldots$$

A given eigenvalue λ may thus be associated with a length scale:

$$\ell(\lambda) = \pi \sqrt{\frac{\bar{\sigma}}{\lambda}}, \qquad \bar{\sigma} = \frac{1}{|\Omega|} \int_\Omega \sigma(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x}.$$

Matrix Formalism approximation

Truncate Laplace eigenfunction basis at a level N_{eig}:

$$M^{\text{MF}}(\boldsymbol{x}, t) = \sum_{k=1}^{N_{\text{eig}}} \phi_k(\boldsymbol{x})\nu_k(t) = \boldsymbol{\phi}^{\mathsf{T}}(\boldsymbol{x})\boldsymbol{\nu}(t)$$

where $\boldsymbol{\phi} = (\phi_1, \ldots, \phi_{N_{\text{eig}}})^{\mathsf{T}} \in \mathbb{R}^{N_{\text{eig}}}$ is ordered such that $0 = \lambda_1 < \lambda_2 \leq \lambda_3 \leq \ldots$ and $\boldsymbol{\nu} = (\nu_1, \ldots, \nu_{N_{\text{eig}}})^{\mathsf{T}} \in \mathbb{C}^{N_{\text{eig}}}$. Then $\boldsymbol{\nu}$ is solution to

$$\frac{\partial \boldsymbol{\nu}}{\partial t} = \left(\mathbf{L} + \underline{\mathrm{i}}\gamma f(t)\mathbf{A}(\boldsymbol{g})\right)\boldsymbol{\nu}(t),$$

where $\mathbf{L} = \operatorname{diag}(\lambda_1, \ldots, \lambda_{N_{\text{eig}}})$, $\mathbf{A}(\boldsymbol{g}) = \int_\Omega \boldsymbol{g} \cdot \boldsymbol{x}\boldsymbol{\phi}(\boldsymbol{x})\boldsymbol{\phi}^{\mathsf{T}}(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}$, and $\boldsymbol{\nu}(0) = \int_\Omega \rho(\boldsymbol{x})\boldsymbol{\phi}(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}$.

Matrix Formalism approximation – PGSE sequence

For PGSE, the Bloch-Torrey operator in the Laplace basis is constant:

$$\mathbf{K}(\boldsymbol{g}) = \mathbf{L} + \underline{\mathrm{i}}\gamma \mathbf{A}(\boldsymbol{g}) \in \mathbb{R}^{N_{\text{eig}} \times N_{\text{eig}}}.$$

The final magnetization is given by

$$M^{\mathrm{MF}}(\boldsymbol{x}, T_{\text{echo}}) = \boldsymbol{\phi}^{\mathsf{T}}(\boldsymbol{x}) \mathrm{e}^{-\delta \mathbf{K}^*} \mathrm{e}^{-(\Delta - \delta)\mathbf{L}} \mathrm{e}^{-\delta \mathbf{K}} \boldsymbol{\nu}(0)$$

where * denotes the complex conjugate transpose, as opposed to $^{\mathsf{T}}$.

Matrix Formalism approximation – general case

For an *arbitrary* time profile f, the Bloch-Torrey operator is time-dependent:

$$\mathbf{K}(f, \boldsymbol{g})(t) = \mathbf{L} + \underline{\mathrm{i}}\gamma f(t)\mathbf{A}(\boldsymbol{g}),$$

Using a piece-wise constant approximation[7] of the time profile:

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

where $\{\mathcal{I}_i\}_{i=1,\dots,N_{\text{int}}}$ are intervals such that $[0, T_{\text{echo}}] = \bigcup_{i=1}^{N_{\text{int}}} \mathcal{I}_i$, $f(t) = f_i$ for $t \in \mathcal{I}_i$, $\delta_i = |\mathcal{I}_i|$, and $\mathbf{K}_i(\boldsymbol{g}) = \mathbf{L} + \underline{\mathrm{i}}\gamma f_i \mathbf{A}(\boldsymbol{g})$.
To compute the constants: quadrature

$$f_i = \frac{1}{\delta_i} \int_{\mathcal{I}_i} f(t)\,\mathrm{d}t \approx \frac{1}{2}(f(\min \mathcal{I}_i) + f(\max \mathcal{I}_i))$$

Finite Element discretization I

In SpinDoctor, we manually construct the finite element problem, based on [8].

Divide the domain Ω into N_{node} points $\boldsymbol{q}_1, \cdots, \boldsymbol{q}_{N_{\text{node}}} \in \mathbb{R}^3$ and N_{element} tetrahedra. Piece-wise linear (P_1-elements) basis functions $\boldsymbol{\varphi} = (\varphi_1, \ldots, \varphi_{N_{\text{node}}})^{\mathsf{T}} \in \mathbb{R}^{N_{\text{node}}}$ are defined by:

$$\varphi_j(\boldsymbol{q}_k) = \delta_{jk} \quad \text{(Kronecker symbol)},$$

linear on each tetrahedron that touches node j. The nodes include double nodes on $\Gamma_i^{\text{in,out}}$ and $\Gamma_i^{\text{out,ecs}}$, for which there is one basis function for each side of each node. Their shared support lies on $\Gamma_i^{\text{in,out}}$ or $\Gamma_i^{\text{out,ecs}}$, which is of measure zero in 3D ("$\mathrm{d}\boldsymbol{x}$"), but not in 2D ("$\mathrm{d}\Gamma$").

Finite Element discretization II

Mass matrix:

$$\mathbf{M} = \int_\Omega \boldsymbol{\varphi}(\boldsymbol{x})\boldsymbol{\varphi}^\mathsf{T}(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}$$

Stiffness matrix:

$$\mathbf{S} = \int_\Omega \sigma(\boldsymbol{x})\nabla\boldsymbol{\varphi}(\boldsymbol{x})\,(\nabla\boldsymbol{\varphi})^\mathsf{T}(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} \quad (\nabla\boldsymbol{\varphi} \in \mathbb{R}^{N_\text{node}\times 3})$$

Flux matrix (for $\Gamma = \bigcup_{i=1}^{N_\text{cell}} \Gamma_i^\text{in,out} \cup \Gamma_i^\text{out,ecs}$):

$$\mathbf{Q} = \int_\Gamma \kappa(\boldsymbol{x})\tilde{\boldsymbol{\varphi}}(\boldsymbol{x})\tilde{\boldsymbol{\varphi}}^\mathsf{T}(\boldsymbol{x})\,\mathrm{d}\Gamma, \quad \tilde{\varphi}_j = \begin{cases} -\varphi_j & j \text{ out-node} \\ \varphi_j & j \text{ in-node or ECS-node} \end{cases}$$

Moment matrices

$$\mathbf{J}^u = \int_\Omega u\,\boldsymbol{\varphi}(\boldsymbol{x})\boldsymbol{\varphi}^\mathsf{T}(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}, \quad u = x, y, z$$

Finite element discretization – BTPDE solution

Finite element approximation of the magnetization:

$$M(\boldsymbol{x}, t) = \sum_{j=1}^{N_{\text{node}}} \varphi_j(\boldsymbol{x})\xi_j(t) = \boldsymbol{\varphi}^{\mathsf{T}}(\boldsymbol{x})\boldsymbol{\xi}(t).$$

The vector of coefficients $\boldsymbol{\xi} \in \mathbb{C}^{N_{\text{node}}}$ is the solution of

$$\mathbf{M}\frac{\partial \boldsymbol{\xi}}{\partial t} = -\left(\underline{\mathrm{i}}\gamma f(t)\mathbf{J}(\boldsymbol{g}) + \mathbf{S} + \mathbf{Q}\right)\boldsymbol{\xi}(t),$$

where $\xi_j(0) = \rho(\boldsymbol{q}_j)$ are the intial conditions and $\mathbf{J}(\boldsymbol{g}) = g_x\mathbf{J}^x + g_y\mathbf{J}^y + g_z\mathbf{J}^z$. These three matrices are only assembled once.

Finite element discretization – Matrix Formalism solution I

Find $\mathbf{L} = \mathrm{diag}(\lambda_1, \ldots, \lambda_{N_{\mathrm{eig}}})$ and $\mathbf{P} = (\boldsymbol{p}_1, \ldots, \boldsymbol{p}_{N_{\mathrm{eig}}}) \in \mathbb{R}^{N_{\mathrm{node}} \times N_{\mathrm{eig}}}$ such that

$$\mathbf{MPL} = (\mathbf{S} + \mathbf{Q})\mathbf{P}$$

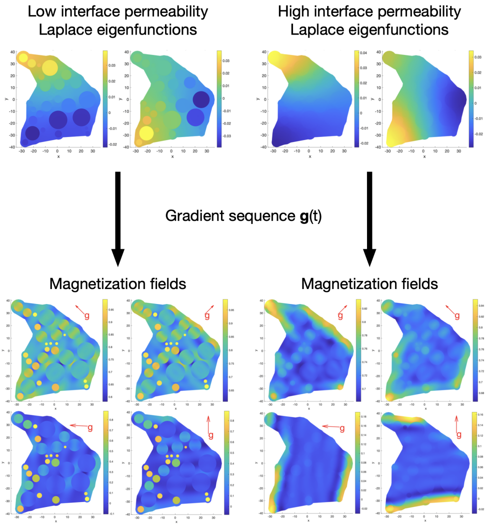
Then $\phi = \mathbf{P}^{\mathsf{T}} \varphi$. The Matrix Formalism solution has the same expression as before, by replacing the moment matrices

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

The MATLAB command `eigs` can exploit the matrix properties (sparsity, symmetry, $\mathbf{M}$ positive definite), and compute a subset N_{eig} of all the eigenvalues N_{node}.
Choice of N_{eig}: filter length scales, by setting $\ell_{\min}$:

$$N_{\mathrm{eig}} = \min\{n \mid \ell(\lambda_n) \leq \ell_{\min}\}$$

Finite element discretization – Matrix Formalism solution II



Performance improvements

- Allocation, evade copying
- Vectorization
- Parallelization (outer loops only) – solve for different gradient sequences in parallel
- Only compute quantities once (e.g. matrix assembly)
- Optimize at the right level – inner loops
- Exploit assumptions in specific configurations, without loss of generality
    - Constant quantities
    - Specific time profiles f
    - Sparsity patterns, symmetric and positive definite FE matrices
- Reusing and saving solutions
- MATLAB specific improvements

Other modifications

- New input parameter system: MATLAB scripts
- Possibility of adding T_2-relaxation terms in BTPDE and MF
- Support for arbitrary time profiles in all solvers
- Remove PDE-toolbox dependency for eigenvalue decomposition. The use of the Parallel computing toolbox is optional
- Thorough commenting
- Consistent naming conventions and code style
  - Self-explanatory names
  - Air!
  - Hierarchy and indexing
  - Code factorization and reuse – less duplicate files
- Modularity – users can change parts of the code
- Merge "modules" (NeuronModule, MatrixFormalismModule)

Model gallery

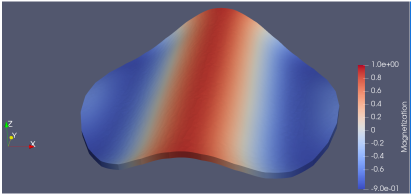
Provide a selection of models of different fidelity levels and assumptions:

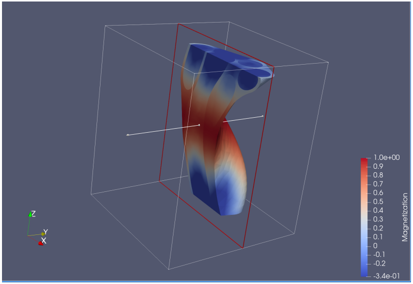
- BTPDE – High fidelity, outputs magnetization
- Matrix Formalism – Reduced order model, but well chosen functional bases of arbitrary precision. Outputs magnetization
- Signal approximation using b-values and ADC or a MF diffusion tensor (Gaussian Approximation)
- Homogenization techniques for the ADC[9]
- Short Time Approximation (STA)[10, 11, 12]
- Analytical solutions for certain geometries[13, 14]
- Monte Carlo simulations (not included in SpinDoctor)

The different solvers use the same data format.

Postprocessing

- Compute signal from magnetization
- Fit ADC from signal
- Error estimations
- Useful plots
- Solution behavior using Matrix Formalism with eigenfunction bases
- Data analysis in Paraview





Future of SpinDoctor I

- MATLAB is good for prototyping. Julia is efficient and expressive. Python can be combined with a C++ backend.
- Adapt for other finite element types, possibly using a pre-built software (Gridap[15], FEniCS[16])

```
a(u, v) = (
    − ∫(∂_t(u) ∗ v)∗dΩ
    − ∫( im∗γ∗f(t) ∗ g⋅x ∗ u ∗ v )∗dΩ
    − ∫( σ ∗ ∇(u) ⋅ ∇(v) )∗dΩ
    + ∫( κ ∗ jump(u) ∗ v )∗dΓ
)
```

- Further explore ODE solvers – take full advantage of the sparse, linear and interval-wise constant Bloch-Torrey operator. *DifferentialEquations.jl*[17] has an enormous gallery of optimized solvers.

Future of SpinDoctor II

Subject/Item	MATLAB	SciPy	deSolve	DifferentialEquations.jl	Sundials	Hairer	ODEPACK/Netlib /NAG	JiTCODE	PyDSTool	FATODE	GSL	BOOST	Mathematica	Maple
Language	MATLAB	Python	R	Julia	C++ and Fortran	Fortran	Fortran	Python	Python	Fortran	C	C++	Mathematica	Maple
Selection of Methods for ODEs	Fair	Poor	Fair	Excellent	Good	Good	Good	Poor	Poor	Good	Poor	Fair	Fair	Fair
Efficiency*	Poor	Poor	Poor	Excellent	Excellent	Good	Good	Good	Good	Good	Fair	Fair	Fair	Good
Tweakability	Fair	Poor	Good	Excellent	Excellent	Good	Good	Fair	Fair	Fair	Fair	Fair	Good	Fair
Event Handling	Good	Good	Fair	Excellent	Good**	None	Good**	None	Fair	None	None	None	Good	Good
Symbolic Calculation of Jacobians and Autodifferentiation	None	None	None	Excellent	None	None	None	None	None	None	None	None	Excellent	Excellent
Complex Numbers	Excellent	Good	Fair	Good	None	None	None	None	None	None	None	Good	Excellent	Excellent
Arbitrary Precision Numbers	None	None	None	Excellent	None	None	None	None	None	None	None	Excellent	Excellent	Excellent
Control Over Linear/Nonlinear Solvers	None	Poor	None	Excellent	Excellent	Good	Depends on the solver	None	None	None	None	None	Fair	None
Built-in Parallelism	None	None	None	Excellent	Excellent	None	None	None	None	None	None	Fair	None	None
Differential-Algebraic Equation (DAE) Solvers	Good	None	Good	Excellent	Good	Excellent	Good	None	Fair	Good	None	None	Good	Good
Implicitly-Defined DAE Solvers	Good	None	Excellent	Fair	Excellent	None	Excellent	None	None	None	None	None	Good	None
Constant-Lag Delay Differential Equation (DDE) Solvers	Fair	None	Poor	Excellent	None	Good	Fair (via DDVERK)	Fair	None	None	None	None	Good	Excellent
State-Dependent DDE Solvers	Poor	None	Poor	Excellent	None	Excellent	Good	None	None	None	None	None	None	Excellent
Stochastic Differential Equation (SDE) Solvers	Poor	None	None	Excellent	None	None	None	Good	None	None	None	None	Fair	Poor
Specialized Methods for 2nd Order ODEs and Hamiltonians (and Symplectic Integrators)	None	None	None	Excellent	None	Good	None	None	None	None	None	Fair	Good	None
Boundary Value Problem (BVP) Solvers	Good	Fair	None	Good	None	None	Good	None	None	None	None	None	Good	Fair
GPU Compatibility	None	None	None	Excellent	Good	None	None	None	None	None	None	Good	None	None
Analysis Addons (Sensitivity Analysis, Parameter Estimation, etc.)	None	None	None	Excellent	Excellent	None	Good (for some methods like DASPK)	None	Poor	Good	None	None	Excellent	None

*Efficiency takes into account not only the efficiency of the implementation, but the features of the implemented methods (advanced timestepping controls, existence of methods which are known to be more efficient, Jacobian handling)

**Event handling needs to be implemented yourself using basic rootfinding functionality

For more detailed explainations and comparisons, see the following blog post:

http://www.stochasticlifestyle.com/a-comparison-between-differential-equation-solver-suites-in-matlab-r-julia-python-c-and-fortran

Scale	None	Poor	Fair	Good	Excellent
Explanation	Functionality does not exist	Functionality exists, but is feature-incomplete	The basic features exist	The basic features exist and some extra tweakability exists. May include extra methods for efficiency.	Has all of the basic features and more. Extra features for flexibility and efficiency.

7

[7] http://www.stochasticlifestyle.com/wp-content/uploads/2019/08/de_solver_software_comparsion.pdf

References I

Jing-Rebecca Li, Van-Dang Nguyen, Try Nguyen Tran, Jan Valdman, Cong-Bang Trang, Khieu Van Nguyen, Duc Thach Son Vu, Hoang An Tran, Hoang Trong An Tran, and Thi Minh Phuong Nguyen.
SpinDoctor: A MATLAB toolbox for diffusion MRI simulation.
NeuroImage, 202:116120, 2019.

Chengran Fang, Van-Dang Nguyen, Demian Wassermann, and Jing-Rebecca Li.
Diffusion MRI simulation of realistic neurons with SpinDoctor and the Neuron Module.
NeuroImage, 222:117198, 2020.

Jing-Rebecca Li, Try Tran, and Van-Dang Nguyen.
Practical computation of the diffusion MRI signal of realistic neurons based on Laplace eigenfunctions.
NMR in Biomedicine, 33, 07 2020.

References II

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

Paul T. Callaghan and Janez Stepianik.
Frequency-domain analysis of spin motion using modulated-gradient NMR.
Journal of Magnetic Resonance, Series A, 117(1):118–122, November 1995.

Mark D. Does, Edward C. Parsons, and John C. Gore.
Oscillating gradient measurements of water diffusion in normal and globally ischemic rat brain.
Magn. Reson. Med., 49(2):206–215, 2003.

References III

Alexandre V Barzykin.
Theory of spin echo in restricted geometries under a step-wise gradient pulse sequence.
Journal of Magnetic Resonance, 139(2):342–353, 1999.

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

Simona Schiavi, Houssem Haddar, and Jing-Rebecca Li.
A macroscopic model for the diffusion mri signal accounting for time-dependent diffusivity.
SIAM Journal on Applied Mathematics, 2016.
Accepted.

References IV

Partha P Mitra, Pabitra N Sen, Lawrence M Schwartz, and Pierre Le Doussal.
Diffusion propagator as a probe of the structure of porous media.
Physical review letters, 68(24):3555–3558, 1992.

Partha P. Mitra, Pabitra N. Sen, and Lawrence M. Schwartz.
Short-time behavior of the diffusion coefficient as a geometrical probe of porous media.
Phys. Rev. B, 47:8565–8574, Apr 1993.

Simona Schiavi, Houssem Haddar, and Jing-Rebecca Li.
Correcting the short time ADC formula to account for finite pulses.
ISMRM Workshop, 09 2016.

References V

- Denis Grebenkov.
  NMR survey of reflected brownian motion.
  Reviews of Modern Physics, 79(3):1077–1137, 2007.

- Denis S. Grebenkov.
  Pulsed-gradient spin-echo monitoring of restricted diffusion in multilayered structures.
  Journal of Magnetic Resonance, 205(2):181–195, August 2010.

- Santiago Badia and Francesc Verdugo.
  Gridap: an extensible Finite Element toolbox in Julia.
  Journal of Open Source Software, 5(52):2520, 2020.

- Martin Alnæs, Jan Blechta, Johan Hake, August Johansson, Benjamin Kehlet, Anders Logg, Chris Richardson, Johannes Ring, Marie E Rognes, and Garth N Wells.
  The FEniCS project version 1.5.
  Archive of Numerical Software, Vol 3, 2015.

References VI

Christopher Rackauckas and Qing Nie.
DifferentialEquations.jl – a performant and feature-rich ecosystem for solving differential equations in Julia.
Journal of Open Research Software, 5(1), 2017.