Modeling Neurodegenerative Medical Diseases Using the Fisher-Kolmogorov Equation

Mario Capodanno, Valerio Grillo, Emanuele Lovino
M.Sc. High-Performance Computing Engineering
Politecnico di Milano, Milan, Italy

June 7, 2025

Abstract

This project aims to report the details of the mathematical and numerical model of the Fisher-Kolmogorov equation, which plays a crucial role in modeling protein spreading in neurodegenerative diseases such as Alzheimer's and Parkinson's. The project is focused on developing a numerical solver for this equation by employing finite elements for spatial discretization and the backward Euler method for time discretization. In the first sections, we address the problem by analyzing both the semi-discrete and fully discrete Galerkin formulations. The following sections are dedicated to the software implementation and numerical tests, first on a simpler one-dimensional case and then on a three-dimensional mesh representing a simplified version of half of the human brain. The 3D case is analyzed in depth, including stability and error analysis, yielding qualitative and quantitative results consistent with the reference sources.

1 Introduction

As aging trends have been continuously increasing in recent years, neurodegeneration is set to become an inevitable major challenge in medicine and public health. A critical aspect of these diseases is the absence of symptoms in the early stages, which typically become noticeable only one or two decades after the onset of brain abnormalities. Although modeling neurodegeneration remains an open problem, recent studies suggest that the misfolding and aggregation of prion-like proteins into toxic and insoluble conformations play a fundamental role in developing neurodegenerative diseases. In particular, one of the most common proteins undergoing this process is α-synuclein, which is believed to be closely linked to Parkinson's disease. A simplified description of the macroscopic spreading of misfolded proteins can be obtained by modeling it through the Fisher-Kolmogorov equation, a nonlinear diffusion-reaction partial differential equation.

2 The Mathematical Model

To model the complex phenomena of the growth and spreading of the concentration of misfolded protein in the human brain, we consider the Fisher-Kolmogorov (FK) equation, a non-linear diffusion-reaction equation. The problem is time-dependent (with time $t \in (0, T]$) while space $\mathbf{x} \in \Omega \subset \mathbb{R}^d \quad (d = 2, 3)$.

$$\begin{cases} \dfrac{\partial c}{\partial t} = \operatorname{div}(\mathbf{D}\,\nabla c) + \alpha\, c\,(1 - c) + f & \text{in } \Omega \times (0, T],\ \Omega \subset \mathbb{R}^d\ (d = 2, 3), \\ (\mathbf{D}\,\nabla c) \cdot \mathbf{n} = 0 & \text{on } \partial\Omega \times (0, T], \\ c(\mathbf{x}, 0) = c_0(\mathbf{x}) & \text{in } \Omega. \end{cases} \tag{1}$$

In our case, the solution $c = c(\mathbf{x}, t)$ represents the concentration of misfolded proteins, which is constrained to vary between 0 and 1 due to a re-parameterization of the equation, with 0 representing the absence of misfolded proteins and 1 representing their high prevalence. The parameter α characterizes the growth of spreading, while the diffusion tensor $\mathbf{D} = \mathbf{D}(\mathbf{x})$ characterizes the spatial dispersion of misfolded proteins throughout the domain Ω.
Specifically for prion propagation modelling, $\mathbf{D}$ often combines isotropic extracellular diffusion (magnitude $d_{\text{ext}} \geq 0$) and anisotropic diffusion along axonal pathways (magnitude $d_{\text{axn}} \geq 0$):

$$\mathbf{D} = d_{\text{ext}}\mathbf{I} + d_{\text{axn}}(\mathbf{n} \otimes \mathbf{n}), \tag{2}$$

where the unit vector field $\mathbf{n}(\mathbf{x})$ indicates the local axonal fiber direction.
Equation (1) employs a homogeneous Neumann condition $(\mathbf{D}\,\nabla c) \cdot \mathbf{n} = 0$ on the entire boundary $\partial\Omega$ for $t \in (0, T]$ and no Dirichlet condition is applied.

Assumption: For the analysis of equation (1), we assume the coefficients and forcing terms possess the following levels of regularity:

- $\alpha \in L^\infty(\Omega)$.
- $\mathbf{D} \in L^\infty(\Omega, \mathbb{R}^{d\times d})$ and $\exists d_0 > 0\ \forall \xi \in \mathbb{R}^d : d_0|\xi|^2 \leq \xi^T \mathbf{D} \xi\ \forall \xi \in \mathbb{R}^d$.
- $f \in L^2((0, T]; L^2(\Omega))$.
- $c_0 \in L^2(\Omega)$.

According to the previous *Assumption*, and specifically when $f = 0$ and $\phi_N = 0$, the equation (1) admits traveling wave solutions, provided the initial condition satisfies $0 \leq c_0(\mathbf{x}) \leq 1$ for all $\mathbf{x} \in \Omega$. Furthermore, these conditions guarantee that the solution remains bounded within the same interval, $0 \leq c(\mathbf{x}, t) \leq 1$, for all $\mathbf{x} \in \Omega$ and $t > 0$. This particular setup possesses two spatially uniform steady-state solutions: an unstable equilibrium at $c = 0$ and a stable equilibrium at $c = 1$.

3 Weak Formulation

We begin by deriving the weak formulation to the problem. Let $v \in V = H^1(\Omega) = \{v \in L^2(\Omega)\,,\, D^1 v \in L^2(\Omega)\}$. At all times $t \in (0,T]$, we look for $c(t) \in V$ and $c(0) = c_0$.

$$\int_\Omega \frac{\partial c}{\partial t} v \, d\Omega = \int_\Omega \mathrm{div}(\mathbf{D}\nabla c) v \, d\Omega + \int_\Omega \alpha c(1-c) v \, d\Omega + \underbrace{\int_\Omega f v \, d\Omega}_{\text{assumption on } \Omega} \to 0$$

By applying Green's formula for the divergence operator

$$\int_\Omega \mathrm{div}(\mathbf{D}\nabla c) v \, d\Omega = -\int_\Omega \mathbf{D}\nabla c \cdot \nabla v \, d\Omega + \underbrace{\int_{\partial\Omega} (\mathbf{D}\nabla c) \cdot \mathbf{n} v \, d\gamma}_{\text{Neumann condition}} \to 0$$

Then we obtain

$$\int_\Omega \frac{\partial c}{\partial t} v \, d\Omega = -\int_\Omega \mathbf{D}\nabla c \cdot \nabla v \, d\Omega + \int_\Omega \alpha c(1-c) v \, d\Omega$$

Up to now, the problem reads:

for each $t \in (0,T)$, find $c \in V$ such that

$$\begin{cases} \int_\Omega \frac{\partial c}{\partial t} v \, d\Omega = \underbrace{-\int_\Omega \mathbf{D}\nabla c \cdot \nabla v \, d\Omega + \int_\Omega \alpha c(1-c) v \, d\Omega}_{\text{-b(c)(v)}} & \forall v \in V \\ c(\mathbf{x}, 0) = c_0(\mathbf{x}) & \text{in } \Omega \end{cases} \tag{3}$$

We put the problem in its residual form

$$R(c)(v) = \int_\Omega \frac{\partial c}{\partial t} v \, d\Omega + b(c)(v)$$

the weak formulation reads:

$\forall t \in (0,T)$, find $c \in V$ such that:

$$\begin{cases} R(c)(v) = 0 & \forall v \in V \\ c(\mathbf{x}, 0) = c_0(\mathbf{x}) & \text{in } \Omega \end{cases} \tag{4}$$

4 Numerical Modeling

4.1 Semidiscrete Galerkin Formulation

We now introduce a mesh $\mathcal{T}_h$ over the domain Ω and the finite element space

$$X_h^r := \{v_h \in [\mathcal{C}^0(\Omega)]^d, \forall k \in \mathcal{T}_h, v_h|_k \in \mathbb{P}^r, r = 1\}$$

and set $V_h = V \cap X_h^r$, $\dim(V_h) = N_h < +\infty$, and we introduce a basis over V_h. The semi-discrete formulation reads:

$\forall t \in (0,T)$, find $c_h(t) \in V_h$ such that:

$$\begin{cases} R(c_h)(v_h) = 0 & \forall v_h \in V_h \\ c_h(\mathbf{x},0) = c_{0,h}(\mathbf{x}) & \text{in } \Omega \end{cases} \tag{5}$$

Where $c_{0,h}(\mathbf{x})$ is a suitable approximation of $c_0(\mathbf{x})$.

4.2 Fully Discrete Galerkin Formulation

We now discretize through time by using the Implicit Euler method. Let us partition the interval $[0,T]$ into the subintervals $(t_n, t_{n+1}]$ with $n = 0, 1, \ldots, N_{T-1}, t_{N_T} = T$. Each sub-interval has length Δt.
We denote with a superscript n the approximate solution at time t_n, i.e. $c_h^n \approx c_h(t_n)$

The fully discrete formulation of the problem can be expressed as:

$$\underbrace{\int_\Omega \frac{c_h^{n+1} - c_h^n}{\Delta t} v_h \, d\Omega + b(c_h^{n+1})(v_h)}_{R^{n+1}(c_h^{n+1})(v_h)} = 0 \quad \forall v_h \in V_h, \quad n = 0, 1, \ldots, N_{T-1} \tag{6}$$

This is a nonlinear problem, due to b being nonlinear in c_h^{n+1}. Therefore, we employ Newton's method for its solution. To this end, we introduce a generalized concept of derivative that also works for functionals and bilinear forms, called Fréchet derivative. For a given functional $G : V \to \mathbb{R}$ the Fréchet derivative at a point $u \in V$ is a linear operator $A(u) = \frac{dG}{du} : V \to \mathbb{R}$ such that

$$\lim_{\|\delta\| \to 0} \frac{|G(u+\delta) - G(u) - A(u)(\delta)|}{\|\delta\|} = 0.$$

we compute the derivative $a(c_h^{n+1})(\delta_h, v_h)$ of the discrete residual $R^{n+1}(c_h^{n+1})(v_h)$:

$$\begin{aligned} R(c_h^{n+1} + \delta_h)(v_h) &= \int_\Omega \frac{c_h^{n+1} + \delta_h - c_h^n}{\Delta t} v_h \, d\Omega + b(c_h^{n+1} + \delta_h)(v_h) = \\ &= \int_\Omega \frac{c_h^{n+1} - c_h^n}{\Delta t} v_h \, d\Omega + \int_\Omega \frac{\delta_h}{\Delta t} v_h \, d\Omega + b(c_h^{n+1} + \delta_h)(v_h) \end{aligned}$$

we now expand the bilinear form

$$b(c_h^{n+1} + \delta_h)(v_h) = \int_\Omega \mathbf{D}\nabla(c_h^{n+1} + \delta_h) \cdot \nabla v_h \, d\Omega - \int_\Omega \alpha(c_h^{n+1} + \delta_h)(1 - c_h^{n+1} - \delta_h) v_h \, d\Omega =$$

$$= \int_\Omega \mathbf{D}\nabla c_h^{n+1} \cdot \nabla v_h \, d\Omega + \int_\Omega \mathbf{D}\nabla \delta_h \cdot \nabla v_h \, d\Omega - \int_\Omega \alpha c_h^{n+1}(1 - c_h^{n+1}) v_h + \alpha \delta_h (1 - 2c_h^{n+1}) v_h - \alpha \delta_h^2 v_h d\Omega$$

This leads to

$$R(c_h^{n+1}+\delta_h)(v_h) - R(c_h^{n+1})(v_h) = \underbrace{\int_\Omega \frac{\delta_h}{\Delta t} v_h \, d\Omega + \int_\Omega \mathbf{D}\nabla\delta_h \cdot \nabla v_h \, d\Omega - \int_\Omega \alpha\delta_h(1-2c_h^{n+1})v_h \, d\Omega}_{a(c_h^{n+1})(\delta_h, v_h)} +$$
$$+ o(\|\delta_h^2\|)$$

Notice that the bilinear form a is linear with respect to both δ_h and v_h. Therefore to solve the problem, we proceed as follows: given an initial solution u_h^0 we iterate for $n = 0, 1, \ldots, N_{T-1}$ and compute u_h^{n+1} by solving (5) using Newton's method.
At any fixed time n, Newton's method reads:
Given the initial guess $c_h^{n+1,(0)} = c_h^n$, iterate for $k = 0, 1, 2, \ldots$ until convergence:

1. assemble and solve the linear system:

$$a(c_h^{n+1,(k)})(\delta_h^{(k)}, v_h) = -R(c_h^{n+1,(k)})(v_h) \quad \forall v_h \in V_h.$$

2. Set $c_h^{n+1,(k+1)} = c_h^{n+1,(k)} + \delta_h^{(k)}$.

We need to perform two nested loops: the outer loop iterates through time (over the index n), while the inner loop applies Newton's method (iterating over index k). Within the inner loop we repeatedly assemble the system. As Newton's method is not guaranteed to converge, if the residual doesn't fall below a given tolerance in a maximum number of iterations, we assume that Newton is not converging and stop the execution.

5 Error and stability estimation

5.1 Stability of semi-discrete formulation

In this context, boundedness does not necessarily imply stability due to the non-linear nature of the problem. Although we do not provide a formal proof of stability here, we assume it as a necessary condition for convergence.

5.2 Stability of fully-discrete formulation

Since Backward-Euler method was used for the discretization with respect to time of the problem, we have unconditionally stability $\forall \Delta t \geq 0$.

5.3 Convergence

General estimate.

$$\left\| c(t^{(k)}) - c_h(t^{(k)}) \right\|_{L^2(\Omega)}^2 \;\leq\; C\big(h^r + \Delta t^{\,p(\vartheta)}\big), \tag{7}$$

where C is a constant, independent of the discretization parameters h and Δt; it typically depends only on the problem data (e.g. coefficients, domain geometry, final time T) and on the regularity of the exact solution c. Here $k = 0, 1, \ldots, N$ is the discrete time–step index defined by

$$t^{(k)} = t_0 + k\,\Delta t,$$

where $\Delta t = T/N$ denotes the uniform time–step size on the interval $(0, T)$ and $t_0 = 0$.

Estimate for linear elements and backward Euler. For linear finite elements ($r = 1$) and the backward-Euler time discretization ($p(\vartheta) = 1$),

$$\left\| c(t^{(k)}) - c_h(t^{(k)}) \right\|^2_{L^2(\Omega)} \;\leq\; C\big(h + \Delta t\big), \tag{8}$$

Hence the method is first–order accurate in both space and time.

6 Software Implementation

A finite element solver for the efficient numerical solution of nonlinear partial differential equations was implemented in C++ using the deal.II library. This solver was applied to three distinct problems: a one-dimensional simulation of prion-like spreading, a three-dimensional model for spreading within the brain, and a three-dimensional convergence study on a cubic domain with a known analytical solution to verify the numerical approach. The workflow is divided into four primary stages (i.e. files `3D/src/FisherKolmogorov3D.cpp` and `3D/src/FisherKolmogorov3D.hpp`): problem setup, system assembly, numerical solution, and results output. The steps can be mapped to the main methods that the class exposes:

- `setup()`: performs all initialization tasks such as creation and partitioning of the mesh among the available MPI processes, initialization of the finite element space element type and corresponding quadrature rule, the distribution of DoFs across the cells, and the initialization of the algebraic structures for the linear system.

- `assemble_system()`: responsible for constructing finite element matrices and vectors. Iterates over the active cells and, for each quadrature point, evaluates the current and previous solutions along with their gradients and the diffusion and reaction term to compute the local contributions to assemble the mass matrix and the residual vector.

- `solve_linear_system()`: instantiates a `SolverControl` object configured with a maximum number of iterations (`max_cg_iterations`) and a tolerance scaled by the current residual norm (`cg_tolerance_factor * residual_vector.l2_norm()`), this ensures that the linear solver stops when an acceptable convergence level relative to the current nonlinear residual is reached. Using the Trilinos wrappers the Conjugate Gradient solvers is chosen, given the symmetry of the mass matrix, with SSOR precondition to accelerate convergence and preserve the symmetry of the system.

- `solve_newton()`: applies the Newton iterative method until the residual norm falls below a tolerance or a maximum number of iterations is reached. In each iteration, the method `assemble_system()` is used to rebuild the Jacobian matrix and residual vector using the current solution, and then, if the residual is higher than the chosen tolerance, the assembled linear system is solved using the `solve_linear_system()` method.

- `output`: responsible for exporting the solution by using the `DataOut` object to attach the computed solution vector and the mesh partitioning data. Then, the function writes the results in VTU format and generates a corresponding PVTU record. This setup is needed for parallel visualization tools as it aggregates data from all MPI processes (i.e. showing the partitioning on Paraview).

The main time loop is in the `solve()` method, which calls `solve_newton()` and `output()` repeatedly, increasing the simulation time (`time`) from 0 to the final time (`T`) in steps of `deltat`. Initial conditions are set using `VectorTools::interpolate` with the function `u_0`. Parameters like the diffusion coefficient (`alpha`), time step (`deltat`), and solver tolerances are handled by the `ParameterReader` class, reading settings from the `parameters.prm` file.

The `DiffusionTensor.hpp` provides a hierarchy of tensor classes to model anisotropic diffusion in the simulation. The base class `DiffusionTensor<DIM>` inherits from deal.II `TensorFunction<2,DIM>` and defines an interface with two diffusion coefficients: `dext` (external diffusion) and `daxn` (axonal diffusion). The `value()` method constructs a rank-2 tensor by combining isotropic diffusion (`dext`) with directional diffusion (`daxn`) along fiber directions determined by the abstract `computeFiber()` method. Derived classes implement different fiber orientation patterns:

1. `IsotropicDiffusionTensor<DIM>`: uniform diffusion with `daxn=0`.
2. `RadialDiffusionTensor<DIM>`: orients fibers radially from a center point.
3. `CircumferentialDiffusionTensor<DIM>` creates circular fiber patterns.

7 Numerical Results

7.1 Case 1: Prion-like spreading in 1D

To visualize the spatio-temporal dynamics that emerge from the Fisher–Kolmogorov reaction–diffusion model,

$$\frac{\partial c}{\partial t} = d\,\frac{\partial^2 c}{\partial x^2} + \alpha\, c\,(1-c), \tag{9}$$

we solved (9) in the one–dimensional domain $\Omega = \{x \mid -1 \le x \le 1\}$ over the time interval $\mathcal{T} = \{t \mid 0 \le t \le 20\}$.

Spatial and temporal discretizations. The domain was partitioned into $N = 200$ linear elements, while the interval $\mathcal{T}$ was divided into $n_{\text{step}} = 200$ uniform steps of size $\Delta t = 0.1$.

Initial and boundary data. The concentration field was initialised as $c(x, 0) = 0$ throughout Ω, except at the centre node $x = 0$ where a small seed $c(0, 0) = 0.1$ was imposed using a Gaussian function. Homogeneous Neumann conditions, $\partial c/\partial x = 0$ on $\partial\Omega$, eliminate boundary fluxes.

Parametric variation. We explored the sensitivity of the solution to

- *growth*: $\{1\alpha,\, 2\alpha,\, 4\alpha\}$ with $\alpha = 1$;
- *spreading* (diffusivity): $\{1d,\, 2d,\, 4d\}$ with $d = 10^{-4}$.

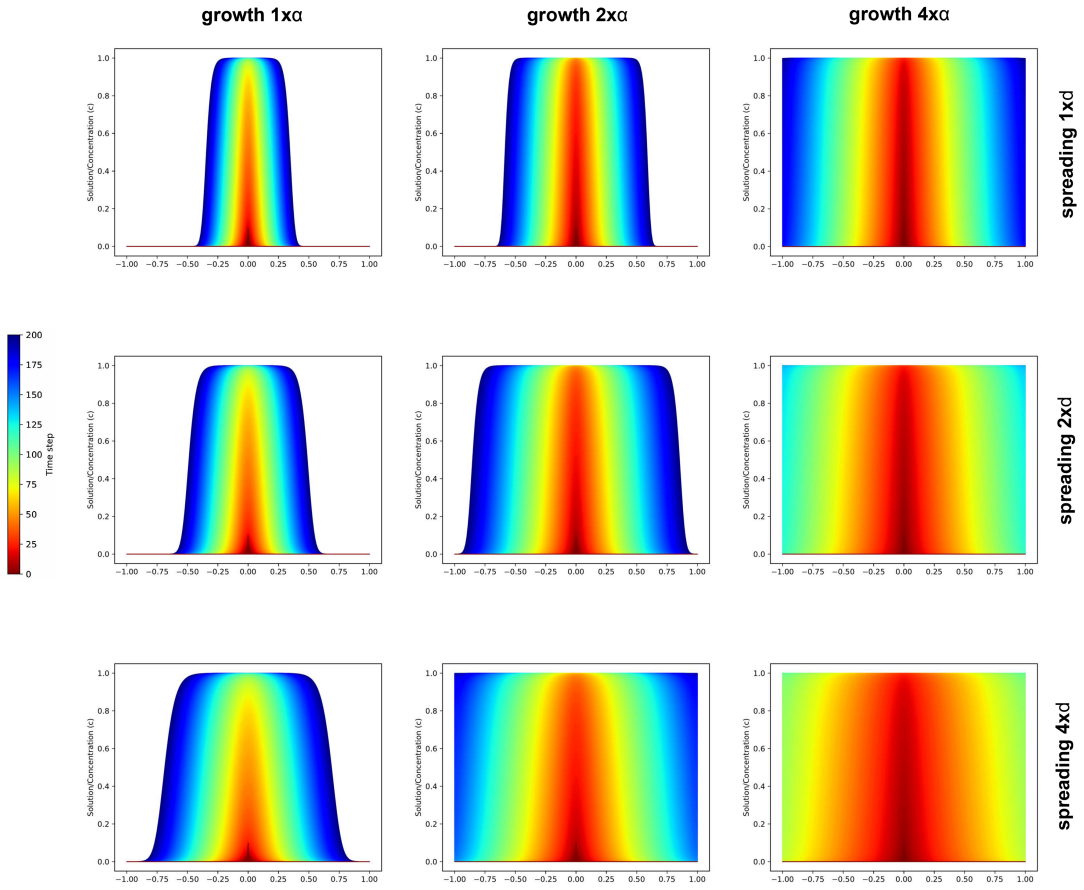

Figure 1: Rows vary diffusion $\{1d, 2d, 4d\}$; columns vary growth $\{1\alpha, 2\alpha, 4\alpha\}$. Heatmaps show concentration $c(x,t)$ from $t = 0$ (red) to $t = 20$ (blue) after seeding $c_0 = 0.1$ at $x = 0$. Higher values of growth raise the local peak, while higher values of diffusion broaden and speeds the advancing front.

7.2 Case 2: Convergence Analysis in 3D

To assess the spatial accuracy of our formulation in three space-dimensions we use a unitary cubic domain $\Omega = (0,1)^3$.

Spatial and temporal discretizations. The cube is partitioned with a sequence of *tetrahedral* elements generated with GMSH. The representative element sizes are $h = \{0.5, 0.25, 0.125, 0.0625, 0.03125\}$, so that each successive mesh is obtained by uniform refinement. Unless otherwise stated, we use continuous, piecewise-polynomial finite-element spaces of degree $r = 1$. Time is advanced by a backward Euler step of size $\Delta t = 0.01$, resulting in $n_{\text{step}} = 100$ uniform steps ($T = 1.0$).

Manufactured solution and parameters. Following Corti *et al.* [2], we prescribe the exact solution

$$c_{\text{ex}}(x,y,z,t) = \big(\cos(\pi x)\cos(\pi y)\cos(\pi z)\big)\,\mathrm{e}^{-t}, \tag{10}$$

and choose an *isotropic* diffusion tensor $\mathbf{D} = d_{\text{ext}}\mathbf{I}$ with $d_{\text{ext}} = 1$ and reaction coefficient $\alpha = 0.1$. The forcing term is obtained by substitution of (10) into the equation (1).

FE convergence and results. At the final time $T = 1.0$ we evaluate the discrete errors $e_{L^2} = \|c_{\text{ex}} - c_h\|_{L^2(\Omega)}$ and $e_{H^1} = \|c_{\text{ex}} - c_h\|_{H^1(\Omega)}$. Because $\Delta t \ll h$ in all the tests, the spatial term is expected to dominate.

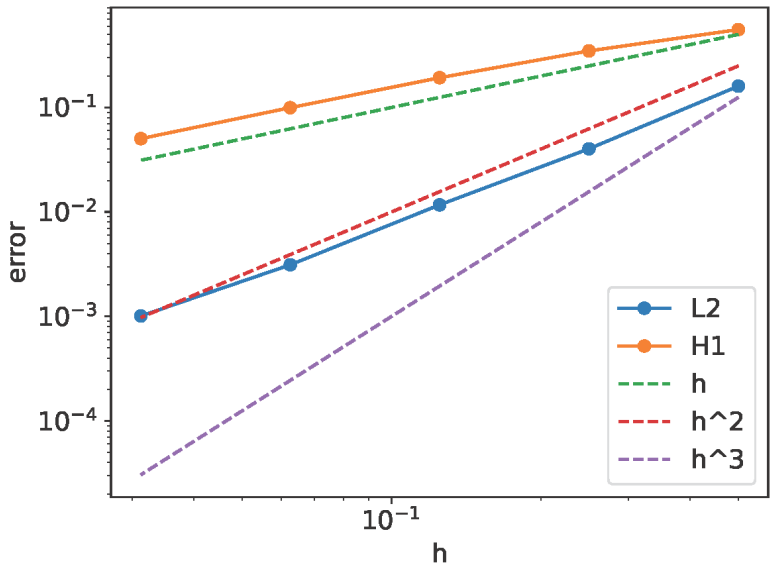

Figure 2: Convergence test on the manufactured 3-D solution (10). Solid lines with markers show the computed L^2–error (blue) and H^1–error (orange) at $T = 10^{-3}$; dashed lines indicate the reference slopes h (green), h^2 (red) and h^3 (violet). The L^2–error follows the h^2 slope whereas the H^1–error follows the h slope, in accordance with the theory.

Figure 2 plots the e_{L^2} and e_{H^1} errors versus h on log–log axes together with the reference slopes h, h^2 and h^3. For the linear space ($r = 1$) we observe:

- an $\mathcal{O}(h^2)$ decay of the L^2–error, in perfect agreement with the optimal rate $r = 2$ predicted by the theory;
- an $\mathcal{O}(h)$ decay of the H^1–error, again matching the theoretical rate.

7.3 Case 3: Half-Brain Mesh in 3D

To simulate the propagation of misfolded proteins in a more realistic setting, we solve the Fisher-Kolmogorov equation on a three-dimensional tetrahedral mesh representing a human brain hemisphere. This case considers four different diffusion implementations: *isotropic, radial, circumferential* diffusion.

Spatial and Temporal Discretization. Starting from equation (1), we employ continuous, piecewise-linear finite elements ($r = 1$). For temporal integration, we use the backward Euler method, which is consistent with the fully discrete formulation discussed earlier.

Initial and Boundary Conditions. The initial protein concentration $c(\mathbf{x}, 0)$ is designed to represent a localized pathological onset. A value of $c = 0.9$ is prescribed

within a small cubic region defined by $55 < x < 65$, $75 < y < 85$, and $35 < z < 45$ (`brain-h3.0.msh` mesh). Elsewhere, the concentration is initialized as $c = 0$. Homogeneous Neumann boundary conditions, $(D\nabla c) \cdot \mathbf{n} = 0$, are imposed on the external boundary $\partial\Omega$, modeling a no-flux condition.

Results. Here we report the results of *isotropic* diffusion. Given its nature, we expect the misfolded proteins to spread uniformly from the initial seed region.

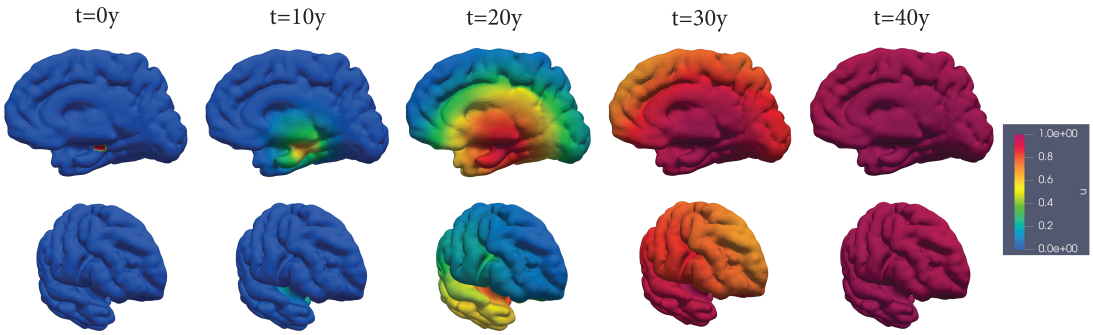

Figure 3: Columns show snapshots at $t = 0$, 10, 20, 30, 40 y; the upper row illustrates the medial view and the lower row the lateral view of the right hemisphere. The colour map (blue $\rightarrow$ red) encodes the concentration $c \in [0, 1]$. Starting from the initial seed at $t = 0$, the concentration diffuses uniformly across the cortex and, by $t = 40$ y, saturates the entire domain.

Spatio-temporal evolution of the scalar field $c(\mathbf{x}, t)$ human brain hemisphere surface. Parameter set $d_{\text{ext}} = 4.0$, $d_{\text{axn}} = 40.0$, $\alpha = 0.3$, $\Delta t = 2$ y, $T = 40$ y.

To verify how the geometry of the diffusion tensor $\mathbf{D}(\mathbf{x})$ shapes the propagation, we repeated the simulation with three prototypical tensors while keeping every kinetic parameter fixed. In addition to the baseline case *isotropic*, we also considered the tensors *circumferential* and *radial*. Figure 4 contrasts the resulting concentration fields at the intermediate time $t = 16$ y, before global saturation occurs but well after the initial transients have decayed, showing the differences each transport tensor produces.

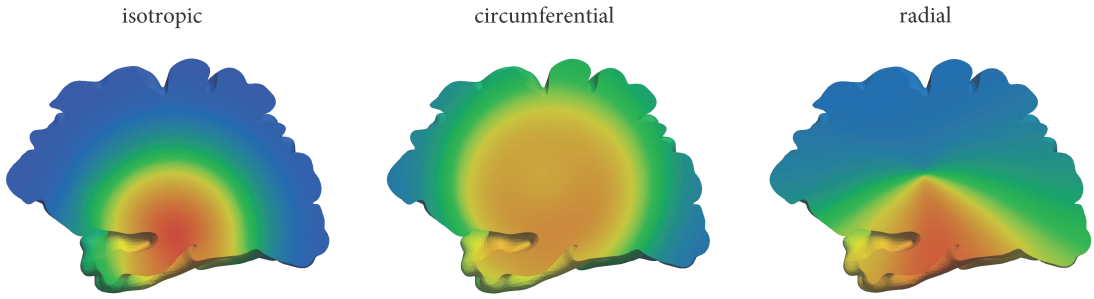

Figure 4: At $t = 16$ y, isotropic, circumferential, radial transport tensor produces distinct concentration patterns.

8 Execution Time Analysis

In this section, we present timing results obtained from local (single-node) simulations using different numbers of MPI processes. The goal is to understand how each phase of the computation (assembling the system, setup, solving the linear system, and writing output) behaves as we increase the parallelism. First, we show the absolute wall-times for each phase; then, we examine the relative percentage breakdown of those phases.

All local tests were performed on a workstation equipped with an AMD EPYC 7742 64-core processor and 1TB of RAM. Since the EPYC 7742 provides 64 hardware threads (64 cores), 64 was the maximum number of physical threads used. Each run was invoked using the `mpirun --use-hwthread-cpus` option to exploit all available threads.

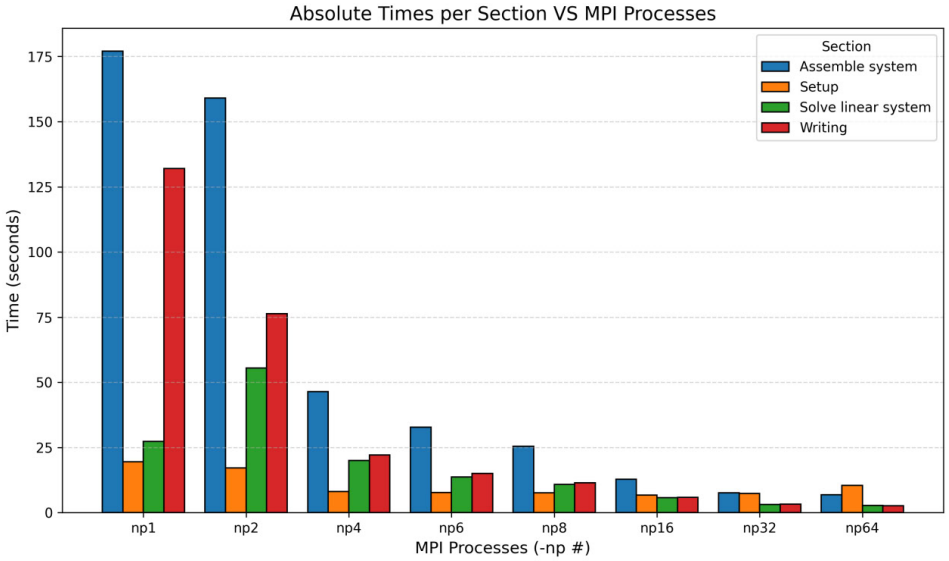

Figure 5: Absolute times (in seconds) per section versus the number of MPI processes for local simulations. Each group of four bars corresponds to a different `np#` run.

Comment on Figure 5. As the number of MPI processes increases from np1 to np64, the time spent in the "Assemble system" phase (blue bars) drops significantly, from roughly 178s at np1 down to about 8s at np64, demonstrating good parallel speedup. The "Setup" time (orange bars) actually grows slightly with more processes, rising from np4 to np64, likely due to overhead in distributing data or initializing communication. The "Solve linear system" phase (green bars) shows increasing time from np1 to np2, but then appears to decrease until it reaches around 3-4s for np32 and np64. Finally, "Writing" (red bars) decreases from about 134s to 3s, reflecting that I/O is splitting across ranks. In sum, assembly and writing benefit most from additional MPI parallelization.

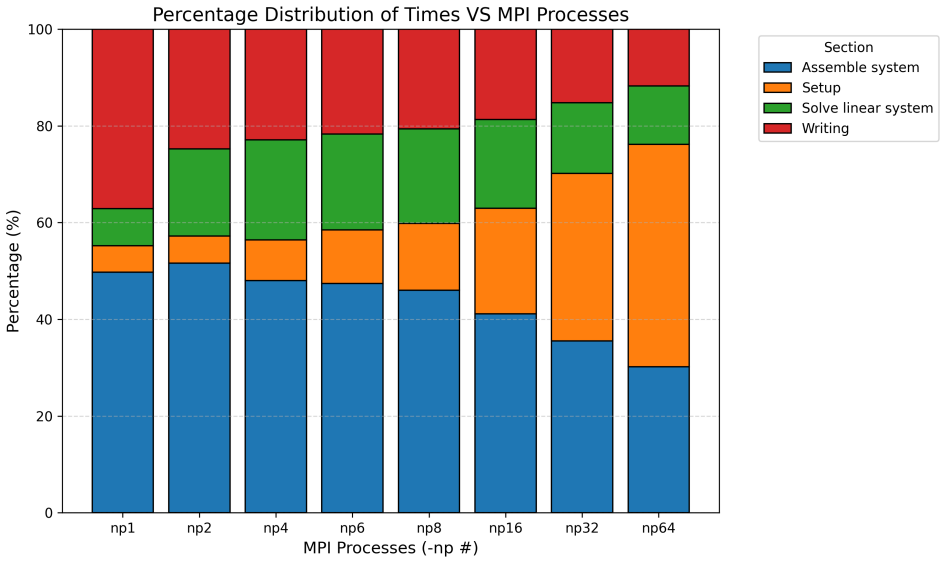

Figure 6: Percentage distribution of times per section versus the number of MPI processes for local simulations.

Comment on Figure 6. When looking at each run's total time in percentages, the "Assemble system" phase (blue) still takes the most time but gradually decreases, from about 50% at np1 to around 30% at np64. The "Setup" portion (orange) grows from around 5% to nearly 45% as we add more processes, indicating that startup overhead becomes more significant in the overall workflow. The "Solve linear system" phase (green) climbs from about 8% at np1 to roughly 15% at np64, showing that solver operations become more relevant as assembly costs decrease. Lastly, "Writing" (red) drops from approximately 38% down to around 10%, meaning that I/O takes a smaller fraction of runtime as it parallelizes. Together, these plots confirm that while assembly remains a significant cost on a single node, its relative weight decreases with additional MPI ranks, and setup overhead becomes increasingly dominant at higher process counts.

9 Conclusion

In this work we have proposed a numerical solver for the solution of the Fisher-Kolmogorov equation applied to the spreading of α-synuclein proteins in neurodegenerative diseases such as Alzheimer and Parkinson, deriving the semidiscrete and fullydiscrete formulation and employing the Newton method to solve the non-linear problem.
We firstly applied the model in a one-dimensional case following the main source of this work. The numerical convergence tests were presented on the three-dimensional case (cubic domain). As shown above, the convergence tests confirmed the theoretical results expected.
Finally, we presented a simulation of the spreading of α-synuclein proteins in a three-dimensional brain hemisphere mesh which provided a more realistic enviroment for our simulations.

References

[1] J. Weickenmeier, M. Jucker, A. Goriely, and E. Kuhl, "A physics-based model explains the prion-like features of neurodegeneration in Alzheimer's disease, Parkinson's disease, and amyotrophic lateral sclerosis," *Journal of the Mechanics and Physics of Solids* 124 (2019) 264–281. https://doi.org/10.1016/j.jmps.2018.10.013

[2] M. Corti, F. Bonizzoni, L. Dede', A. M. Quarteroni, and P. F. Antonietti, "Discontinuous Galerkin methods for the Fisher–Kolmogorov equation with application to α-synuclein spreading in Parkinson's disease," *Computer Methods in Applied Mechanics and Engineering* 417 (2023) 116450. https://doi.org/10.1016/j.cma.2023.116450

[3] Salsa, S. (2016). Introduzione. In: Equazioni a derivate parziali. UNITEXT(), vol 98. Springer, Milano. https://doi.org/10.1007/978-88-470-5785-2_1

[4] deal.II Developers, *The deal.II Library: Reference Documentation*, version 9.6.0, 2024. https://www.dealii.org/