Final Project - Monodomain Equation - Scientific Learning

Michele Cattaneo, Nicolai Hermann, Oliver Tryding

June 2024

Refer to our GitHub repo for all the code and relevant outputs.

1 Finite Element Method (FEM)

The monodomain PDE is given as the following:

$$\begin{aligned} \frac{\partial u}{\partial t} &= \nabla \cdot \Sigma_h \nabla u + \nabla \cdot \Sigma_d \nabla u - a\,(u - f_r)\,(u - f_t)\,(u - f_d) && \text{in } \Omega \times I, \\ u &= u_0 && \text{at } t = 0, \end{aligned} \tag{1}$$

with the initial conditions:

$$u_0 = \begin{cases} 1, & \text{if } x \geq 0.9 \text{ and } y \geq 0.9, \\ 0, & \text{otherwise.} \end{cases}$$

The values for Σ_h, Σ_d, a, f_r, f_t and f_d are given in the project description.

1.1 The Implicit-Explicit Method (IMEX) time integration scheme for a time-step dt

Starting from the Equation 1:

$$\frac{\partial u}{\partial t} = \nabla \cdot \Sigma_h \nabla u + \nabla \cdot \Sigma_d \nabla u - a\,(u - f_r)\,(u - f_t)\,(u - f_d)\,.$$

We can derive the IMEX scheme representation for each time-step t^n by separating the diffusion part and the reaction part. We then discretize the time derivative and the diffusion term implicitly. The reaction term on the other hand is treated explicitly. For brevity, we also will only display a single conductivity factor Σ instead of the previous two $\Sigma_h + \Sigma_d$.

$$\begin{aligned} \frac{u^{n+1} - u^n}{\Delta t} &= \nabla \cdot \Sigma \nabla u^{n+1} - a\,(u^n - f_r)\,(u^n - f_t)\,(u^n - f_d) \\ &\iff \\ \frac{u^{n+1}}{\Delta t} - \nabla \cdot \Sigma \nabla u^{n+1} &= \frac{u^n}{\Delta t} - a\,(u^n - f_r)\,(u^n - f_t)\,(u^n - f_d)\,. \end{aligned}$$

1.2 The weak formulation for the problem solved at each time step t^n

The weak formulation of the problem is to find $u \in H^1(\Omega)$ such that for all test functions $v \in H^1(\Omega)$:

$$\begin{aligned} &\int_\Omega \frac{u-u}{\Delta t} v \, d\overline{x} - \\ &\int_\Omega \nabla \cdot \Sigma \nabla u \cdot v \, d\overline{x} + \\ &\int_\Omega a(u-f_r)(u-f_t)(u-f_d) v \, d\overline{x} = 0. \end{aligned} \tag{2}$$

We can rewrite equation 2 to be solved for each time-step t^n with the IMEX scheme in the same way as in 1.1.

$$\int_\Omega \frac{u^{n+1}}{\Delta t} v d\overline{x} - \int_\Omega \nabla \cdot \Sigma \nabla u^{n+1} \cdot v d\overline{x} = \int_\Omega \frac{u^n}{\Delta t} v d\overline{x} - \int_\Omega a(u^n - f_r)(u^n - f_t)(u^n - f_d) v d\overline{x}. \tag{3}$$

We can now apply the divergence theorem to the diffusion terms in Equation 3:

$$\int_\Omega \nabla \cdot \Sigma \nabla u^{n+1} \cdot v d\overline{x} = -\int_\Omega \Sigma \nabla u^{n+1} \nabla \cdot v d\overline{x} + \int_{\partial\Omega} (\Sigma \nabla u^{n+1} \cdot n) v ds, \tag{4}$$

where the homogeneous boundary conditions implies $\Sigma \nabla u^{n+1} \cdot n = 0$ and thus the second term on the left-hand side disappears. Inserting Equation 4 into Equation 3 gives:

$$\int_\Omega \frac{u^{n+1}}{\Delta t} v d\overline{x} + \int_\Omega \Sigma \nabla u^{n+1} \nabla \cdot v d\overline{x} = \int_\Omega \frac{u^n}{\Delta t} v d\overline{x} - \int_\Omega a(u^n - f_r)(u^n - f_t)(u^n - f_d) v d\overline{x}. \tag{5}$$

1.3 Algebraic representation associated with a finite element formulation

First, we define the basis functions ϕ_i over the elements. The approximate solution u_h at any point in Ω can be expressed as a linear combination of these basis functions:

$$u_h = \sum_j U_j \phi_j,$$

where U_j are the nodal values of the approximate solution. We discretize the weak form in equation 5, using the basis functions ϕ_i:

$$\begin{aligned} &\sum_j U_j^{n+1} \int_\Omega \frac{\phi_j}{\Delta t} \phi_i d\overline{x} + \sum_j U_j^{n+1} \int_\Omega \Sigma \nabla \phi_j \cdot \nabla \phi_i d\overline{x} \\ &= \sum_j U_j^n \int_\Omega \frac{\phi_j}{\Delta t} \phi_i d\overline{x} - \int_\Omega a(u_h^n - f_r)(u_h^n - f_t)(u_h^n - f_d) \phi_i d\overline{x}. \end{aligned} \tag{6}$$

This leads to the matrix form:

$$\sum_j \left(\frac{M_{ij}}{\Delta t} + K_{ij} \right) U_j^{n+1} = \sum_j \frac{M_{ij}}{\Delta t} U_j^n - F_i^n, \tag{7}$$

where the matrices and vectors are defined as:

$$
\begin{aligned}
K_{ij} &= \int_\Omega \Sigma \nabla \phi_j \cdot \nabla \phi_i \; d\overline{x}, \\
M_{ij} &= \int_\Omega \phi_i \phi_j d\overline{x}, \\
F_i^n &= \int_\Omega a(u_h^n - f_r)(u_h^n - f_t)(u_h^n - f_d)\phi_i d\overline{x}.
\end{aligned} \tag{8}
$$

The resulting linear system for each time step t^n can be written as:

$$
\left(\frac{M}{\Delta t} + K\right) U^{n+1} = \frac{M}{\Delta t} U^n - F^n. \tag{9}
$$

1.4 Modification of `assembleDiffusion`

The provided code `assembleDiffusion` has been extended to take two arguments Σ_h and Σ_d and set the diffusivity value for each element.

1.5 MATLAB code

Code has been implemented in MATLAB to solve the problem. The stiffness matrix is assembled with the extended `assembleDiffusion` script. The mass matrix is assembled using the MATLAB function `assembleMass`. The mass matrix M is assembled using a reference element matrix and the Jacobian determinants of the elements. The reference element for a triangular mesh is defined by the three points $(0, 0)$, $(0, 1)$, and $(1, 0)$. We define the basis functions on this reference element as $\phi_0 = 1 - x - y$, $\phi_1 = x$, and $\phi_2 = y$, giving the reference mass matrix:

$$
M_{\text{ref}} = \frac{1}{24} \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix}.
$$

This matrix is derived from the integral of the product of the basis functions over the reference element. The global mass matrix is then constructed by summing the contributions from all elements, each scaled by the determinant of the Jacobian for the respective element.

The stiffness matrix K is assembled using the `assembleDiffusion` function. This function calculates the local stiffness matrix for each element by integrating the gradients of the basis functions, weighted by the material properties (diffusion coefficients Σ_h and Σ_d). These local matrices are then summed into the global stiffness matrix.

The load vector F is assembled using the `assembleLoadVector` function. The load vector for each element is computed as in 8:

$$
F_i^n = \int_\Omega a(u_h^n - f_r)(u_h^n - f_t)(u_h^n - f_d)\phi_i d\overline{x}.
$$

where u_h^n is the finite element approximation of u at time step n, and ϕ_i are the basis functions. This integral is approximated using a quadrature rule. Specifically, for each element, the integral is evaluated at several quadrature points, with contributions weighted by the quadrature weights and the determinant of the Jacobian for the element. The local load vector is then assembled into the global load vector.

These matrices are then input into Equation 9 to get the solution for each time-step.

1.6

The diffusivity has been modified for the healthy and diseased elements by using the provided flags in the mesh.

dt	mesh	Σ_d	Activation time (ms)		Is M matrix?		Potential exceeds?	
			Normal	Lumped M	Normal	Lumped M	Normal	Lumped M
0.1	128	$0.1\Sigma_h$	32.2	32.9	No	Yes	Yes (-0.002, 1.025)	Yes (1.011)
0.1	256	$0.1\Sigma_h$	32.4	32.6	No	Yes	Yes (1.011)	Yes (1.011)
0.05	128	$0.1\Sigma_h$	30.0	30.6	No	Yes	Yes (-0.011, 1.017)	No
0.05	256	$0.1\Sigma_h$	30.2	30.4	No	Yes	Yes(1.0002)	No
0.1	128	Σ_h	30.6	31.2	No	Yes	Yes (1.001)	Yes (1.011)
0.1	256	Σ_h	30.8	31	No	Yes	Yes (1.011)	Yes (1.012)
0.05	128	Σ_h	28.55	29.15	No	Yes	No	No
0.05	256	Σ_h	28.75	28.9	No	Yes	No	No
0.1	128	$10\Sigma_h$	28.9	29.5	No	Yes	Yes (1.011)	Yes (1.010)
0.1	256	$10\Sigma_h$	29.1	29.2	No	Yes	Yes (1.011)	Yes (1.010)
0.05	128	$10\Sigma_h$	26.95	27.5	No	Yes	No	No
0.05	256	$10\Sigma_h$	27.1	27.25	No	Yes	No	No

Table 1: Results for various dt, mesh configurations and Σ_d. If the boundary on the potential was exceeded the maximum potential is written in parenthesis. Further, we ran all experiments with a lumped matrix and without to compare them.

In Table 1 we see some results and observations from applying FEM to the monodomain problem. Regarding the activation time, we can observe that it decreases with finer mesh resolution, smaller time steps, and higher diffusion coefficients.

In the column specifying lumped M we use a lumped mass matrix. Meaning $M_{Lumped} = diag(sum(M, 2))$ is used instead of M. Without this, some nodes see a negative potential for some time-steps. The cause of this could be that the system matrices are not M-matrices due to having positive off-diagonal elements. The off-diagonal entries in the original mass matrix may introduce numerical errors resulting in a slightly negative potential. Lumping the mass matrix decreases the off-diagonal values and thus makes the system matrices into M-matrices.

We can further observe that even while using the lumped mass matrix, the potential is still exceeded when using larger time-steps.

1.7

For a graphic visualization of the solution, there are videos in in GitHub repository at this link. These videos are named `solution_{mesh size}_{`Σ_d`}_{time steps}.mp4`.

2 Physics Informed Neural Networks (PINNs)

2.1 Define a loss function for solving (1)

Starting from the original PDE from the problem statement:

$$\begin{aligned} \frac{\partial u}{\partial t} &= \nabla \cdot (\Sigma_h \nabla u + \Sigma_d \nabla u) - f(u) \quad \text{in } \Omega \times I, \\ u &= u_0 \qquad\qquad\qquad\qquad\quad \text{at } t = 0, \end{aligned} \tag{10}$$

where $\Omega = (0,1)^2$ and $I = (0, T_f]$, we formulate the PDE residual by taking all terms of the equation on one side and ensuring that they evaluate to 0, effectively solving the equation. Let $\mathbf{x} = (x_1, x_2) \in \Omega$. The PINN will model u to minimize the PDE residual:

$$r(u_\theta, \mathbf{x}, t) := \nabla \cdot (\Sigma_h \nabla u_\theta(\mathbf{x}, t) + \Sigma_d \nabla u_\theta(\mathbf{x}, t)) - f(u_\theta(\mathbf{x}, t)) - \frac{\partial u_\theta(\mathbf{x}, t)}{\partial t} \stackrel{!}{=} 0 \tag{11}$$

To minimize the PDE residual r we chose to use the mean squared error (MSE) over a dataset of collocation points $\mathcal{D}$:

$$\mathcal{L}_{PDE}(u_\theta) = \frac{1}{\mid \mathcal{D}_{pde} \mid} \sum_{(\mathbf{x},t) \in \mathcal{D}_{pde}} ||r(u_\theta, \mathbf{x}, t)||^2 \tag{12}$$

The construction of the collocation points in $\mathcal{D}$ will be detailed in the next subsection.

To enforce homogeneous Neumann boundary conditions we can employ a similar strategy. We must note that we only want to eliminate the spatial gradients perpendicular to the boundary. Forcing the spatial gradients along the boundary to be zero would constrain the solution drastically. Given a set of boundary points $\mathcal{D}_{bc}$, let $\mathcal{D}_{bc}^{x_1} \subset \mathcal{D}_{bc}$ be the subset where $\mathbf{x} = (x_1, x_2)$ contains x_1 such that it takes a boundary value. We then employ the MSE only on the partial derivative $\| \frac{\partial}{\partial x_1} u_\theta(\mathbf{x}, t) \|^2$. Similarly for the subset of boundary points $\mathcal{D}_{bc}^{x_2}$ where x_2 takes a boundary value, we push the partial derivative w.r.t. x_2 to zero. Overall, we obtain the boundary condition loss as follows:

$$\mathcal{L}_{BC} = \frac{1}{\mid \mathcal{D}_{bc}^{x_1} \mid} \sum_{(\mathbf{x},t) \in \mathcal{D}_{bc}^{x_1}} \| \frac{\partial}{\partial x_1} u_\theta(\mathbf{x}, t) \|^2 + \frac{1}{\mid \mathcal{D}_{bc}^{x_2} \mid} \sum_{(\mathbf{x},t) \in \mathcal{D}_{bc}^{x_2}} \| \frac{\partial}{\partial x_2} u_\theta(\mathbf{x}, t) \|^2 \tag{13}$$

Note that because of corner points, $\mathcal{D}_{bc}^{x_1} \cap \mathcal{D}_{bc}^{x_2} \neq \emptyset$ and we penalize both directional gradients for corner points. In the PINN implementation we do consider soft constraints for initial conditions to warm up the training and the corresponding loss is simply defined as a MSE between $u_\theta(\mathbf{x}, t)$ and u_0 when $t = 0$.

$$\mathcal{L}_{IC} = \frac{1}{\mathcal{D}_{ic}} \| u_\theta(\mathbf{x}, t) - u_0(\mathbf{x}) \|^2 \tag{14}$$

The final loss function is composed of a weighted sum of the PDE residual and the boundary condition losses:

$$\mathcal{L} = w_{PDE}\, \mathcal{L}_{PDE} + w_{BC}\, \mathcal{L}_{BC} + w_{IC} \mathcal{L}_{IC} \tag{15}$$

where $\mathcal{L}_{IC}$ is only present if the soft constraints are used. See Section 2.4 for more details. Choosing appropriate weights w_{PDE}, w_{BC} and w_{IC} is crucial for the optimization process. To minimize the effect of those hyperparameters we will learn them instead. We apply gradient ascent on both weights such that they will automatically assign more weight to the component with a higher loss magnitude.

2.2 Generate the dataset for training and explain the strategy

2.2.1 Construct a set of collocation points

To generate a dataset $\mathcal{D}$ of collocation points we relied on Hammersly sampling. We generate two datasets; one for points in the internal domain, called $\mathcal{D}_{pde}$ and one for the bounadries, called $\mathcal{D}_{bc}$. The first dataset is simply composed of sampled points in the $(0, T_f) \times (0,1) \times (0,1) = I \times \Omega$ domain. The second dataset is instead initially sampled from the $(0, T_f) \times (0,1)$ domain, resulting in 2-dimensional points sampled in the time and one spatial dimension. We then add the third

dimension by padding the 2-dimensional samples with the boundary values, namely 0 and 1. The 2-dimensional points are of the form (t, x) with $t \in (0, T_f)$ and $x \in (0, 1)$. For each boundary point, we add the third dimension as $(t, 0, x)$, $(t, 1, x)$,$(t, x, 0)$ and $(t, x, 1)$.

2.2.2 Devise a strategy to identify if the collocation points belong to domain Ω_h or Ω_d

For each diseased area Ω_{d1}, Ω_{d2} and Ω_{d3} , modeled as a circle, we know its center and radius. To determine whether a collocation point is in Ω_h or Ω_d, we go through all of them and determine whether the Euclidean distance between the point and any of the three centers is less than the corresponding radius. We assume that the circles do not overlap and as soon as a point is within the radius distance, we assign a label between 1 and 3, while label 0 is reserved for Ω_h. Figure 1 shows collocation points in the space domain Ω colored according to their aforementioned label. We then can map each label to the specific values of Σ_h and Σ_d.

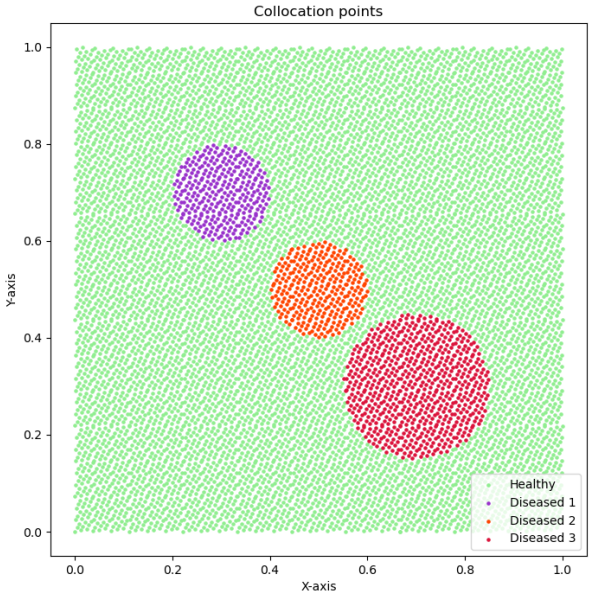


Figure 1: Collocation points colored according to region they belong to, either Ω_h or Ω_d.

2.2.3 Employ this strategy to construct a pair of collocation points and diffusitivity parameters.

In the previous sections, we obtained a dataset of collocation points of shape $(b, 3)$ where $b =\mid \mathcal{D}_{pde} \mid$ and a label for each point representing the domain they belong to in a vector of shape $(b, 1)$. We then mapped each label to its corresponding electrical diffusivity parameters, depending on the domain in which it is found. The pairs of collocation points and their electrical diffusivity parameter are implicitly formed by accessing the i-th collocation point and the i-th parameter.

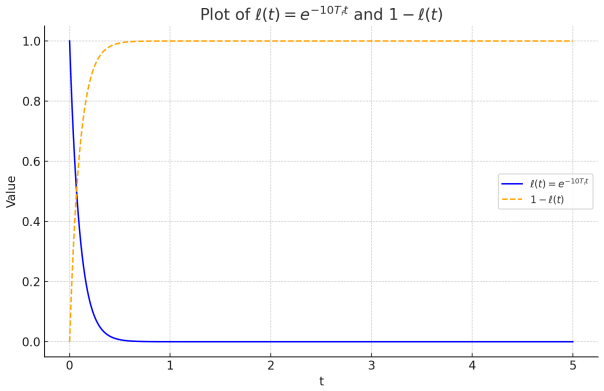


Figure 2: Visualization of $\ell(t)$ to see its behavior for a max time interval of $T_{\{} = 1$.

2.3 Construct a function to compute the initial condition and employ the hard constraints approach to enforce the initial condition in an exact manner.

To impose hard constraints for the initial conditions we can modify the PINN to have its output composed of a linear combination of a trainable component u'_θ and the known initial conditions u_0. We then want to enforce that that $u_\theta(\mathbf{x}, 0) = u_0(\mathbf{x})$. This can be achieved by defining a function $\ell(t)$ such that for $t = 0$ the output coming from the trainable component is eliminated. For $t > 0$, the overall output is composed of both initial conditons and the neural network's output, which has to learn to counteract the initial conditions contribution to fit the equation and boundary conditions. The new model's output can thus be defined as:

$$u_\theta(\mathbf{x}, t) = \ell(t)\ u_0 + (1 - \ell(t))\ u'_\theta(\mathbf{x}, t) \tag{16}$$

We chose to use the length factor function ℓ to be defined as:

$$\ell(t) = \exp(-10t) \tag{17}$$

which is visualized in Figure 2. Utilizing exponentials instead of linear interpolation reduces the influence of the initial condition as time proceeds. The affine combination saturates quicker compared to a linear interpolation. This can reduce the learning complexity for the underlying neural network u'_θ as it doesn't need to unlearn the effects of the length scale function throughout the entire time interval but only a smaller period.

2.4 Train the model using your favorite optimizer.

Training the model turned out to be challenging. Our PINN is a multi-layer perceptron (MLP) with hyperbolic tangent activation functions between hidden layers. We use a linear activation function (i.e no activation) for the output layer to perform regression. While exploring different hyperparameters we investigated various methods:

- The non-linear activation functions are trainable such that their steepness can adapt to our task. This means that given the output x of the linear layer, the output of the activation is $x' = \tanh(\alpha x)$ where parameter α is a learnable parameter that is unique for each activation.

We assumed that given the discontinuities in the problem, allowing the model to adapt its activations, such as making them very steep, could help.

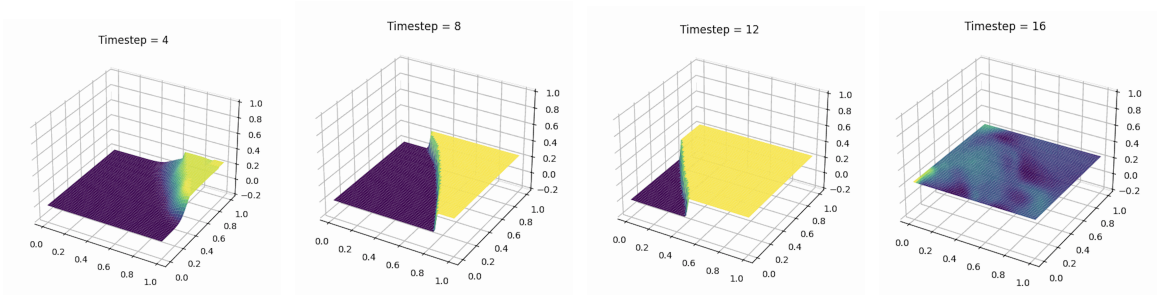
- Furthermore, the weights w_{pde} and w_{bc} are scalars that can be learnable too; after computing the gradients on the computational graph, we apply gradient ascent. We chose Adam to do so although manual gradient ascent could have been applied too.
- Since we assume that the system should propagate the impulse from the initial condition over the entire domain we realized that the model has to learn to model steep changes in both the spatial and temporal domain. Especially very close to the initial condition it is faced with discontinuities. Therefore it seemed reasonable to apply Fourier features to the input domain. This allows the model to model high-frequency changes better.
- Since the hard initial condition introduces discontinuities we tried to enforce the initial condition weakly so the model could focus more on the dynamics of the system and after approximately half the epochs we activated the hard constraint hoping that the model could cope with it better now that it's already close to the actual solution.
- We generally used Adam, however, we were rarely satisfied with the solutions it converged to. Still, Adam is fast, so we optimized with Adam first and later switched to LBFGS which typically converged to much better solutions.
- Lastly, we observed that especially in the beginning most of the domain is constant and only small areas undergo a change. Since a constant function that does not change over time at either of f's roots is a solution to the monodomain equation, we figured that learning the zero function (zero is a root of f) would result in a very small pde residual across large parts of the domain. Therefore we feared that this large area (where it is trivial to converge) would average out the loss. The model could simply get the areas wrong where change should happen as it becomes negligible compared to the broad constant zero area. The model would find more interest in pursuing a zero function than focusing on the monodomain dynamics. We introduced trainable weights for every data point to help the model focus on the relevant areas. The weights are trained with gradient ascent to increase the weights strongly where the dynamics are violated. Unfortunately, it seemed to have little effect.

2.5 After the training, compute a solution (uNN). (Hint: For creating the video, fix the value of $t = 0, dt, 2dt, 3dt, \ldots, Tf$. Obtain the value of uNN at a given time, stack all these values at each time step, and stack these values similarly to previous tasks to generate a video. Feel free to use any libraries or move the data to MATLAB.)

We created gifs of our solutions, where some frames were used to create Fig. 3 and Fig. 4. Further, we attached them to our submission and our GitHub.

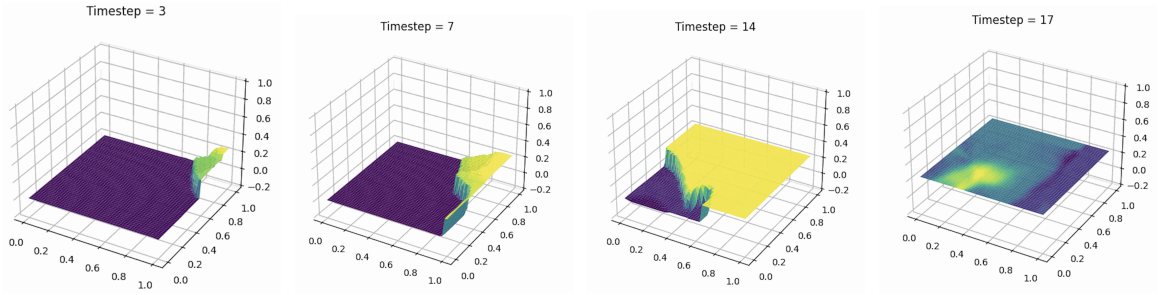
2.6 Compare the results of the FEM with the PINNs solution and state your conclusions.

As mentioned, obtaining meaningful results with the PINN has been very hard. The PINN tended to learn the 0 function or other functions that did not look as expected. The results that made more sense always converged to ~ 0.238, one of the roots of f. We present in Figure 3 a result for the case



(a) (b) (c) (d)

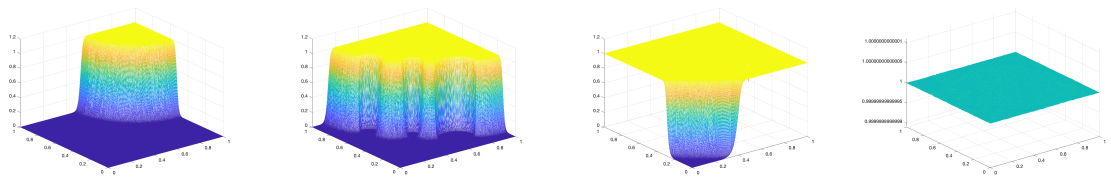
Figure 3: PINN result on the healthy tissue with $T_f = 5$.



(a) (b) (c) (d)

Figure 4: PINN result on the tissue containing diseased areas with $T_f = 5$.

without diseased areas and $T_f = 5$. The corresponding gif file can be found here. Compared to the FEM results we clearly see that the value of convergence is not the same and that the convergence is reached way too quickly. In the figure, the timestep value is between 0 and 50, corresponding to the index in the time dimension for the test tensor of points in $I \times \Omega$ of shape $(50, 50, 50)$. In Figure 4, we considered the case with diseased areas having parameter $\Sigma_d = 10\Sigma_h$. The git can be found at here. Comparing the results with the FEM, the same remarks done previously apply here too. Additionally, we can see that the diseased areas have a contribution in the behaviour of the solution over time. The case having $\Sigma_d = 0.1\Sigma_h$ did not produce meaningful results after trying many PINN hyperparameter combinations, and we therefore do not report any result for it. We can conclude that this problem is very hard to handle with a PINN, in particular because of the discontinuities and steep function that needs to be learned, and handling the time correctly. FEM on the other hand, while it requires more theoretical work to set up the system, manages to nicely generate a solution to the problem. We provide in Figure 5 4 snapshot of the solution using FEM on a fine mesh, using $\Sigma_d = 0.1\Sigma_h$ and $T_f = 35$.



(a) timestep 175/700. (b) timestep 350/700. (c) timestep 525/700. (d) timestep 700/700.

Figure 5: FEM results with $\Sigma_d = 0.1\Sigma_h$, $T_f = 35$ and $dt = 0.05$ on the finest mesh.