Bidomain Model

Author : Grégoire Pourtier (group 2, biology with Conrad Schloer)
Supervisor : Jürgen Fuhrmann

Table of Contents

1. Introduction : problem overview

The bidomain equations was first suggest by Otto Schmitt, an american enginner, in 1969 before being formulated mathematically at the end of 70s. It model the propagation of electrical potential waves in the myocardium, also called heart muscle.

The bidomain equations are part of a long list of models that describe the phenomena of cardiac electrophysiology. It is the most complete mathematical model for describing the spread of cardiac electrical activity. An important application of this model is for simulating defibrillation. The model is expressed through a nonlinear system of partial differential equations, composed of a coupled parabolic and elliptic partial differential equations and one ordinary differential equation (originally just a set of coupled partial differential equations).

The coupled partial differential equations desribed two electrical potential, u_i which is the intracellular potential and u_e which is the extracellular potential. Then the ordinary differential equation models the ion activity, v, through the cardiac cell membranes. We refered to a coupled set of partial differential equations since the intracellular and extracellular potentials are solved simultaneously. We chose to use the representation introduced in [1] for the bidomain model's equations. We have the following three species problem :

$$\begin{aligned} \frac{\partial u}{\partial t} = \frac{1}{\epsilon} f(u,v) + \nabla \cdot (\sigma_i \nabla u) + \nabla \cdot (\sigma_i \nabla u_e) \qquad (1) \\ \nabla \cdot (\sigma_i \nabla u + (\sigma_i + \sigma_e) \nabla u_e) = 0 \qquad (2) \\ \frac{\partial v}{\partial t} = \epsilon g(u,v) \qquad (3) \end{aligned}$$

where

$$u = u_i - u_e$$

u is defined as the action or transmembrane potential, σ_i and σ_e are second order tensors representing, respectively, the intracellular and the extracellular tissue’s electrical conductivity in each spatial direction and ϵ is a parameter linked to the ratio between the repolarization rate and the tissue excitation rate. γ is a parameter controlling the ion transport and β is related to the cell excitability. Moreover, we decided to follow the model from [1] to describe the ionic activity across the cellular membrane, i.e. that we're using the FitzHugh– Nagumo equations, which are a simplification of the Hodgkin–Huxley equations, therefore we can write f and g as :

$$\begin{aligned} f(u,v) = u - \frac{u^3}{3} - v \\ g(u,v) = u + \beta - \gamma v \end{aligned}$$

If we investigate into the formulation of the problem above, we noticed that the system of partial differential equations models a nonlinear reaction-diffusion system.

By examining the system, we can see that the first equation of the system depends on time (since it's a parabolic PDE), there is two flux terms, which are linear and one nonlinear reaction term. The source term is equal to zero. Then in the second PDE, there is only one linear flux defined (no reaction and source term). Finally the ODE left is defined with a reaction term.

During this project we had time to investigate the implementation of the 1D problem and to start looking at the 2D problem. We implemented the problem by using the VoronoiFVM.jl package, which is a solver for coupled nonlinear partial differential equations based on the Voronoi finite volume method.

To facilitate the implementation with the **VoronoiFVM.jl**. package, we can rewrite the equations to have the flux and reaction terms on the left hand side and the source on the right hand side. We obtain :

$$\frac{\partial u}{\partial t} - \nabla \cdot (\sigma_i \nabla u + \sigma_i \nabla u_e) - \frac{1}{\epsilon}(u - \frac{u^3}{3} - v) = 0$$

$$-(-\nabla \cdot (\sigma_i \nabla u + (\sigma_i + \sigma_e) \nabla u_e)) = 0$$

$$\frac{\partial v}{\partial t} - \epsilon(u + \beta - \gamma v) = 0$$

since the operator $\nabla \cdot$ is linear.

2. Implementation of the numerical methods

To solve the problem described in the previous section, we can't find an analytical solution so we have to set up a numerical method which will compute an approximate function using a numerical method. We chose to use the voronoi finite volume method to solve our problem.

First we have to define a discretization. The idea is that instead of solving a continuous problem, we solve a large algebraic system called the discrete problem, i.e. we search the value of the unknown functions in a large number of points.

To obtain this discrete problem we're using the FVM which has the advantage of conserving local flux. We will have first to divide our computational domain into a finite number of representative elementary volumes. Then from the divergence form of our problem we will be able to apply the Gauss theorem. This theorem enables to convert the volume integral over the divergence into a surface integral across the boundaries. So instead of integrate the fluxes inside the cells, we integrate these fluxes across the boundary of the cells. This simplifies the PDE substantially.

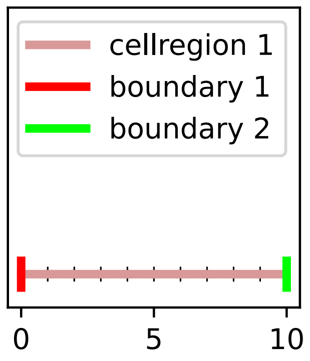
Moreover since one of the partial differential equations of our system depends on time, we will have to discretize in time and in space. To do that, we will use the Rothe method, which consists to first discretize in time and then in space.

2.1 Finite volume space discretization approach

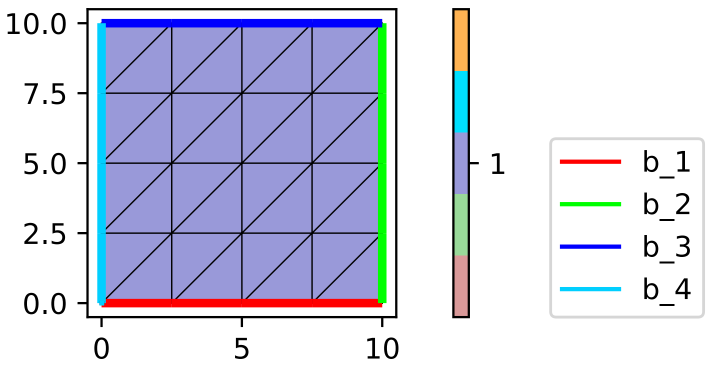
We approximate the solution $u, u_e \text{ and } v$ (our three species) from the bidomain model only with respect to the space discretization. This will result in a spatial semi-discretization. We optain this spatial semi-discretization of the bidomain model through the voronoi finite volume method.

First we need to generate a mesh. This will be done by the package **ExtendableGrids.jl**.

For the 1D case, we create a 1D grids from a vector of monotonicaly increasing x-axis positions. We obtain :



Then for the 2D case, we create a 2D tensor product grids from two vectors of x and y coordinates. This results in the creation of a grid of quadrilaterals. Then, each of them is subdivided into two triangles, resulting in a boundary conforming Delaunay grid. We obtain :



```
create_grid (generic function with 1 method)
```

Once we subdivided the computational domain into a finite number of edge (in 1D) or triangle (in 2D), we can build the Voronoi cells which define the representative elementary volumes (or control volumes) for our finite volume method. These voronoi cells respect particular properties which will be useful to solve our problem (e.g. orthogonality of the interfaces between two collation points).

The collocation points of our control volumes are defined as the vertices from the mesh we generated earlier.

Our computational domain is now subdivided in a finite number of control volumes.

We can start by discretizing the continuity equation in each PDEs of the system. We rewrite our system as following:

$$u_t - \nabla \cdot \overrightarrow{j_1} + r_1(u,v) = 0 \qquad \textbf{(1)}$$

$$\text{with } \overrightarrow{j_1} = \sigma_i(\nabla u + \nabla u_e) \;\text{ and }\; r_1(u,v) = -\frac{1}{\epsilon} f(u,v)$$

$$-\nabla \cdot \overrightarrow{j_2} = 0 \qquad \textbf{(2)}$$

$$\text{with } \overrightarrow{j_2} = \sigma_i \nabla u + (\sigma_i + \sigma_e)\nabla u_e$$

$$u_t + r_2(u,v) = 0 \qquad \textbf{(3)}$$

$$\text{with } r_2(u,v) = -\epsilon g(u,v)$$

Then we integrate the spatial derivative over the control volumes ω_k.
For the equation (1), we have :

$$u_t - \int_{\omega_k} \nabla \overrightarrow{j_1} \, \mathrm{d}\omega + \int_{\omega_k} r_1(u,v) \, \mathrm{d}\omega = 0$$

$$\Rightarrow u_t - \int_{\partial\omega_k} \overrightarrow{j_1} \cdot \overrightarrow{n}_\omega \, \mathrm{d}s + \int_{\omega_k} r_1(u,v)\, \mathrm{d}\omega = 0 \qquad \text{(by Gauss' law)}$$

$$\Rightarrow u_t - \sum_{l \in N_k} \int_{\sigma_{kl}} \overrightarrow{j_1} \cdot \overrightarrow{\nu}_{kl} \, \mathrm{d}s + \sum_{m \in G_k} \int_{\gamma_{kl}} \overrightarrow{j_1} \cdot \overrightarrow{n}_m \, \mathrm{d}s + \int_{\omega_k} r_1(u,v)\, \mathrm{d}\omega = 0$$

For the equation (2), we have :

$$-\int_{\omega_k} \nabla \overrightarrow{j_2} \, \mathrm{d}\omega = 0$$

$$\Rightarrow -\int_{\partial\omega_k} \overrightarrow{j_2} \cdot \overrightarrow{n}_\omega \, \mathrm{d}s = 0 \qquad \text{(by Gauss' law)}$$

$$\Rightarrow -\sum_{l \in N_k} \int_{\sigma_{kl}} \overrightarrow{j_2} \cdot \overrightarrow{\nu}_{kl} \, \mathrm{d}s - \sum_{m \in G_k} \int_{\gamma_{kl}} \overrightarrow{j_2} \cdot \overrightarrow{n}_m \, \mathrm{d}s = 0$$

For the equation (3), we have :

$$u_t + \int_{\omega_k} r_2(u,v)\, \mathrm{d}\omega = 0$$

We are able, thanks to the collocation points, to approximate the fluxes by finite difference using the neigboring control volumes. We define flux functions :

$$\text{for } \overrightarrow{j_1} = \sigma_i(\nabla u + \nabla u_e), \text{we use } g_1(u^k, u^l, u_e^k, u_e^l) = \sigma_i(u^k - u^l) + \sigma_i(u_e^k - u_e^l)$$

for $\overrightarrow{j_2} = \sigma_i \nabla u + (\sigma_i + \sigma_e)\nabla u_e$, we use $g_2(u^k, u^l, u_e^k, u_e^l) = \sigma_i(u^k - u^l) + (\sigma_i + \sigma_e)(u_e^k -$

These flux functions will be important in the implementation of our problem with the VoronoiFVM.jl package.

Given a control volume ω_k, we now integrate the different equations of the system of PDEs over space-time control volume $\omega_k \times (t^{n-1}, t^n)$, divide by τ^n :
For the equation (1) :

$$\int_{\omega_k} (\frac{\partial u}{\partial t} - \nabla \cdot (\sigma_i \nabla u + \sigma_i \nabla u_e) - \frac{1}{\epsilon} f(u,v)) \, \mathrm{d}\omega = 0$$

$$\Rightarrow \int_{\omega_k} (\frac{1}{\tau^n}(u^n - u^{n-1}) - \nabla \cdot (\sigma_i \nabla u + \sigma_i \nabla u_e) - \frac{1}{\epsilon} f(u,v)) \, \mathrm{d}\omega = 0$$

$$\Rightarrow \frac{1}{\tau^n} \int_{\omega_k} (u^n - u^{n-1}) \, \mathrm{d}\omega - \int_{\omega_k} \nabla \cdot (\sigma_i \nabla u + \sigma_i \nabla u_e) \, \mathrm{d}\omega - \int_{\omega_k} \frac{1}{\epsilon} f(u,v)) \, \mathrm{d}\omega = 0$$

$$\Rightarrow -\sum_{l \in N_k} \int_{\sigma_{kl}} (\sigma_i \nabla u + \sigma_i \nabla u_e) \cdot \vec{n}_{kl} \, \mathrm{d}\gamma - \int_{\gamma_k} (\sigma_i \nabla u + \sigma_i \nabla u_e) \, \mathrm{d}\gamma - \int_{\omega_k} \frac{1}{\epsilon} f(u,v)) \, \mathrm{d}\omega$$

$$+ \frac{1}{\tau^n} \int_{\omega_k} (u^n - u^{n-1}) \, \mathrm{d}\omega = 0$$

$$\Rightarrow \underbrace{\frac{|\omega_k|}{\tau^n}(u^n - u^{n-1})}_{\to M} - \underbrace{\sum_{l \in N_k} \frac{|\sigma_{kl}|}{h_{kl}} (\sigma_i (u^k - u^l + u_e^k - u_e^l)) - \sum_{m \in G_k} |\gamma_{km}| (\alpha_m u^k + \beta_m u_e^k)}_{\to A}$$

$$\underbrace{-\frac{|\omega_k|}{\epsilon} f(u,v)}_{F(u,v)}$$

$$\Rightarrow \frac{1}{\tau^n}(Mu^n - Mu^{n-1}) - A(u + u_e) + F(u,v) = 0$$

For the equation (2):

$$\int_{\omega_k} \nabla \cdot (\sigma_i \nabla u + (\sigma_i + \sigma_e) \nabla u_e) \, \mathrm{d}\omega = 0$$

$$\underbrace{\sum_{l \in N_k} \frac{|\sigma_{kl}|}{h_{kl}} \sigma_i (u^k - u^l) + \sum_{m \in G_k} |\gamma_{km}| (\alpha_m u^k)}_{\to A_1} + \underbrace{\sum_{l \in N_k} \frac{|\sigma_{kl}|}{h_{kl}} (\sigma_i + \sigma_e)(u_e^k - u_e^l) + \sum_{m \in G_k} |\gamma_{km}| (\alpha_n}_{\to A_2}$$

$$\Rightarrow A_1 u + (A_1 + A_2) u_e = 0$$

For the equation (3) :

$$\underbrace{\frac{|\omega_k|}{\tau^n}(v^n - v^{n-1})}_{\to M_1} + -\epsilon \underbrace{|\omega_k| g(u,v)}_{G(u,v)}$$

$$\Rightarrow M_1 v - \epsilon G(u,v) = 0$$

From this we obtain a large (depending on the number of REV), sparse matrix that we have to solve either by using direct or iterative solvers.

2.2 Time discretization approach

We now look at the time discretization which will consist of replacing the time derivative in the equations (1) and (3) by a difference quotient.

The strategy to solve a parabolic initial boundary value problem is that we are not trying to solve the whole problem at once but we consider a sequence of solutions by time integrations.

Let $\tau > 0, \quad \tau = \dfrac{T}{N_k}, \; N_k \in \mathbb{N}$ be the time stepsize.

For i = 1 ... N_k , solve

$$\frac{u^n - u^{n-1}}{\tau} - \nabla \cdot (\sigma_i \nabla u^\theta + \sigma_i \nabla u_e^\theta) - \frac{1}{\epsilon}(u^\theta - \frac{(u^3)^\theta}{3} - v^\theta) = 0$$

$$-\nabla \cdot (\sigma_i \nabla u^\theta + (\sigma_i + \sigma_e) \nabla u_e^\theta) = 0$$

$$\frac{u^n - u^{n-1}}{\tau} - \epsilon(u^\theta + \beta - \gamma v^\theta) = 0$$

with $u^\theta = \theta u^n + (1-\theta)u^{n-1}$, $u_e^\theta = \theta u_e^n + (1-\theta)u_e^{n-1}$ and $v^\theta = \theta v^n + (1-\theta) v^{n-1}$

We can already notice that the equation (2) of the system doesn't depend on time, so we will have to solve a system equations for each step time.
There is several possibilities to conduct the time discretization on our equation (1) and (3) (lot of them are introduces in [1]). We chose to look at the θ-scheme. The θ-scheme has three special cases :

- for $\theta = 1$: backward (implicit) Euler method
- for $\theta = \frac{1}{2}$: Crank-Nicolson scheme

- for $\theta = 0$: forward (explicit) Euler method

We present the backward Euler method :

$$M\frac{u^{n+1} - u^n}{\Delta t} - A(u^{n+1} + u_e^{n+1}) + F(u^{n+1}, v^{n+1}) = 0$$

$$M\frac{v^{n+1} - v^n}{\Delta t} - \epsilon G(u^{n+1}, v^{n+1})$$

with Δt defined as a constant time step.

3. Numerical solutions methods

We performed our space and time discretization of our computational domain. We can now solve the system our bidomain equations. We look at the full discretization which consists of combining the voronoi finite volume method and some Euler-scheme (here we implemented the backward euler method).

The backward and forward Euler method have a first order convergence in time and the CN method has a second order convergence in time. Moreover for the backward and CN method, to compute the solution for one time step we have to solve one system of equations, this is not the case with the forward Euler method (only a product matrix vector).

The explicit Euler method could be consider since it's the same order of convergence as the implicit Euler method but there is an advantage to use this scheme because we don't have to solve a system of equations to get a solution.

The three special cases from our θ-scheme are studied in detail in [1]. The stability in the L^2 norm is investigated. From [1], we see that for the backward Euler scheme, the stability of the method holds if and only if the time step $\Delta t = O(\epsilon)$. Then for the explicit Euler method, we need to add more constraints to the time step (see 3.2.4 of [1]). The time step Δt has to be $O(h^2)$, where h is the grid size. This condition for stability limits the use of the forward Euler method.

The cheap explicit Euler method only works for very small time steps. That is a compensation with respect to the fact that we don't have to solve a linear system.

We chose to use a backward Euler scheme so we don't worry much about the stability of the scheme.

4. Results of the Bidomain Model

We present in this section the numerical results we obtained for the 1D case. We will look at the 1D results on a 1D grid and then the 1D results on a 2D grid.
We used the **VoronoinFVM.jl** package to solve the bidomain problem for the 1D case.

4.1 Solve 1D

We solved the 1D case of the bidomain model on the domain $[0, L]$, with $L = 70$.
The 1D bidomain model is written as :

$$\frac{\partial u}{\partial t} - \frac{\partial}{\partial x}(\sigma_i \frac{\partial u}{\partial x} + \sigma_i \frac{\partial u_e}{\partial x}) - \frac{1}{\epsilon}(u - \frac{u^3}{3} - v) = 0$$
$$-(-\frac{\partial}{\partial x}(\sigma_i \frac{\partial u}{\partial x} + (\sigma_i + \sigma_e)\frac{\partial u_e}{\partial x})) = 0$$
$$\frac{\partial v}{\partial t} - \epsilon(u + \beta - \gamma v) = 0$$

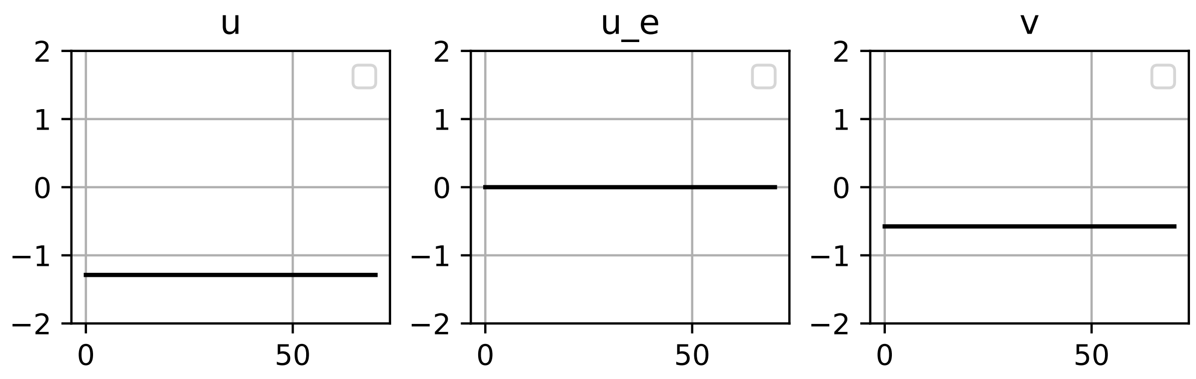
We implemented this system with homogeneous Neumann boundary conditions such that $\partial_x u(0) = \partial_x u(L) = 0$ and $\partial_x u_e(0) = \partial_x u_e(L) = 0$.

Since we have pure Neumann boundary conditions, we add a supplementary condition otherwise the linear system of equations to solve would be singular. We define the homogeneous Dirichlet boundary condition $u_e(0) = 0$.

We introduced the constant from $[1]$, i.e. $\epsilon = 0.1, \beta = 1.0, \gamma = 0.5$ and the conductivity tensors (in 1D, scarlars) $\sigma_i = \sigma_e = 1.0$.

We implemented the discrete problem from the section 2 with the VoronoiFVM.jl package. It is described in the functions storage, reaction and flux. Since we have a discrete problem with some nonlinearity, the VoronoiFVM.jl package is using the Newton iteration scheme to solve the nonlinear system of equations.

In a first part, we solved the stationary problem, i.e. without the time derivative component.



Later we looked at the unstationary problem. We define $T = 50$ which is the time end of the

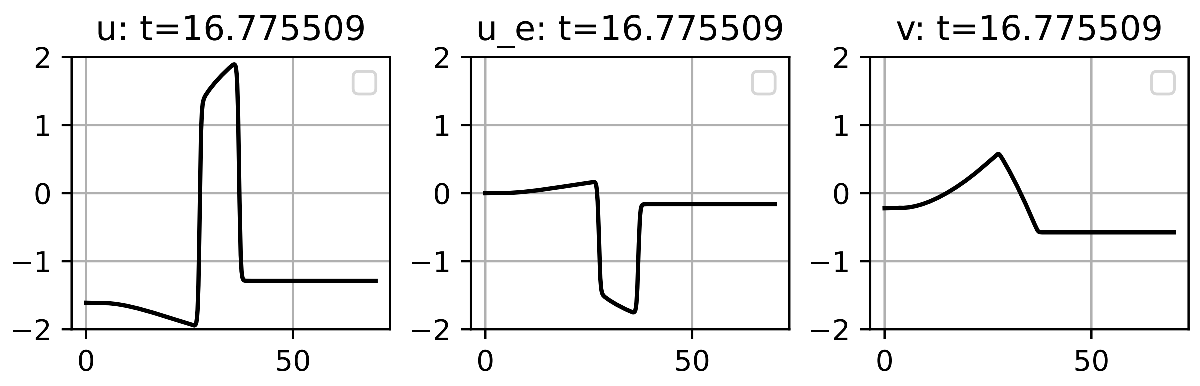
evolution of our system of PDEs. We chose this time so we have time to see the system reach the stationary state.
We take the species u and v at the equilibrium value of the system for the initial conditions of ou
time dependent problems.

$$\begin{cases} f(u,v) = 0 \\ g(u,v) = 0 \end{cases}$$

$$\Rightarrow \begin{cases} u - \frac{u^3}{3} - v = 0 \\ u + \beta - \gamma v = 0 \end{cases}$$

So we will have to solve a nonlinear system of equations to find the initial value of the specie u and v. For the species u_e, we choose u_e constant at 0, except for the intervall $[0, \frac{1}{20}L]$ where we apply the excitation so that $u(x) = 2$.

time=



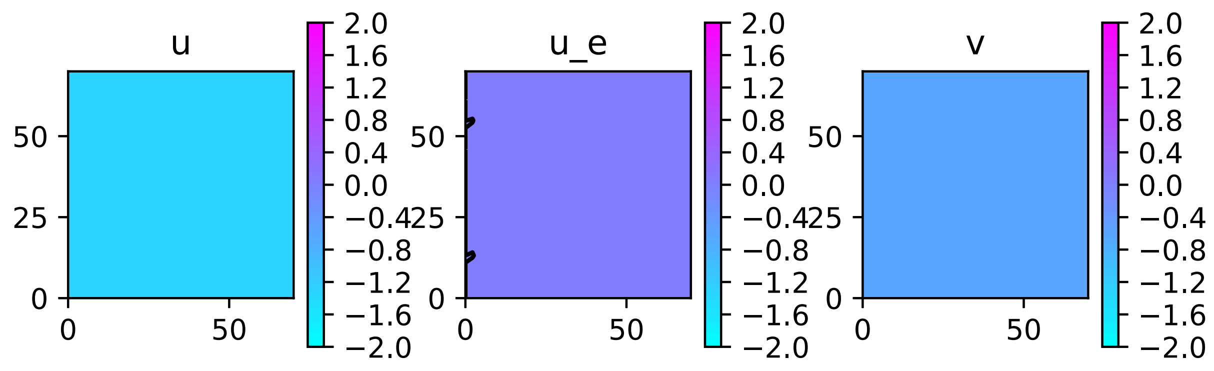
We computed the solution for a rather coarse grid (N = 0.07) with the backward Euler method. We chose a arbitrary time step dt = 0.01.

After several tests, we can see that we obtain a numerical solution for different grid sizes and time steps. The parameter dtgrowth from our implementation, which is defining how fast we increase the timestep (by factor) when approaching stationary is important because if it's above 1.005, we notice that we obtain some convergence error.

4.2 Solve 1D on a 2D grid

We take the same constants as for the previous test. We only change the grid where we solve the problem. This time we take a 2D tensor product mesh.

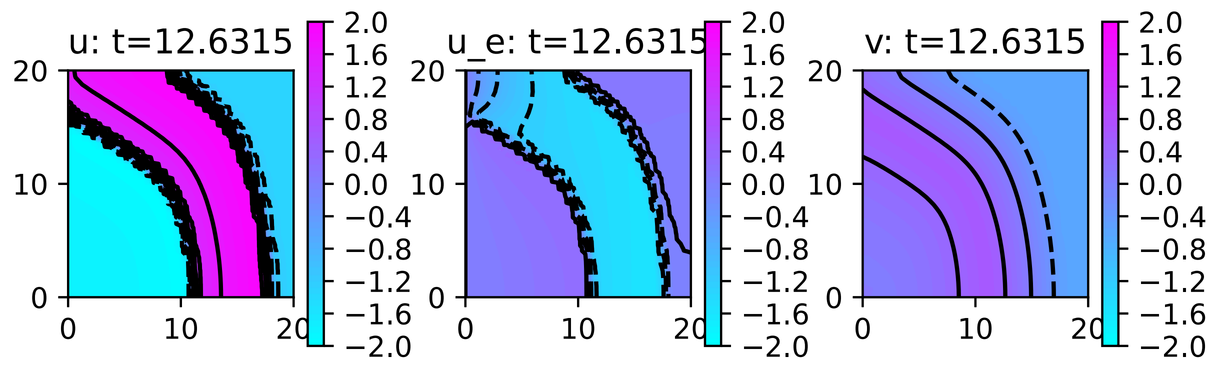
For the stationary problem only the grid changes.



For the unstationary problem we need to modify the initial value, since now we have to take the interval $[0, \frac{1}{20}L]$ in the both directions (x-axis and y-axis).

As result for the backward Euler scheme, we obtain :

time=



5. Performance improvement

Improve the performance to solve one model is important, so we can have better accuracy and a faster time to solve the equations.

In our case, several tracks could be study to improve the performances. We could look into different time discretization to see if the system is solved faster. We could also look into second or third order scheme for a better accuracy of the solutions.

From [1], we can see that a semi-implicit method would be a good idea to implement. It would enable us to obtain a good numerical solution without worrying about stability conditions but also these methods don't need to solve a system of equations for each time step.

Unfortunately, we didn't have enough time to implement the 2D problem.

6. Julia functions

```
evolution (generic function with 1 method)
```

```
grid1d_a = ExtendableGrids.ExtendableGrid{Float64,Int32};
           dim: 1 nodes: 1001 cells: 1000 bfaces: 2
```

```
bidomain_stationary (generic function with 1 method)
```

```
result_bidomain_stationary =
3×1001 Array{Float64,2}:
 -1.28791      -1.28791      -1.28791      …  -1.28791      -1.28791
 -1.80914e-60   1.04792e-31   1.82916e-31       5.29208e-29   5.29263e-29
 -0.57582      -0.57582      -0.57582         -0.57582      -0.57582
```

```
grid2d_a = ExtendableGrids.ExtendableGrid{Float64,Int32};
           dim: 2 nodes: 1089 cells: 2048 bfaces: 128
```

```
result_bidomain_stationary1d_2dgrid =
3×1089 Array{Float64,2}:
 -1.28791      -1.28791      -1.28791      …  -1.28791      -1.28791
 -1.58056e-62   1.11631e-32   1.68765e-32       1.50025e-31   1.57503e-31
 -0.57582      -0.57582      -0.57582         -0.57582      -0.57582
```

```
bidomain (generic function with 1 method)
```

```
result_bidomain =
  (times = Float64[0.0, 0.01, 0.02005, 0.0301502, 0.040301, 0.0505025, 0.060755,
```

```
result_bidomain_1d_2dgrid =
  (times = Float64[0.0, 0.01, 0.02005, 0.0301502, 0.040301, 0.0505025, 0.060755,
```

References

[1] *Semi-implicit time-discretization schemes for the bidomain model* by Marc Ethier and Yves Bourgault.

[2] *Existence and uniqueness of the solution for the bidomain model used in cardiac electrophysiology* by Yves Bourgault, Yves Coudière, Charles Pierre.