University of Bordeaux

Master 1 - 4TMS702U

Cardiac Electrophysiology Simulations

Authors:
Borjan Geshkovski
Thibaut Guégan
Charlotte Rodriguez

Supervisor:
Yves Coudière

November 28, 2017

Contents

Abstract

The aim of this work was to discretize a system of continuous partial differential equations, and code a simple implementation of this discretization in `Python`. The system in question represents a minimal model of the dynamics of wave propagation of the action potentials in human ventricular tissue.

1 Description of the problem

1.1 Physical problem

The physical problem concerns modelling the dynamics of wave propagation of the action potentials in human ventricular tissue. The model we study reproduces realistic characteristics of the tissue. We use the minimal model, which is designed to reproduce tissue-level characteristics of epicardial and endocardial cells. It is for this reason that we use ENDO (*ENDOcardial*) and EPI (*EPIcardial*) data for the simulations.

More precisely, ***epicardial cells*** *are cells of the* ***outer layer*** *of the heart called the pericardium: a double-walled sac containing the heart and the roots of the great vessels.* ***Endocardial cells*** *are cells of the endocardium: the* ***innermost layer*** *of tissue that lines the chambers of the heart.*

It is a minimal model of the action potentials of human ventricular myocytes (*muscle cells*), where resting membrane potential, threshold for excitation, upstroke, action potential morphology, and action potential duration (APD) and conduction velocity (CV) rate dependence were fitted. It contains the minimum number of variables necessary for an action potential model that can reproduce arbitrary APD and CV restitution curves as well as a range of realistic AP shapes.

1.2 Aim of this project

The aim of this project is to discretize the system of continuous equations given below, which represents the minimal model, and to approximate the solution, as well as doing simulations by programming this discretization in `Python`. We then put different type of pulsations to see the propagation of the action potential.

1.3 Discretization of the continuous problem

We discretize the following continous problem:

$$(P)\begin{cases} \partial_t u = \tilde{D}\Delta u - J \\ \partial_t v = \frac{(1-H(u-\theta_v))(v_\infty - v)}{\tau_v^-} - \frac{H(u-\theta_v)v}{\tau_v^+} \\ \partial_t w = \frac{(1-H(u-\theta_w))(w_\infty - w)}{\tau_w^-} - \frac{H(u-\theta_w)w}{\tau_w^+} \\ \partial_t s = \frac{(1+\tanh(k_s(u-u_s)))-2s}{2\tau_s} \\ \frac{\partial u}{\partial n} = 0 \end{cases}$$

where by H we denote the Heaviside function, by $\tilde{D}$ a real scalar, and by $\frac{\partial u}{\partial n}$ Neumann homogeneous boundary conditions. J is dependent on the variables u, v, w, and s. The formulation of the parameters is the same as that of the paper *Minimal model for human ventricular action potentials in tissue*.

A discretization of the Laplacian is necessary. For the time-wise discretization of the first equation, we chose the term Δu to be implicit, and we discretize $\partial_t u$ by using the Euler explicit method. Consequently, at each step of time, the associated matrix has to be inverted:

$$\partial_t u^{n+1} = \tilde{D}\Delta u^{n+1} - J^n.$$

Therefore the discretization we obtain is the following:

$$\frac{u^{n+1} - u^n}{dt} = \tilde{D}\Delta_{approx} u^{n+1} - J^n$$

which can be rewritten as such:

$$(I - dt\Delta_{approx})u^{n+1} = u^n - dtJ^n.$$

Discretization of the Laplacian Δ (space-wise):

Here we see the solution (u) of the problem (P) as a function depending on two variables x and y (space coordinates):

$$\begin{aligned} u : \mathbb{R}^2 &\to \mathbb{R} \\ (x, y) &\mapsto u(x, y), \end{aligned}$$

and the 2D Laplacian being:

$$\Delta u = \partial^2_{x^2} u + \partial^2_{y^2} u.$$

We discretize the second derivatives $\partial^2_{x^2}$ and $\partial^2_{y^2}$ using a centered second order finite difference method, with five points.

Meshing:

We suppose that we have at our disposal a square piece of cardiac tissue (we call it $\Omega =]L_x \max, L_x \min[^2)$ and we mesh a cartesian grid. The solution is approximated at the intersections of the vertical and the horizontal lines. The step dx is uniform.

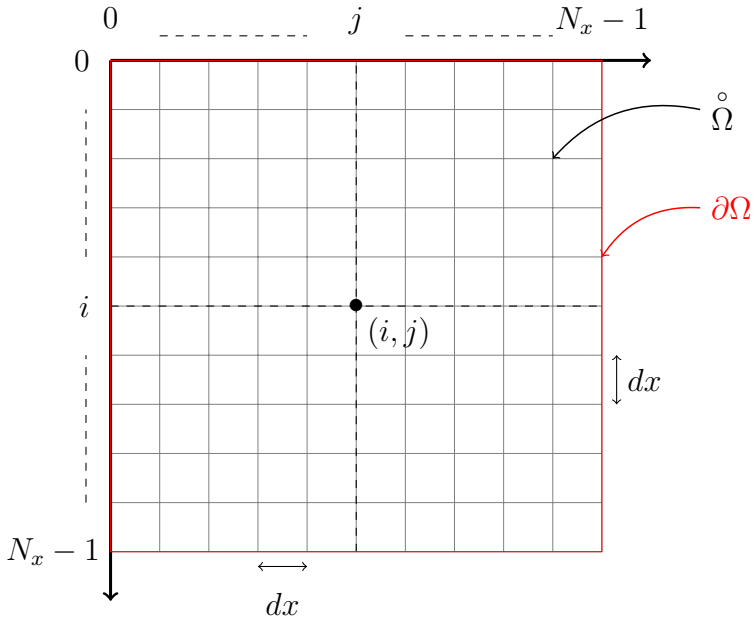

Figure 1: The 2D domain Ω

Indexing:

We use two notations to refer to a point in Ω: x_k or (x_i, y_i), where $x_i = idx$, $y_j = jdx$ and $k = iN_x + j$, for $k = 0 \dots (N_x^2 - 1)$ and $i, j = 0 \dots (N_x - 1)$. The approximation of the solution is denoted u_k, and we have:

$$u_k = u_{approx}(x_i, y_j) = u_{approx}(x_k).$$

By using Taylor expansion, we obtain for the second derivatives:

$$\partial^2_{x^2} u(x,y) = \frac{u(x+dx,y) - 2u(x,y) + u(x-dx,y)}{dx^2} + O(dx^2)$$

$$\partial^2_{y^2} u(x,y) = \frac{u(x,y+dx) - 2u(x,y) + u(x,y-dx)}{dx^2} + O(dx^2)$$

By applying these formulae at the point $x_k = (x_i, y_j)$, we obtain:

$$\partial^2_{x^2} u(x_i,y_j) = \frac{u(x_{i+1},y_j) - 2u(x_i,y_j) + u(x_{i-1},y_j)}{dx^2} + O(dx^2)$$

$$\partial^2_{y^2} u(x_i,y_j) = \frac{u(x_i,y_{j+1}) - 2u(x_i,y_j) + u(x_i,y_{j-1})}{dx^2} + O(dx^2)$$

Using the "k" notation, these approximations yield:

$$\partial^2_{x^2} u(x_k) \simeq \frac{u_{k+1} - 2u_k + u_{k-1}}{dx^2}$$

$$\partial^2_{y^2} u(x_k) \simeq \frac{u_{k+N_x} - 2u_k + u_{k-N_x}}{dx^2}$$

Finally, the approximation of the Laplacian of u at the point $x_k = (x_i, y_j)$ is:

$$\Delta u(x_i,y_j) \simeq \frac{u_{i,j-1} + u_{i-1,j} - 4u_{i,j} + u_{i+1,j} + u_{i,j+1}}{dx^2} \tag{1}$$

or

$$\Delta u(x_k) \simeq \frac{u_{k-N_x} + u_{k-1} - 4u_k + u_{k+1} + u_{k+N_x}}{dx^2}.$$

On the boundary:

Our approximation of the Laplacian Δu can not be calculated on the boundary, i.e. when $x_k \in \partial\Omega$, because we need points that are not in Ω. We use the Neumann homogeneous boundary condition : $\frac{\partial u}{\partial n} = 0$, where

$$\begin{pmatrix} n_x \\ n_y \end{pmatrix}$$

is the outer-pointing normal, in order to approximate the value of u at these non-existent points.

We denote $\partial\Omega = \Gamma_1 \cup \Gamma_2 \cup \Gamma_3 \cup \Gamma_4$, where Γ_1 is the right edge, Γ_2 is the upper edge, Γ_3 is the left edge and Γ_4 is the bottom edge. In addition, on $\Gamma_1 \to n = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$, on $\Gamma_2 \to n = \begin{pmatrix} -1 \\ 0 \end{pmatrix}$, on $\Gamma_3 \to n = \begin{pmatrix} 0 \\ -1 \end{pmatrix}$, and on $\Gamma_4 \to n = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$. This condition can be rewritten as:

$$\partial_x u \times n_x + \partial_y u \times n_y = 0.$$

On Γ_1, $j = N_x - 1$ and $i \in \{0, \dots, N_x - 1\}$, and we have $\partial_y u = 0$. If we use the formula (1), we obtain

$$\Delta u(x_i, y_{N_x-1}) \simeq \frac{u_{i,N_x-2} + u_{i-1,N_x-1} - 4u_{i,N_x-1} + u_{i+1,N_x-1} + u_{i,N_x}}{dx^2}$$

And we can use the limit condition to approximate u_{i,N_x}. A second order, three point, centered finite differences scheme yields:

$$0 = \partial_y u(x_i, y_{N_x-1}) = \frac{u_{i,N_x} - u_{i,N_x-2}}{2dx} \implies u_{i,N_x} = u_{i,N_x-2}$$

We use the same method in order to approximate on Γ_2, Γ_3 and Γ_4. We obtain:

- for $j \in \{0, \dots, N_x - 1\}$, $u_{-1,j} = u_{1,j}$ and $u_{N_x,j} = u_{N_x-2,j}$
- for $i \in \{0, \dots, N_x - 1\}$, $u_{i,-1} = u_{i,1}$

Consequently, for each $k \in \{0, \dots, N_x^2 - 1\}$ we have an approximation of $\Delta u(x_k)$, which is an equation depending on u_k (with $k \in \{0, \dots, N_x^2 - 1\}$). This system can be rewritten as $\Delta U \simeq \Delta_{approx} U$, where Δ_{approx} is the matrix described in 2.2 and

$$U = \begin{pmatrix} u_0 \\ u_2 \\ \vdots \\ u_{N_x^2-1} \end{pmatrix}.$$

2 Implementation

2.1 Input and output

Input:

To represent the ventricular tissue numerically, we use a two dimensional domain (a square), the length of the side being chosen by the user (we denoted the length of the side by N and set $N = 100$). We used multiple initial conditions (for the variables u, v, w and s), including ones suggested in the paper: *Minimal model for human ventricular action potentials in tissue*; we chose to stick with these conditions for our implementation of the minimal model. Initially, we constructed a model without the Laplacian in the first continuous equation, and used the following initial conditions: $u = 0.31$, $v = 1$, $w = 1$ and $s = 0$.

Each side is divided in 100 subdivisions and we used 2000 time steps. The maximal time (T_{max}) was set at 400. Finally, we wrote down all the different numerical values of the given parameters in the aforementioned paper in a `.yml` file, and use them as an input parameter.

Output:

The output of our code is a number of `.pdf` files. Each file contains a plot, representing the state of the system at a certain time. We save a picture of the plot in a `.pdf` file every 2 milliseconds. The number of `.pdf` files is determined by the maximal time T_{max}. For example, if $T_{max} = 400$ (milliseconds), we obtain 200 `.pdf` files, therefore 200 images of the system. The main reason behind this is so that we can create a `.gif` simulation (which would represent some sort of a time lapse) using the images.

2.2 Algorithms

To do the times resolution we only had time to implement the Euler explicit method. This method is based on the Taylor expansion of the function y:

$$y(t + dt) = y(t) + dt\partial_t y(t) + O(dt^2)$$
$$\implies \partial_t y(t) \simeq \frac{y(t + dt) - y(t)}{dt}$$

In we take $\partial_t y(t) = f(t, y)$ in consideration this leads us to:

$$y(t + dt) = y(t) + dtf(t, y)$$

To simplify the matrix construction we have used the Kronecker product, denoted by $\otimes$, which is an operation on two matrices resulting in a block matrix:

$$A \otimes B = \begin{pmatrix} a_{11}B & \cdots & a_{1j}B \\ \vdots & \ddots & \vdots \\ a_{i1}B & \cdots & a_{ij}B \end{pmatrix}$$

Using this algorithm, we can construct our matrix for the discrete Laplacian:

$$\Delta_{approx} = I \otimes A + (A + 4I) \otimes I = \left(\begin{array}{c|c|c|c} A & 2I & 0 & 0 \\ \hline I & A & I & 0 \\ \hline 0 & I & A & I \\ \hline 0 & 0 & 2I & A \end{array}\right)$$

$$\text{where } A = \begin{pmatrix} -4 & 2 & 0 & \cdots & 0 \\ 1 & \ddots & 1 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & 1 & \ddots & 1 \\ 0 & \cdots & 0 & 2 & -4 \end{pmatrix}$$

2.3 Modules

We used `Python` 's main scientific and computational modules: `numpy` and `scipy`. In particular, we used the following functions that facilitated our work (we use the prefix `np` instead of `numpy`):

- `np.array`: creates an array.
- `np.linspace`: returns evenly spaced numbers over a specified interval.
- `np.meshgrid`: returns coordinate matrices from coordinate vectors.
- `scipy.sparse.linalg.spsolve`: solves the sparse linear system $Ax = b$, where b may be a vector or a matrix.

- `np.reshape`: gives a new shape to an array without changing its data.
- `scipy.integrate.odeint`: solves a system of ODE's.

As `numpy` supports a much greater variety of numerical types than `Python` does, we used double precision float data-type for the elements of the arrays.

We used the `matplotlib` module for plotting our simulations. We used the following functions:

- `pcolormesh`: creates a pseudocolor plot of a 2D array.
- `title, colorbar, savefig, clf`: generic functions that set the title, the colorbar, save the figure and clear the figure, respectively.

We also used `argparse` and `yaml` for writing down the parameters of the different models.

2.4 Our work

To model the problem we needed a few functions:

- `heaviside` (the Heaviside function)
- `vinf`, `taumv`, `taumw`, `tauso`, `taus`, `tauo`, `winf`, `Jfi`, `Jso`, `Jsi` (representing different variables used in the equations)
- `Lap_neumann_2d` (an implementation of the Laplacian discretization with Neumann boundary conditions)
- `func_sys`

They can be found in the `function.py` file. Most of them are self explanatory but we need to clarify the last two. For `Lap_neumann_2d`, you may refer to 2.2. Concerning the last function, it is the ordinary part of the system used in the Euler method.

Every 2 milliseconds we use `pcolormesh` to plot the solution at a given time and save the figure in a `.pdf` file with the `savefig` function. Afterwards, thanks to the `convert` function from the `imagemagick` packet we are able to animate our solution.

We added two different pulsations, a band starting from the left edge as well as a square in the middle of the square domain.

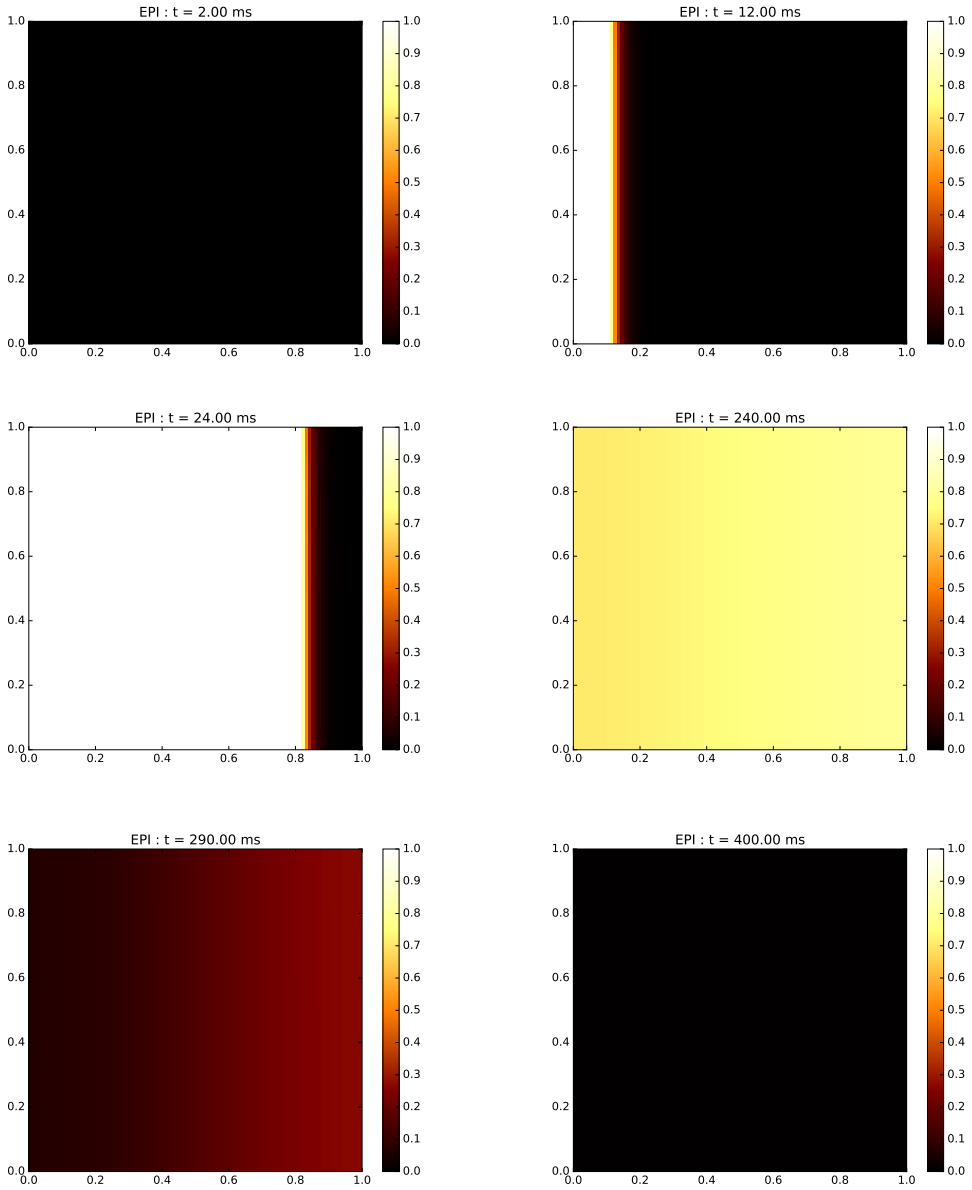

Figure 2: Minimal model with EPI parameters

2.5 Perspectives

Because of the short time frame, a few objectives remain for us to do in the future. An interesting perspective would be a theoretical study of the stability, truncation error and the order of convergence of the method. This would allow for a more complete understanding of the theoretical nature of the model, as well as its reliability and precision.

The simulations were done on a two dimensional square domain, for the sake of simplification. An adaptation of the model for a three dimensional domain would allow an even more realistic representation of the natural phenomenon.

Finally, we could use an improved method for discretizing the continuous system called the Rush Larsen solver (at a K order), initially for a two dimensional domain.

3 Logistics

3.1 Contributions

The project was for the most of its part a collective work done during TD courses by the majority of the team. The `Python` code was however mainly written by Thibaut. As for Louis, who joined the team late in the semester, understanding of the problem and the code was facilitated by help from the others, as teamwork helped him understand the majority of the work done beforehand.

Date	Work accomplished	Time	Participants
20/09	-Setup of a `GitHub` repository -Reading/Comprehension of the minimal model -Discretization of the model using the finite differences method with the Euler explicit method	3h	4
27/09	-Implementation and plotting of the minimal model functions without the Laplacian	3h	4
04/10	-Discretization of the minimal model taking boundary conditions into consideration -Test with the fonction $u = \cos(k_1\pi x)\cos(k_2\pi y)$: error not small enough $\Longrightarrow$ wrong matrix	3h	4
11/10	-Correction of the matrix -Correction of the second member -Test with the fonction $u = \cos(k_1\pi x)\cos(k_2\pi y)$: error ok decrease at a $O(h^2)$ rate	3h	4
18/10	-Print colormesh : does not show the expected result (difference of potential does not go back to zero)	3h	4
25/10	-The previous error was due to the scale of the colormap -Test with different sets of parameters (first impulse EPI : 0.5, PB : 1)	3h	5
08/11	-Sketch of the paper -Test with a different impulse : rectangle -Try new impulse : rectangle + square latter	2h	5
15/11	-Sketch of the paper	2h	5
22/11	-Writing of the paper	2h	5
29/11	-Writing of the paper	3h	5

3.2 Material

The programming language used for simulations was `Python` (version `3.5`), including modules such as `numpy, scipy` and `matplotlib`. We used `emacs` as our main text editor, and the work was done on a `Debian Linux` distribution.

Concerning organisation of the code and version control, we used `GitHub`, which allowed us to easily synchronize work done outside of class (here is a link to our repository: https://github.com/tguegan/HeartAttack).

We used LaTeX to write this paper. The team wrote simultaneously by using the `Overleaf .tex` editor.

3.3 Difficulties

Precisely 80% of us have already written code in a programming language, 60% of whom in `Python`, including libraries such as `numpy, scipy` and `matplotlib`. This has allowed us to progress early in the semester and spend more time on the theoretical issues, such as the discretization.

For the remaining 20% who haven't written code before, learning `Python` late in the semester meant a lack of time to spend understanding the nature of the problem.

3.4 Enhancements

At the beginning of the semester, we concentrated a significant part of our efforts on discretizing the Laplacian in the two dimensional domain, using the finite differences method. After multiple simulations, we found out that the discretization we had used before had been false. The reason for this was a misunderstanding of how to implement the Neumann boundary conditions in the discretization of the problem, more particularly in the matrix that results from said discretization.

Another point where we could've improved on was the theoretical understanding of the scheme. We weren't certain of our choice mainly because of the fact that we didn't know much of it's consistence or stability, therefore even less of its convergence.

4 Conclusion

We've amassed a big amount of knowledge during this project. Having been introduced to the finite differences method in the *Approximation des EDP 1* course, we had the opportunity to complete our understanding of this method, and see how it may be applied to a non-standard equation (contrary to the heat equation or the wave equation, for example). In addition, we have significantly improved our `Python` programming skills, and saw how it is used in modelling and simulations of physical models. The teamwork was

excellent, which made this project, in addition to our interest in the subject, highly enjoyable.