Finite Difference Method for Simulating Tissue Wave Propagation

Akwasi Darkwah Akwaboah*

***Abstract*— Action potential propagation in a tissue are often governed by known differential equations. Such equations often are formulated as diffusion (classical Poisson) style over 1-, 2- or 3- dimensional space with respect to time and can be solved by several numerical methods. In this report, a typical tissue equation governing a 2D 3cm × 3cm tissue is solved over several time steps using an explicit finite difference method. The effect of grid spacing and the time step size on the computational stability is investigated.**

***Index Terms*— Computational Biology, wave propagation, finite difference**

I. Introduction

Numerical modeling has provided insights into tissue excitability. An instance of this is the solution of the cardiac bidomain and monodomain equations for simulating action potentials across the various cell membranes. Generally, wave propagation in tissues are often formulated as elliptical partial differential equations (classical Poisson equation). Several numerical techniques exist for solving such equations. These include finite difference methods, finite element methods, finite volume methods and operator splitting. Finite difference methods (FDM) are the simplest to implement, but however may be characterized by significant approximation errors resulting from inadequate temporal and spatial discretization.
FDM is based on Taylor series approximation. Typical first-order approximation with sufficiently small discretization can yield computational stability. Finite Element Methods often guarantee more accurate solution but are relatively abstruse as complex trial functions are required for spatial extrapolations within the various elements. FDM may be formulated as explicit, semi-explicit and implicit approaches with consideration for their accompanying requirements. Explicit methods, i.e. Forward Euler methods though ensures instantaneous variable update computation without dependence on future values. This method is simple however often inaccurate due to their small-time-step requirement needed for computational stability. Implicit methods, i.e. backward Euler methods on the contrary are absolutely stable, as they depend on future values. They are however computationally expensive as future derivatives must be computed often analytically. Semi-implicit methods are hybrids that combines the simplicity of forward Euler (explicit) methods and the computational stability of backward Euler (implicit) methods[1].
In this report, I employ an explicit finite difference method to simulate time course of wave propagation of in a 2D 3cm×3cm tissue shown in figure 1. This computational mesh grid is governed by an elliptical partial differential equation shown in equation 1 below;

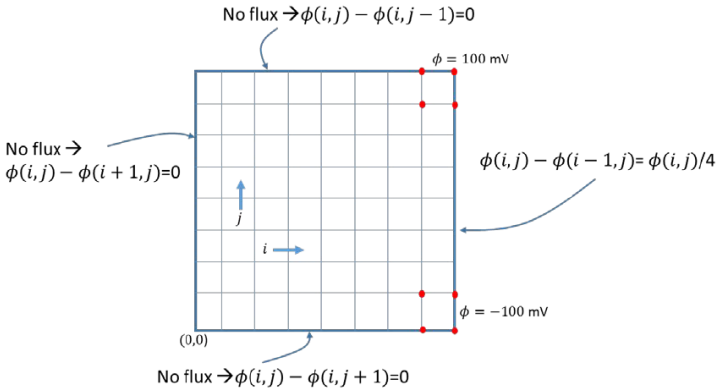

Figure 1. Computational mesh grid - tissue discretization with boundary conditions.

$$\nabla^2\Phi = \frac{\Phi}{R} + C_m\frac{\delta\Phi}{\delta t} \tag{1}$$

Where the Laplacian,

$$\nabla^2\Phi = \frac{\delta^2\Phi}{\delta x^2} + \frac{\delta^2\Phi}{\delta y^2} \tag{2}$$

By explicit formulation,

$$\frac{\delta^2\Phi}{\delta x^2} = \frac{\Phi(i+1,j,k) - 2\Phi(i,j,k) + \Phi(i-1,j,k)}{(\Delta x)^2} \tag{3}$$

$$\frac{\delta^2\Phi}{\delta y^2} = \frac{\Phi(i,j+1,k) - 2\Phi(i,j,k) + \Phi(i,j-1,k)}{(\Delta y)^2} \tag{4}$$

*Akwasi Darkwah Akwaboah is with the Electronics Engineering Department, Norfolk State University

Substituting equations 3 and 4 into 2 yields equation 5

$$\nabla^2\Phi = \frac{\Phi(i+1,j,k) - 2\Phi(i,j,k) + \Phi(i-1,j,k)}{(\Delta x)^2} + \frac{\Phi(i,j+1,k) - 2\Phi(i,j,k) + \Phi(i,j-1,k)}{(\Delta y)^2} \quad (5)$$

Setting $\Delta x = \Delta y = \Delta h$

$$\nabla^2\Phi = \frac{\Phi(i+1,j,k) - 4\Phi(i,j,k) + \Phi(i,j+1,k) + \Phi(i-1,j,k) + \Phi(i,j-1,k)}{(\Delta h)^2} \quad (6)$$

Also,

$$\frac{\delta\Phi}{\delta t} = \frac{\Phi(i,j,k+1) - \Phi(i,j,k)}{\delta t} \quad (7)$$

Substituting equations 6 and 7 into equation 1

$$\frac{\Phi(i+1,j,k) - 4\Phi(i,j,k) + \Phi(i,j+1,k) + \Phi(i-1,j,k) + \Phi(i,j-1,k)}{(\Delta h)^2} = \frac{\Phi(i,j,k)}{R} + \frac{\Phi(i,j,k+1) - \Phi(i,j,k)}{\Delta t} \quad (8)$$

$$\frac{R\Delta t}{C_m(\Delta h)^2}[\Phi(i+1,j,k) - 4\Phi(i,j,k) + \Phi(i,j+1,k) + \Phi(i-1,j,k) + \Phi(i,j-1,k)] = C_m\Delta t\Phi(i,j,k) + C_m\Phi(i,j,k+1) - \Phi(i,j,k) \quad (9)$$

Setting $r = \frac{R\Delta t}{C_m(\Delta h)^2}$ and given a tissue nodal resistance per unit area $R = 1\Omega m^{-2}$ and nodal capacitance per unit area , $C_m = 1Fm^{-2}$ renders $r = \frac{\Delta t}{(\Delta h)^2}$. Thus, equation 9 reduces to equation 10 below

$$\boldsymbol{\Phi(i,j,k+1) = r[\Phi(i+1,j,k) + \Phi(i-1,j,k) + \Phi(i,j+1,k) + \Phi(i,j-1,k)] - (4r + \Delta t - 1)\Phi(i,j,k)} \quad (10)$$

The reader must note that i, j, k are used as indices for horizontal (x-axis), vertical (y-axis) and time (z-axis) respectively. The significance of equation 10 is the computation of future potentials $\boldsymbol{\Phi(k+1)}$ as a function of present potentials $\boldsymbol{\Phi(k)}$. No flux boundary conditions were assumed for the left, bottom and top edges, whereas a boundary flux of $\frac{\delta\Phi}{\delta n} = \frac{\Phi}{4}$ is applied to the right edge.

II. Methods (Implementation Strategy)

The wave propagation in the 2D tissue slab governed by equations 1 and 10 were simulated using python code (attached in the appendix and available via this GitHub link). NumPy and Matplotlib modules were used. Mesh grid points (nodes) were implemented as array elements. A 3-dimensional array comprising the x-axis (columns – *i*), y-axis (rows – *j*) and time (depth – *k*). An anode potential of 100mV and cathode potential of -100mV is applied to the upper right corner and bottom right corner respectively as shown in figure 1. These electrode potentials are sustained through the time course.

To determine the suitable grid spacing for numerical stability, the Courant-Friedrichs-Lewy (CFL) condition[2][3] is employed in the explicit computation. This condition is presented in equation 11 below;

$$\Delta t \leq \frac{\chi C_m(\Delta h)^2}{2(\sigma_l + \sigma_t)} \quad (11)$$

Where χ is the surface to volume ratio, which is assumed to be unity and consequently $r \equiv \frac{\chi}{2} = \frac{1}{2}$, Δt represents the time step. σ_l and σ_t are the longitudinal and transverse conductivity both inversely proportional to resistance per unit area, R, i.e. $\sigma_l = \sigma_t = \frac{k}{R}$. Where k is unity, $\sigma_l = \sigma_t = 1\Omega m$. As such equation 11 reduces to equation 12 below;

$$\Delta t \leq \frac{r(\Delta h)^2}{2} = \frac{(\Delta h)^2}{4} \quad (12)$$

As such, an optimum grid spacing, Δh = 0.2cm requires a time step of 0.1ms.

III. Results: Evolution of Signal over Time

A base (control) grid spacing of $\Delta h = 0.2$cm and accompanying maximum time step for computational stability, Δt = 0.1ms computed using the CFL condition shown in equation 11. Wave propagation over 25 time steps are simulated. Selected sequential plots presenting the evolution of the potential with respect to time are shown in this section

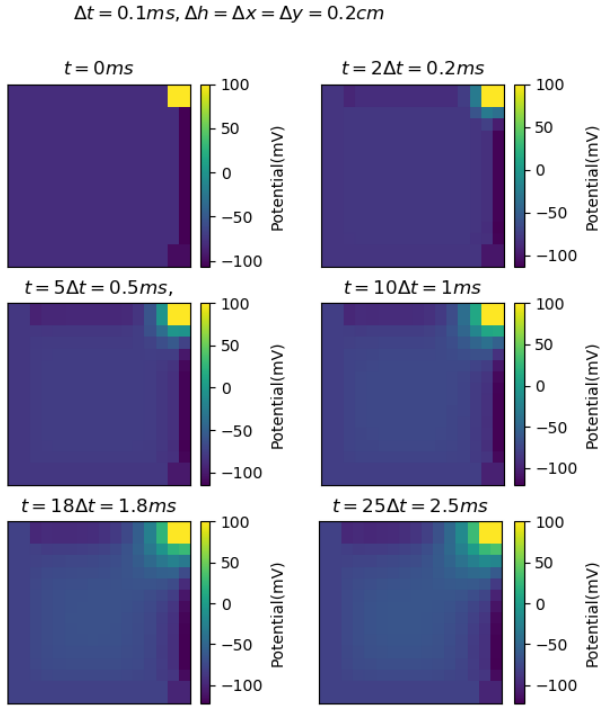

Figure 2. Control: Evolution of potential/ wave propagation over the time course: Δh = 0.2cm and Δt = 0.1ms. Numerically Stable

IV. Grid Spacing, Time Steps & Computational Stability

Computational instability may arise when grid spacing is increased without compensating for the time step size constrained by the CFL condition. However, an increased grid spacing without increasing the time step will yield numerical stability as the maximum time step for numerical stability is proportional to the chosen grid spacing, thus the maximum time step will be extended (increased to 0.225ms). To demonstrate this, *Protocol 1* – the control grid spacing is increased by 150% (i.e. $\Delta h = 1.5 \times 0.2\text{cm} = 0.3\text{cm}$), while the control time step, $\Delta t = 0.1\text{ms}$ are used. This yielded numerical stable outcome as expected. Plots for this is shown below.

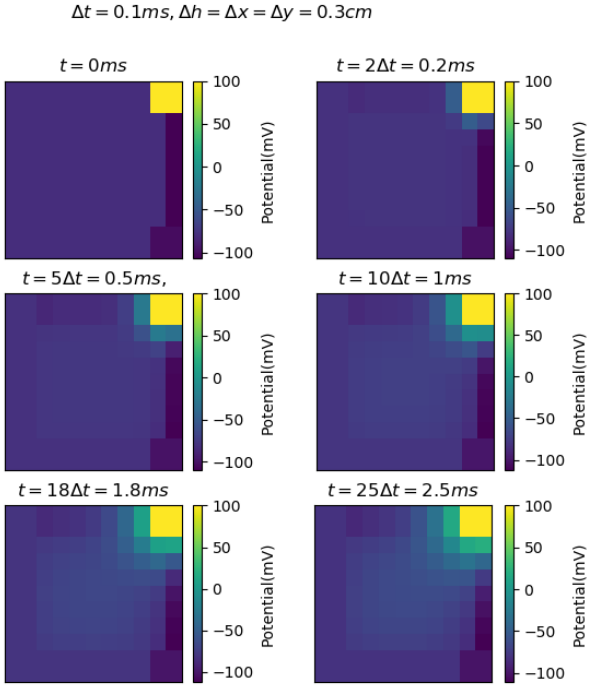

Figure 3. Protocol 1: Wave propagation in fewer grid nodes i.e. larger grid spacing, $\Delta h = 0.3\text{cm}$ and $\Delta t = 0.1\text{ms}$. Numerically stable

To demonstrate computational instability when the CFL condition is defied, *Protocol 2* – (intuitively opposite to *Protocol 1*) the control time step is increased 150% while the control grid spacing is kept the same (i.e. $h = 0.2\text{cm}$). Resultant plots are shown in figure 4. A summary table comparing the various conditions used in the control and the 2 protocols is presented in table 1 below. Optimal time step and grid spacing corresponding to the control modifications are also included. The optimum value of r , a criterion for computational stability from equation 12 can be derived as; $r = \frac{2\Delta t}{(\Delta h)^2} = 0.5$

TABLE 1. SUMMARY: EFFECT OF TIME STEPS AND GRID SPACING ON NUMERICAL STABILITY

	Δt(ms)	**Δh**(cm)	**r**	**Numerically Stable?**
Control	0.1*	0.2*	0.50	Yes
Protocol 1	0.1 (0.225*)	0.3	0.22	Yes
Protocol 2	0.15	0.2 (0.7746*)	0.75	No (CFL condition not met i.e. r > 0.5)

* - optimal values for Δt or Δh for the other parameter defined

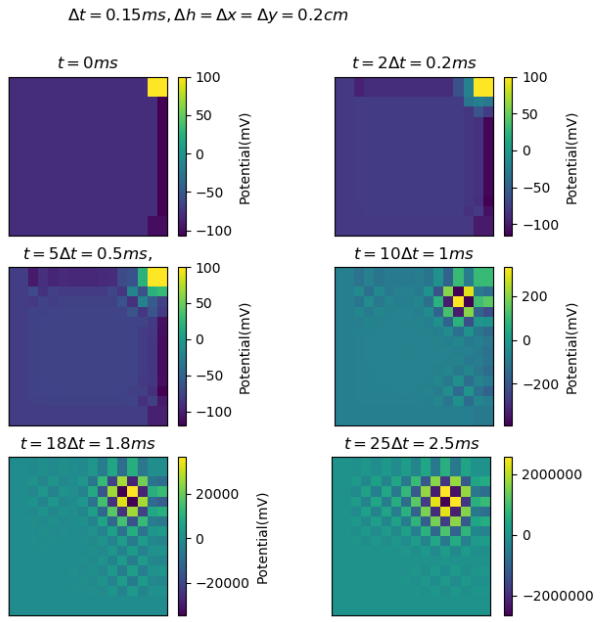

Figure 4. Protocol 2: wave propagation with numerical instability due to the CFL criteria not met. $\Delta h = 0.2\text{cm}$ and $\Delta t = 0.15\text{ms}$

Table 1 is intended to help the reader to briefly appreciate the role of the grid spacing and time step size in ensuring numerical stability for forward Euler FDM. The largest time step that maintains stability for protocol 1 from table 1 is larger than that of the control. This indicative of direct proportionality between grid spacing and time step size shown in equations 11 and 12.

V. Impulse Propagation: Unsustained Stimulation

Up until now, all simulations were run with sustained electrode potential over all the time steps. This section retires this work with how an impulse (i.e. stimulation only at t = 0) is propagated over the tissue over time using protocol 1 optimum conditions. Plots for this is shown in figure 5.

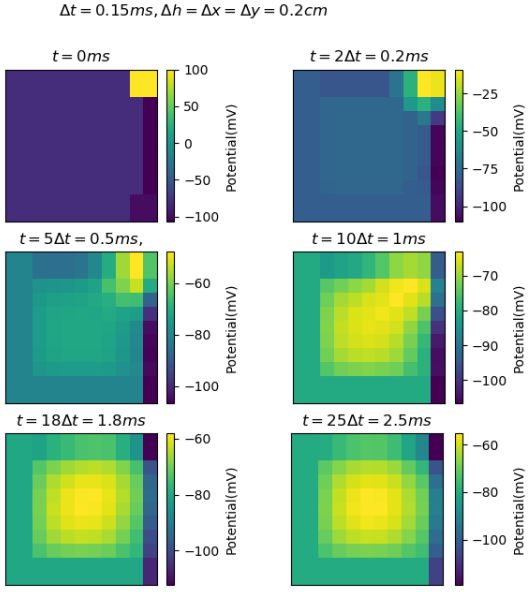

Figure 5. Impulse propagation, electrode potential only applied at time, t = 0

VI. Limitations

Tissue parameters, capacitance per unit area, C_m, resistance per unit area, R and the surface to volume ratio χ, were assumed to be unity. However, practical values may differ, in which optimum parameters can be recalculated.

VII. Conclusion

In this report, I present an explicit finite difference method for simulating tissue wave propagation. The role of grid spacing and time steps on the numerical stability is investigated.

VIII. References

[1] E. J. Vigmond, R. Weber dos Santos, A. J. Prassl, M. Deo, and G. Plank, "Solvers for the cardiac bidomain equations," *Progress in Biophysics and Molecular Biology*, vol. 96, no. 1–3. pp. 3–18, Jan-2008.

[2] R. Courant, K. Friedrichs, H. L.-M. annalen, and undefined 1928, "Über die partiellen Differenzengleichungen der mathematischen Physik," *Springer*.

[3] R. Weber, D. Santos, G. Plank, S. Bauer, and E. J. Vigmond, "Preconditioning Techniques for the Bidomain Equations."

IX. Appendix

A. *Python Implementation*

(alternatively available via Github link)

```
# Written by Akwasi Darkwah Akwaboah
# Description: Finite Difference Method (FDM) to simulate tissue potential
# Date: 11/26/2019

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.colors as colors

# Step 1: Initialize the grid numbers for the solution regions
delta_t = 0.01
#delta_t = 0.01 * 1.50

delta_h = 0.2
#delta_h = 0.2 * 1.5

time_limit = 27
t = np.arange(0.0, time_limit * delta_t, delta_t)
x = np.arange(0.0, 3 + delta_h, delta_h)
y = np.arange(0.0, 3 + delta_h, delta_h)

R = 1
Cm = 1

r = ((R / Cm) * (delta_t)) / (delta_h ** 2)
print(r)

#mesh grid node initialization
V = np.ones((len(x), len(y), time_limit), dtype='float64') * -80
#initial resting potential
Anode = 100
Cathode = -100

# Initialize boundary conditions for each time iteration
# Electrode Stimulation
V[-2:, -2:,:] = Anode  # top right corner
V[:2, -2:, :] = Cathode  # bottom left corner

# #impulse
# V[-2:, -2:, 0] = Anode  # top right corner
# V[:2, -2:, 0] = Cathode  # bottom left corner

V[:, 0, :] = V[:, 1, :]  # left - no flux
V[len(x) - 1, :, :] = V[len(x) - 2, :, :]  # top - no flux
V[0, :, :] = V[1, :, :]  # bottom - no flux
V[:, len(x) - 1, :] = (4 / 3) * V[:, len(x) - 2, :]  # right flux

# Step 2: approximate derivative | finite difference - written by Akwasi
for k in np.arange(0, len(t) - 1):  # time
    for j in np.arange(1, len(y) - 1):  # do not include boundaries
        for i in np.arange(1, len(x) - 1):
            V[i, j, k + 1] = r * (V[j, i + 1, k] + V[j, i - 1, k] + V[j + 1, i, k] + V[j - 1, i, k]) - (
                        (4 * r / R) + delta_t - 1) * V[j, i, k]
    # ensure boundary conditions are maintained through the time course
    V[:, len(x) - 1, :] = (4 / 3) * V[:, len(x) - 2, :]  # right
    V[1, :, :] = V[0, :, :]  # bottom
    V[:, 1, :] = V[:, 0, :]  # left
    V[len(x) - 1, :, :] = V[len(x) - 2, :, :]  # top

    V[-2:, -2:, :] = Anode  # top right corner
    V[:2, -2:, :] = Cathode  # bottom left

    # # impulse propagation
    # V[-2:, -2:, 0] = Anode  # top right corner
    # V[:2, -2:, 0] = Cathode  # bottom left corner

# V = np.flip(V, axis = 0)
for time in np.arange(0, 1, 5):
    fig = plt.figure(figsize=(15, 15))
    fig.suptitle('$\Delta t = 0.1ms, \Delta h = \Delta x = \Delta y = 0.2cm$')
    ax = fig.add_subplot(321)
    im = plt.imshow(V[:, :, time], origin='lower')
    plt.title('$t = 0ms$')
    im.axes.get_xaxis().set_visible(False)
    im.axes.get_yaxis().set_visible(False)
    fig.colorbar(im, label='Potential(mV)', fraction=0.046, pad=0.04)
    #fig.tight_layout()

    ax = fig.add_subplot(322)
    im = plt.imshow(V[:, :, 2], origin='lower')
    plt.title('$t = 2\Delta t = 0.2ms$')
    im.axes.get_xaxis().set_visible(False)
    im.axes.get_yaxis().set_visible(False)
    fig.colorbar(im, label='Potential(mV)', fraction=0.046, pad=0.04)

    ax = fig.add_subplot(323)
    im = plt.imshow(V[:, :, time + 5], origin='lower')
    plt.title('$t = 5\Delta t = 0.5ms, $')
    im.axes.get_xaxis().set_visible(False)
    im.axes.get_yaxis().set_visible(False)
    fig.colorbar(im, label='Potential(mV)', fraction=0.046, pad=0.04)
    #fig.tight_layout()

    ax = fig.add_subplot(324)
    im = plt.imshow(V[:, :, time + 10], origin='lower')
    plt.title('$t = 10\Delta t = 1ms$')
    im.axes.get_xaxis().set_visible(False)
    im.axes.get_yaxis().set_visible(False)
    fig.colorbar(im, label='Potential(mV)', fraction=0.046, pad=0.04)
    #fig.tight_layout()

    ax = fig.add_subplot(325)
    im = plt.imshow(V[:, :, time + 18], origin='lower')
    plt.title('$t = 18\Delta t = 1.8ms$')
    im.axes.get_xaxis().set_visible(False)
    im.axes.get_yaxis().set_visible(False)
    fig.colorbar(im, label='Potential(mV)', fraction=0.046, pad=0.04)
    #fig.tight_layout()

    ax = fig.add_subplot(326)
    im = plt.imshow(V[:, :, time + 25], origin='lower')
    plt.title('$t = 25\Delta t = 2.5ms$')
    im.axes.get_xaxis().set_visible(False)
    im.axes.get_yaxis().set_visible(False)
    fig.colorbar(im, label='Potential(mV)', fraction=0.046)
    #fig.tight_layout()

plt.show()
```