Modeling of Neuron Signaling from Dendrite to Axon Hillock

BENG 221 Problem Solving Report

Sung Min Kim
David Oda
Pokman Cheng

Table of Contents

1. Introduction

The functional unit of the nervous system is a neuron, which processes and transmits signal via electrical conduction. A typical neuron consists of three main parts: dendrite, soma, and an axon (Fig 1). The axon hillock is the region of the soma from which the axon originates.

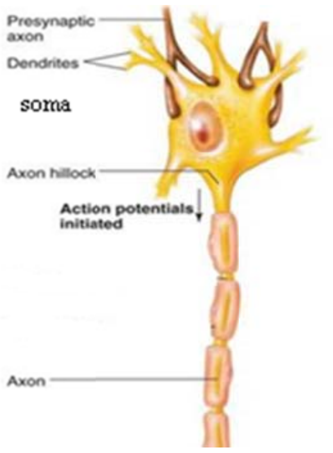


Figure 1. Structure of neuron

Neuron signaling works as follow: First, neurotransmitters from a presynaptic axon bind to ligand-gated ion channels of a postsynaptic dendrite, opening the channels. Through these channels certain ions flow in, get diffused and alter membrane potential. When the potential at the axon hillock exceeds a certain triggering threshold value, an action potential is generated and propagated to the end of axon.

Input signals can either be excitatory or inhibitory. Excitatory signals are caused by the influx of cations which raises the membrane potential (Fig. 2A), increasing the probability of an action potential generation at the axon hillock. Inhibitory signals are caused by a combination of the influx of anions and the efflux of cations which lowers membrane potential (Fig. 2B) decreasing the probability of action potential generation.

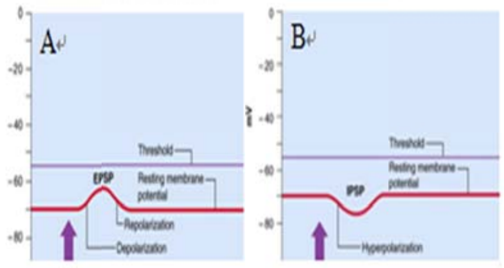


Figure 2. Excitatory and inhibitory input

The altered membrane potential caused by these input signals can be restored back to its original state by various kinds of ATP-driven pumps which are distributed throughout the cell membrane. For example, Na^+/K^+-ATPase (Fig. 3) pumps two potassium ions in and three sodium ions out of the cell to make the membrane potential more negative, compensating the effect of excitatory input signal.

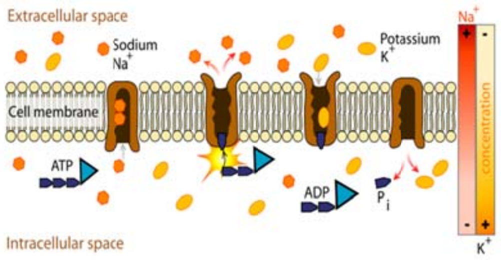


Figure 3. Na^+/K^+ pump

In this study, we modeled the membrane potential of soma that accounts for the following three dynamic properties:

1. Input impulse, both inhibitory and excitatory, caused by the opening of ion channels
2. Restoring force by the ATP-driven pumps
3. Diffusion of electrical potential in soma

2. Problem Setup

Our goal is to find the function u(*x*, *y*, *t*) which we define as the amount of potential deviation from the resting state at location (x,y) of the soma at time t. To simplify, we made the following assumptions.

2.1. Assumptions

1. The shape of the soma is square and flat enough to be treated as 2D plane (Fig. 4).

2. Axon is located at the center of the soma.

3. Restoring pumps are distributed uniformly throughout the membrane (Fig. 4-yellow).

4. Restoring force is linearly proportional to the value of u, which therefore can be expressed as $-Cu(x, y, t)$.

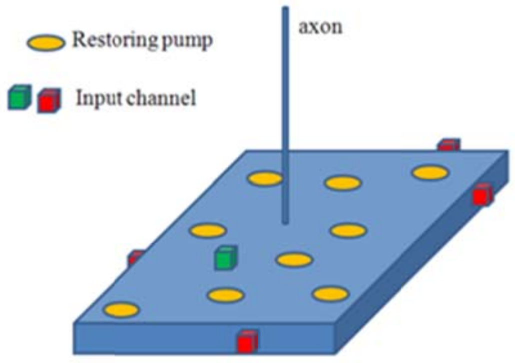


Figure 4. Simplified model of neuron

5. There is no ion flux at the membrane other than through the input channels or restoring pumps.

6. We set the maximum number of input channels to be 4 for the ease of demonstration but this number can be increased if necessary.

7. We placed input channels on the boundary of the square (Fig 4-red) but it can be placed on the middle of the square if necessary (Fig 4-green).

8. Diffusivity (D) of electrical potential inside the cell is uniform.

9. Input impulse exist only at t = 0.

2.2. Differential Equation

Putting all these conditions into an equation using D and C as diffusivity and restoring force constant respectively, we get

PDE: $\frac{\partial u}{\partial t} = D\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) - Cu$

BC: $\frac{\partial u}{\partial x}(0, y, t) = 0$

$\frac{\partial u}{\partial x}(L, y, t) = 0$

$\frac{\partial u}{\partial y}(x, 0, t) = 0$

$\frac{\partial u}{\partial y}(x, L, t) = 0$

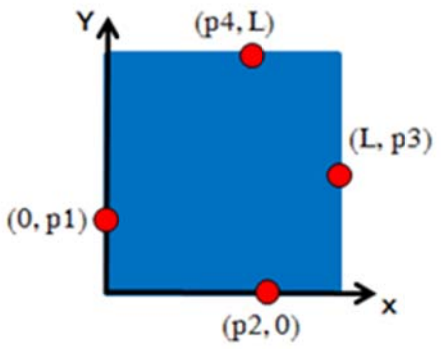


Figure 5. Top view of modeled neuron

IC: $u_0(x, y) = V_1\delta(x)\delta(y-p_1) + V_2\delta(x-p_2)\delta(y) + V_3\delta(x-L)\delta(y-p_3) + V_4\delta(x-p_4)\delta(y-L)$

Above IC is the case where there are total of four input channels, one on each side of the square soma (Fig. 5). V_n is positive for excitatory input signal and negative for inhibitory input signal.

3. Solution

3.1. Analytical Solution

$$u(x, y, t) = \left\{ \begin{array}{l} \frac{A_{0,0}}{4} + \frac{1}{2}\sum_{k=1}^{\infty}(A_{k,0}\cos(\frac{k\pi x}{L}) + A_{0,k}\cos(\frac{k\pi y}{L}))\exp(-D\left(\frac{k\pi}{L}\right)^2 t) + \\ \sum_{m=1}^{\infty}\sum_{m=1}^{\infty} A_{m,n}\cos(\frac{m\pi x}{L})\cos(\frac{n\pi y}{L})\ \exp(-D((\frac{m\pi}{L})^2 + (\frac{n\pi}{L})^2)t) \end{array} \right\} \exp(-Ct)$$

$$A_{m,n} = \frac{4}{L^2}\left\{V_1 \cos\left(\frac{n\pi p_1}{L}\right) + V_2 \cos\left(\frac{m\pi p_2}{L}\right) + V_3 \cos(m\pi)\cos\left(\frac{n\pi p_3}{L}\right) + V_4 \cos(n\pi)\cos\left(\frac{m\pi p_4}{L}\right)\right\}$$

See Appendix A for its derivation.

3.2. Numerical Solution: 2 Dimensional Forward Finite Difference Method

$$u_{x,y,t+1} = \frac{D\Delta t}{\Delta x^2}\ (u_{x+1,y,t} - 2u_{x,y,t} + u_{x-1,y,t}) + \frac{D\Delta t}{\Delta y^2}\ (u_{x,y+1,t} - 2u_{x,y,t} + u_{x,y-1,t}) - C\,\Delta t\, u_{x,y,t}$$

See Appendix B for its derivation.

3.3. Numerical Solution: MATLAB PDE Toolbox

See Appendix C for its usage.

4. Results

We used the following values for the constants in our simulation.

Table 1. Values for constants

Constant description	Variable	Value
Diffusivity of electrical potential in soma	D	$7.0*10^{-4}$ $\mu m^2/s$
Restoring force constant	C	$4.0*10^{-3}$ s^{-1}
Side length of soma	L	1.0 μm
Input signal voltage	V_n	-30 ~ +30 mV

Fig. 6A shows the potential change over time at the center of soma plotted from the analytical solution. As the number of excitatory input increases, each with the magnitude of 30 mV, the membrane potential increases. If we take the threshold to be 30 mV, an action potential is generated only when there are more than two excitatory input signals.

Fig. 6B shows the effect of inhibitory inputs, each with the magnitude of 30 mV. When there are three excitatory and one inhibitory inputs, the potential at the center does not exceed the threshold. Therefore, the presence of one inhibitory input is enough to prevent the action potential generation.

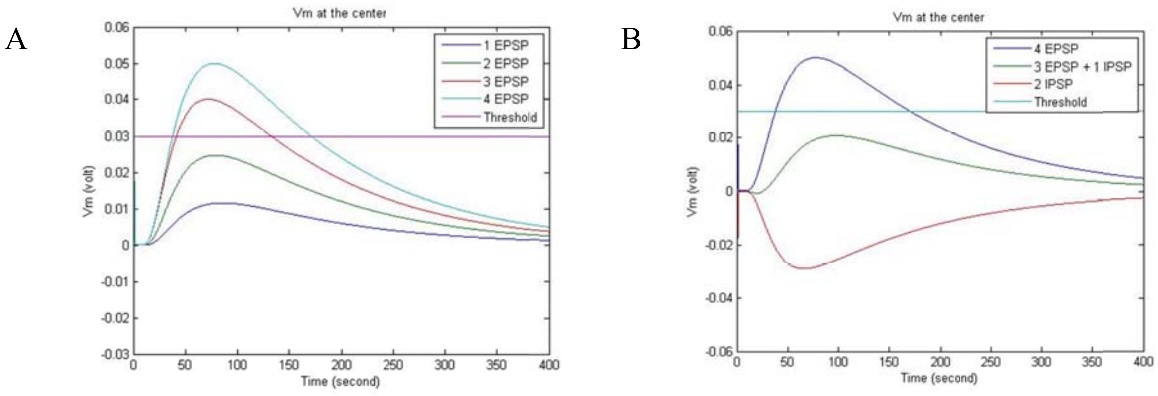


Figure 6. Analytical solution in the presence of excitatory and inhibitory input signals

Fig. 7 shows how numerical solution obtained from 2-dimensional finite difference method (Fig. 7-blue line) approaches the analytical solution (Fig. 7-green line) as $\Delta x, \Delta y$ and Δt decreases. Compared to 1-dimensional case, the 2D finite difference method solution requires larger amount of decrease in $\Delta x, \Delta y$ and Δt and thus more computation time to achieve a reasonable approximation.

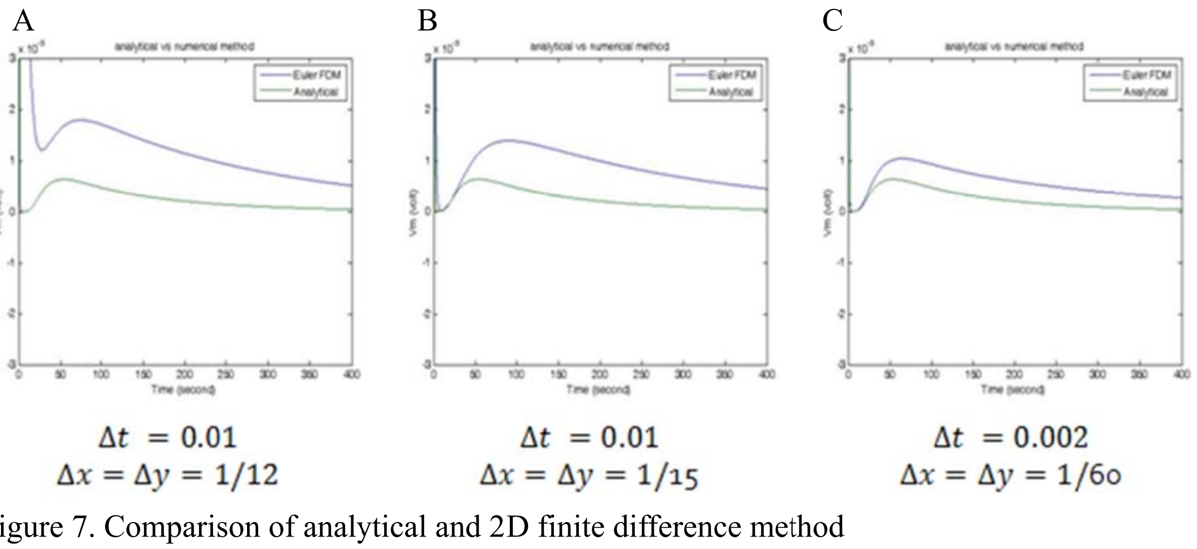


Figure 7. Comparison of analytical and 2D finite difference method

Fig. 8 compares the analytical and numerical solution in the entire region of the soma from t=0s to t=400s with two excitatory and two inhibitory inputs. We can see that the pattern of potential diffusion is very similar in three solutions. As time goes on, the initial peaks are flattened and the membrane potential eventually restores its resting state.

Fig. 9 shows the 2D potential change when the soma is of a circular shape using MATLAB PDE Toolbox.

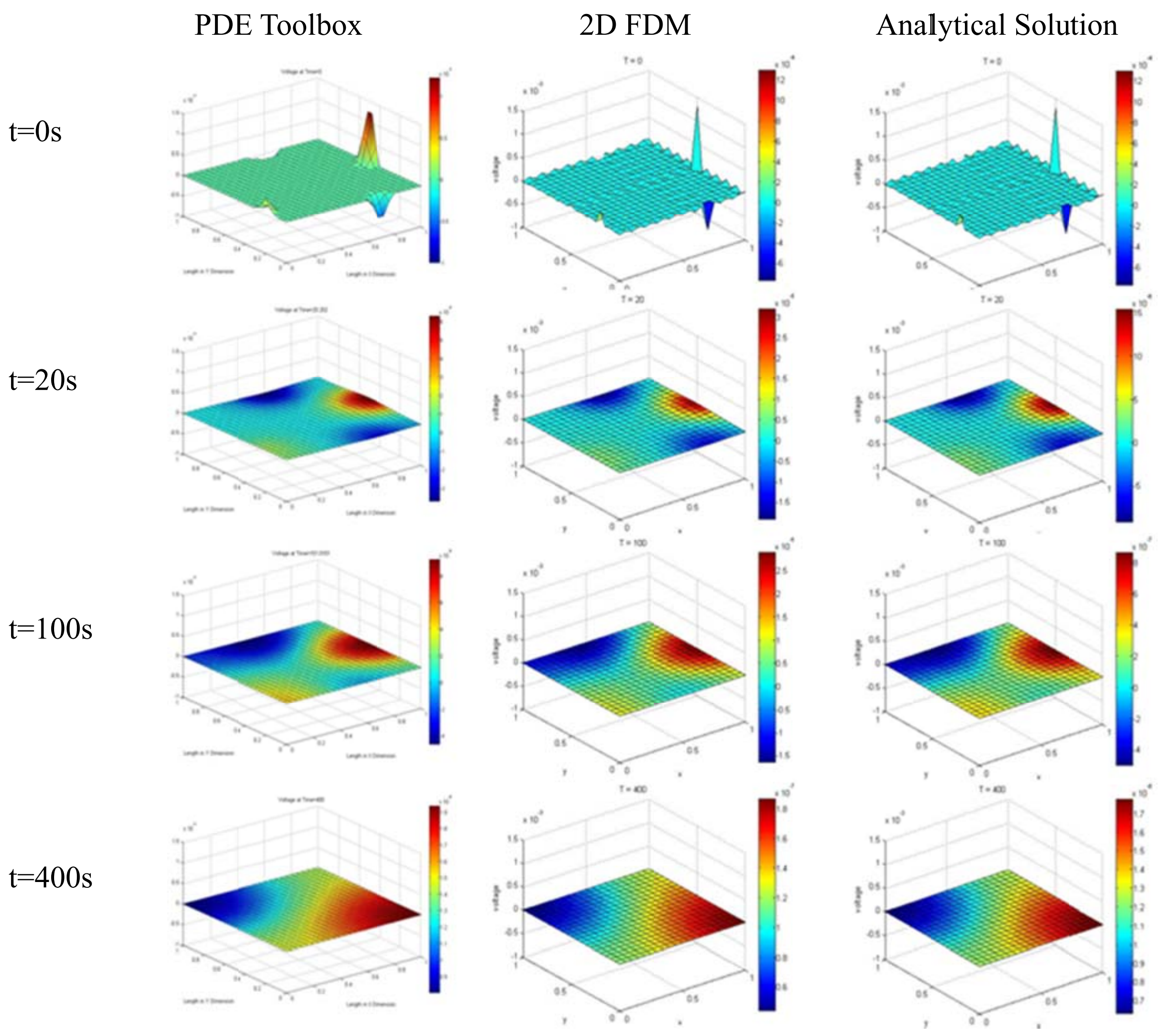


Figure 8. Comparison of analytical and numerical solutions for the entire region of soma

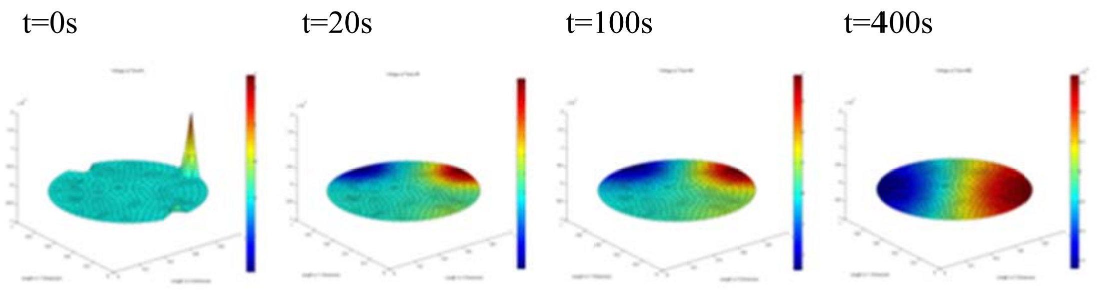


Figure 9. Potential change in a soma of circular shape

5. Conclusion

In this study, we created a model for monitoring the electrical potential in the soma of a neuron. We have shown that it successfully covers the basic features of neuron signaling, such as excitatory and inhibitory inputs as well as the restoring force caused by ATP-driven pumps ,and the diffusion of the electrical potential over the region of the soma.

Our model is scalable in that we can increase the number of inputs at any location in the soma by properly modifying our initial condition, $u_0(x,y)$. We can monitor the potential at any specific location or the entire region of the soma. Also, we can predict whether the action potential would be generated under certain conditions. In this respect, our model can serve as a basic module to quantitatively understand complex behavior of the nervous system.

The PDE toolbox is a powerful way to solve 2D PDE's very easily without having to worry about setting up correct meshes when using the finite element method. However, the PDE toolbox does have its limitations. With the PDE toolbox we are not able to modify the equation to include a restoring force. This means that over time if enough excitatory potentials were fired, we would see an accumulation of voltage that wouldn't normally occur. While the PDE toolbox does provide a great deal of simplicity, it can also lead to inaccurate solutions depending on what needs to be modeled.

6. Future Work

To build a more realistic model, there are several things to be improved. First, we limited the geometry of the soma to a flat 2D square. However in reality, the soma is more or less irregular and 3 dimensional. Second, we limited the input signal to occur all at once at t=0s. However, it would be more realistic to allow the input to be generated at any time. We also need to find a way to generalize the duration of channel opening rather than assuming it to be transient. Using our model we could also try to implement a larger network with multiple neurons interconnected.

With regard to the numerical solution, we used 2-dimensional forward finite difference method because there was no built-in MATLAB function to directly solve our equation. However, this forward iteration was conditionally stable and was computationally expensive to obtain tolerable approximation. Our future work therefore includes implementing advanced iteration algorithms, such as 2D Crank Nicholson or Alternating Direction Implicit (ADI) method.

7. References

1. Lehninger. Principles of Biochemistry. New York, 2008. p.449-453.

2. Neuron. (2012, November 15). In *Wikipedia, The Free Encyclopedia*. Retrieved 22:18, November 15, 2012, from http://en.wikipedia.org/w/index.php?title=Neuron&oldid=523123706

3. Postsynaptic potential. (2012, November 6). In *Wikipedia, The Free Encyclopedia*. Retrieved 22:19, November 15, 2012, from http://en.wikipedia.org/w/index.php?title=Postsynaptic_potential&oldid=521714565

4. Alternating direction implicit method. (2011, November 30). In Wikipedia, The Free Encyclopedia. Retrieved 23:48, November 15, 2012, from http://en.wikipedia.org/w/index.php?title=Alternating_direction_implicit_method&oldid=463329248

5. Heat equation. (2012, November 9). In Wikipedia, The Free Encyclopedia. Retrieved 23:50, November 15, 2012, from http://en.wikipedia.org/w/index.php?title=Heat_equation&oldid=522195550

6. http://humanphysiology2011.wikispaces.com/09.+Nervous+System

7. http://www.eumanagement.eu/Sehsystem/Sehen1/Kurs1c.html

8. http://en.wikipedia.org/wiki/Na%2B/K%2B-ATPase

9. http://en.wikipedia.org/wiki/Finite_difference_method

8. Appendices

8.1. Appendix A: Derivation of Analytical Solution

We use separation of variable letting u(*x*, *y*, *t*) = X(*x*) Y(*y*) T(*t*).

By plugging into the PDE we get

$$XYT' = D(X''YT + XY''T) - CXYT \quad (1)$$

Rearranging

$$\frac{X''}{X} = -\frac{Y''}{Y} + \frac{1}{D}\frac{T''}{T} + \frac{C}{D} = -\lambda \quad (2)$$

Extracting ODE for X(*x*), we get

$$X'' + \lambda X = 0 \quad (3)$$

Rearranging (2)

$$\frac{Y''}{Y} = \frac{1}{D}\frac{T''}{T} + \frac{C}{D} + \lambda = -\mu \quad (4)$$

Extracting ODE for Y(*y*), we get

$$Y'' + \mu Y = 0 \quad (5)$$

For ODE for T, we get

$$\frac{T'}{T} = -D\,(\mu + \lambda) - C \tag{6}$$

By integrating both sides of (6) with respect to variable t,

$$T(t) = e^{-D(\mu+\lambda)t}\, e^{-Ct} \tag{7}$$

From (3) and boundary conditions for X(x)

$\frac{\partial u}{\partial x}$ $(0, y, t) = X'(0)\, Y(y)\, T(t) = 0 \rightarrow X'(0) = 0$

$\frac{\partial u}{\partial x}$ $(L, y, t) = X'(L)\, Y(y)\, T(t) = 0 \rightarrow X'(L) = 0$

We can solve for X(x) by considering three cases where i) $\lambda = 0$, ii) $\lambda < 0$, iii) $\lambda > 0$.

i) $\lambda = 0$:
From (3), $X'' = 0$, $X(x) = Ax + B$. Thus $X'(x) = A$.
From BC, $A = 0$.
Thus we have $X(x) = B = \alpha_0$

ii) $\lambda < 0$:
From (3), $X'' + \lambda X = 0$, $X(x) = Ae^{ax} + Be^{-ax}$, where $a^2 = \lambda$ and $a>0$.
Thus $X'(x) = aAe^{ax} - aBe^{-ax}$
From BC, A=B and $X'(L) = 0 = aA(e^{aL} - e^{-aL})$
Since $e^{aL} - e^{-aL}$ cannot be zero given $L>0$, A = 0 = B which result zero solution.

iii) $\lambda > 0$:
$X(x) = A\cos(ax) + B\sin(ax)$, where $a^2 = \lambda$ and $a>0$.
$X'(x) = -Aa\sin(ax) + Ba\cos(ax)$.
From BC, $X'(0) = B = 0$ and $X'(L) = -Aa\sin(aL) = 0$.
Thus $aL = m\pi$, m = 1,2,3.. and by setting $A = \alpha_m$ we get
$X(x) = \alpha_m \cos\left(\frac{m\pi x}{L}\right)$, m = 1,2,3…, $\lambda = (\frac{m\pi}{L})^2$

Combining i) and iii) and by superposition principle we get

$$X(x) = \alpha_0 + \sum_{m=1}^{\infty} \alpha_m \cos(\frac{m\pi x}{L}) \quad \text{where } \lambda = (\frac{m\pi}{L})^2 \tag{8}$$

From (5) and by using the same procedure in solving for X(x) we get

$$Y(y) = \beta_0 + \sum_{n=1}^{\infty} \beta_n \cos(\frac{n\pi y}{L}) \quad \text{where } \mu = (\frac{n\pi}{L})^2 \tag{9}$$

combining (7), (8) and (9)

$$u(x, y, t) = \left\{\alpha_0 + \sum_{m=1}^{\infty} \alpha_m \cos(\frac{m\pi x}{L})\right\}\left\{\beta_0 + \sum_{n=1}^{\infty} \beta_n \cos(\frac{n\pi y}{L})\right\}$$

$\exp(-D((\frac{m\pi}{L})^2 + (\frac{n\pi}{L})^2)t)\ \exp(-Ct)$

By rearranging the terms and combining the coefficients α and β into $A_{m,n}$

$$u(x, y, t) = \left\{ \begin{array}{c} \frac{A_{0,0}}{4} + \frac{1}{2}\sum_{k=1}^{\infty}(A_{k,0}\cos(\frac{k\pi x}{L}) + A_{0,k}\cos(\frac{k\pi y}{L}))\exp(-D\left(\frac{k\pi}{L}\right)^2 t) + \\ \sum_{m=1}^{\infty}\sum_{m=1}^{\infty} A_{m,n}\cos(\frac{m\pi x}{L})\cos(\frac{n\pi y}{L})\ \exp(-D((\frac{m\pi}{L})^2 + (\frac{n\pi}{L})^2)t) \end{array} \right\} \exp(-Ct)$$

By performing double Fourier expansion with initial condition, we obtain $A_{m,n}$

$$A_{m,n} = \frac{4}{L^2}\int_0^L\int_0^L u_0(x,y)\cos\left(\frac{m\pi x}{L}\right)\cos\left(\frac{n\pi y}{L}\right)dxdy$$

$$= \frac{4}{L^2}\int_0^L\int_0^L \left\{ \begin{array}{c} V_1\delta(x)\delta(y-p_1) + V_2\delta(x-p_2)\delta(y) + \\ V_3\delta(x-L)\delta(y-p_3) + V_4\delta(x-p_4)\delta(y-L) \end{array} \right\} \cos\left(\frac{m\pi x}{L}\right)\cos\left(\frac{n\pi y}{L}\right)\ dxdy$$

$$= \frac{4}{L^2}\left\{V_1 \cos\left(\frac{n\pi p_1}{L}\right) + V_2 \cos\left(\frac{m\pi p_2}{L}\right) + V_3 \cos(m\pi)\cos\left(\frac{n\pi p_3}{L}\right) + V_4 \cos(n\pi)\cos\left(\frac{m\pi p_4}{L}\right)\right\}$$

8.2. Appendix B: Derivation of 2 Dimensional Forward Finite Difference Method

Given the original PDE, $\frac{\partial u}{\partial t} = D\,(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}) - Cu$

we first discretize each term as follows:

$$\frac{\partial u}{\partial t} = \frac{u(x,y,t+\Delta t) - u(x,y,t)}{\Delta t}$$

$$\frac{\partial^2 u}{\partial x^2} = \frac{\frac{u(x+\Delta x,y,t) - u(x,y,t)}{\Delta x} - \frac{u(x,y,t) - u(x-\Delta x,y,t)}{\Delta x}}{\Delta x}$$
$$= \frac{1}{\Delta x^2}\left(u(x+\Delta x,y,t) - 2u(x,y,t) + u(x-\Delta x,y,t)\right)$$

At the boundary where x = 0, $\frac{u(x,y,t)-u(x-\Delta x,y,t)}{\Delta x} = 0$ so

$$\frac{\partial^2 u}{\partial x^2} = \frac{\frac{u(x+\Delta x,y,t) - u(x,y,t)}{\Delta x}}{\Delta x}$$
$$= \frac{1}{\Delta x^2}\left(u(x+\Delta x,y,t) - u(x,y,t)\right)$$

At the boundary where x = L, $\frac{u(x+\Delta x,y,t)-u(x,y,t)}{\Delta x} = 0$ so

$$\frac{\partial^2 u}{\partial x^2} = \frac{-\frac{u(x,y,t) - u(x-\Delta x,y,t)}{\Delta x}}{\Delta x}$$
$$= \frac{1}{\Delta x^2}\left(-u(x,y,t) + u(x-\Delta x,y,t)\right)$$

For y,

$$\frac{\partial^2 u}{\partial y^2} = \frac{\frac{u(x, y + \Delta y, t) - u(x, y, t)}{\Delta y} - \frac{u(x, y, t) - u(x, y - \Delta y, t)}{\Delta y}}{\Delta y}$$

$$= \frac{1}{\Delta y^2} \left(u(x, y + \Delta y, t) - 2u(x, y, t) + u(x, y - \Delta y, t)\right)$$

At the boundary where y = 0,

$$\frac{\partial^2 u}{\partial y^2} = \frac{1}{\Delta y^2} \left(u(x, y + \Delta y, t) - u(x, y, t)\right)$$

At the boundary where y = L,

$$\frac{\partial^2 u}{\partial y^2} = \frac{1}{\Delta y^2} \left(-u(x, y, t) + u(x, y - \Delta y, t)\right)$$

By plugging in to the original PDE $\frac{\partial u}{\partial t} = D\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) - Cu$

$$\frac{u(x,y,t+\Delta t) - u(x,y,t)}{\Delta t} = \frac{D}{\Delta x^2}\{u(x + \Delta x, y, t) - 2u(x, y, t) + u(x - \Delta x, y, t)\}$$
$$+ \frac{D}{\Delta y^2}\{u(x, y + \Delta y, t) - 2u(x, y, t) + u(x, y - \Delta y, t)\} - Cu(x, y, t)$$

By rearranging the terms, we get

$$u(x, y, t + \Delta t) = u(x, y, t) + \frac{D\Delta t}{\Delta x^2}\{u(x + \Delta x, y, t) - 2u(x, y, t) + u(x - \Delta x, y, t)\}$$
$$+ \frac{D\Delta t}{\Delta y^2}\{u(x, y + \Delta y, t) - 2u(x, y, t) + u(x, y - \Delta y, t)\} - C\,\Delta t\, u(x, y, t)$$

In terms of array operation, we can express as

$$u_{x,y,t+1} = \frac{D\Delta t}{\Delta x^2}\left(u_{x+1,y,t} - 2u_{x,y,t} + u_{x-1,y,t}\right) + \frac{D\Delta t}{\Delta y^2}\left(u_{x,y+1,t} - 2u_{x,y,t} + u_{x,y-1,t}\right) - C\,\Delta t\, u_{x,y,t}$$

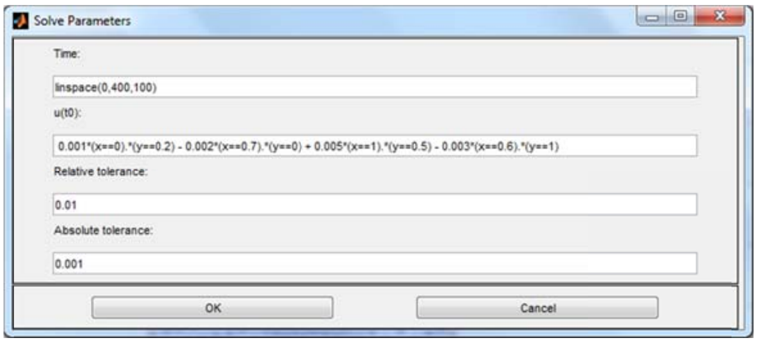
which is stable under condition: $\frac{\Delta t}{\Delta x^2} < \frac{1}{4}$

8.3. Appendix C: MATLAB PDE Toolbox

The numerical solution to our diffusion equation was also computed using the Matlab PDE toolbox. This toolbox has a built in graphical user interface (GUI) that uses the finite element method to solve the partial differential equation and plot the graph over a change in time. The PDE toolbox provides an easy solution to solving 2D PDE's. However, it is limited in the types of equations it is capable of solving, as we will see later.

To solve our 2D PDE, the PDE toolbox was first accessed in Matlab by typing "pdetool" into the command window. We changed the equation to the diffusion equation by selecting it from the drop down menu. The model of the soma was drawn using the rectangle tool and the length and width were set to 1 as we defined it in our problem.

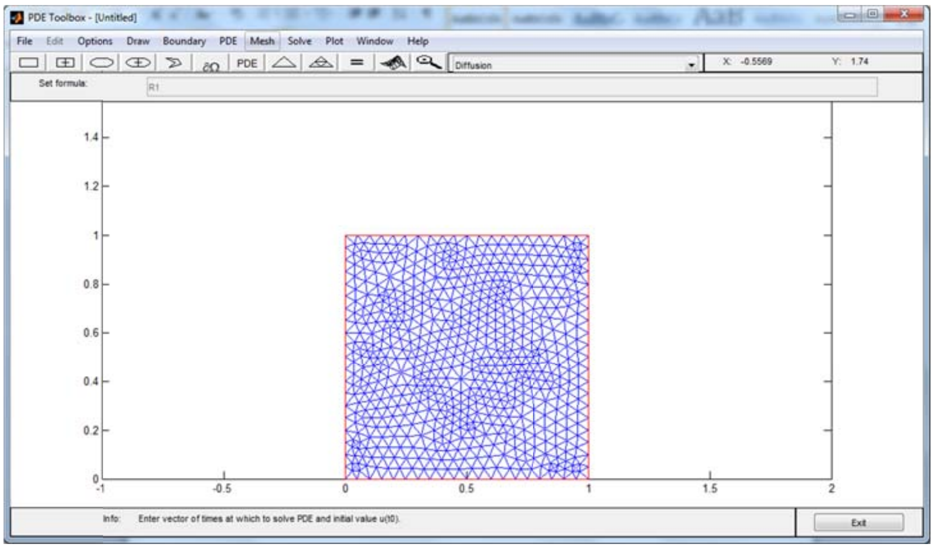
The next step was to enter in our boundary conditions for problem. Because we have flux boundary conditions, we selected the Neumann condition type and set the flux and transfer coefficient to be both 0. Under the PDE Specifications, the diffusion constant was changed to 0.0007 and the type selected was parabolic since our diffusion is dependent on time. For the initial conditions we needed to come up with an equation to implement multiple delta functions. To do this we used the notation x==value, where the value is the position along the x-axis that the delta is to be placed. Using this notation we could then place delta functions along specific points on the x and y axes by multiplying the two positions together.



Time:
linspace(0,400,100)
u(t0):
0.001*(x==0).*(y==0.2) - 0.002*(x==0.7).*(y==0) + 0.005*(x==1).*(y==0.5) - 0.003*(x==0.6).*(y==1)
Relative tolerance:
0.01
Absolute tolerance:
0.001

Figure 10. Setting the time vector and initial conditions.

The final step to finding the numerical solution was to set up the mesh for the model and solve the equation. The mesh was first initialized and then refined to provide a uniform mesh of smaller triangles. This refined mesh leads to a better approximation of the solution. With the mesh in place, the PDE was then solved and visualized with the 3D graphs.



Diffusion
X: -0.5569 Y: 1.74
Set formula: R1
Info: Enter vector of times at which to solve PDE and initial value u(t0).

Figure 11. Initializing and refining the mesh.

8.4. Appendix D: MATLAB Code

```
function Dendrite()
clear;close all; clc

C = .004;       % decaying constant for Na+/K+ ATPase
D = 0.0007;      % diffusivity
L = 1;          % length of the square
T = 400;        % ending time

%inputs: one input for each sides
%      x=0        y=0        x=L        y=L
in = [  L*0.20      L*0.70      L*0.5       L*0.6    ;  % location
       0.001     -0.002      0.005     -0.003 ];   % voltage

mLoc = [L*0.5 L*0.5];   % monitoring location: center of soma
sSize = 20;            % series precision

% grid for x, y, t
nxStep = 20;           % change this value as appropriate for Figure 7.
nyStep = 20;           % change this value as appropriate for Figure 7.
ntStep = 2000;          % change this value as appropriate for Figure 7.

tVec = 0:T/ntStep:T;
xVec = 0:L/nxStep:L;
yVec = 0:L/nyStep:L;

nt = length(tVec);
nx = length(xVec);
ny = length(yVec);

% analytic solution monitoring at the center location
mVecAn = size(nt);
for tIdx=1:nt
   mVecAn(tIdx) = u(mLoc(1),mLoc(2), tVec(tIdx), L, C, D, in, sSize);
end

%-- Experiments for Figure 6 --
%        x=0        y=0        x=L        y=L
ep1 = [  L*0.20      L*0.70      L*0.5       L*0.9    ; % location
        0.03       0.0         0.0         0.0 ];     % voltage
ep2 = [  L*0.20      L*0.70      L*0.5       L*0.9    ; % location
        0.03       0.03       0.0         0.0 ];      % voltage
ep3 = [  L*0.20      L*0.70      L*0.5       L*0.9    ; % location
        0.03       0.03       0.03       0.0 ];       % voltage
ep4 = [  L*0.20      L*0.70      L*0.5       L*0.9    ; % location
        0.03       0.03       0.03       0.03 ];      % voltage

ep3ip1 = [  L*0.20      L*0.70      L*0.5       L*0.9    ;  % location
        0.03       0.03       -0.03       0.03 ];        % voltage
ip2 =  [  L*0.20      L*0.70      L*0.5       L*0.9    ;   % location
        0.0       -0.03       -0.03       0.0 ];         % voltage

tsVal = 0.030; % threshold value

for tIdx=1:nt
```

```
    mv1(tIdx) = u(mLoc(1),mLoc(2), tVec(tIdx), L, C, D, ep1, sSize);
    mv2(tIdx) = u(mLoc(1),mLoc(2), tVec(tIdx), L, C, D, ep2, sSize);
    mv3(tIdx) = u(mLoc(1),mLoc(2), tVec(tIdx), L, C, D, ep3, sSize);
    mv4(tIdx) = u(mLoc(1),mLoc(2), tVec(tIdx), L, C, D, ep4, sSize);
    mvep3ip1(tIdx) = u(mLoc(1),mLoc(2), tVec(tIdx), L, C, D, ep3ip1, sSize);
    mvip2(tIdx) = u(mLoc(1),mLoc(2), tVec(tIdx), L, C, D, ip2, sSize);
    ts(tIdx) = tsVal;
end
figure (1)  % comparing different number of excitatory input signal
plot(tVec,mv1,tVec,mv2,tVec,mv3,tVec,mv4,tVec,ts);
legend('1 EPSP','2 EPSP','3 EPSP','4 EPSP','Threshold');
title('Vm at the center');
ylim([-.03 0.06]);
xlabel('Time (second)');
ylabel('Vm (volt)');

figure (2)  % comparing excitatory vs inhibitory input signal
plot(tVec,mv4,tVec,mvep3ip1,tVec,mvip2,tVec,ts);
legend('4 EPSP','3 EPSP + 1 IPSP','2 IPSP','Threshold');
title('Vm at the center');
ylim([-.06 0.06]);
xlabel('Time (second)');
ylabel('Vm (volt)');

% -- 2D Animation for Analytic Solution --
 potAn = zeros (ny, nx);
 figure(3)
 for tIdx=1:nt
     for xIdx=1:nx
         for yIdx=1:ny
             potAn(yIdx, xIdx) = u(xVec(xIdx),yVec(yIdx), tVec(tIdx), L, C, D,
in, sSize);
         end
     end
     surf(xVec, yVec, potAn);
     xlim([0,L]);
     ylim([0,L]);
     %zlim([-1 1.5]);
     xlabel('x');
     ylabel('y');
     zlabel('voltage');
     %title('T = 400');
     colorbar
     M(tIdx) = getframe;
end
%movie2avi(M,'animation.avi');%,'compression','none');

%-- 2D Finite Difference Method --
potFDM = zeros (ny, nx);
potFDM_nxt = zeros(ny, nx);
for xIdx=1:nx
    for yIdx=1:ny
        potFDM(yIdx,xIdx) = u(xVec(xIdx),yVec(yIdx), 0, L, C, D, in, sSize);
    end
end

dT = T/ntStep;
dX = L/nxStep;
```

```
dY = L/nyStep;
eX = D*dT/dX^2;
eY = D*dT/dY^2;

%-- Single Monitoring Point --
mVecFDM = size(nt);
mxIdx = findIdx(xVec, mLoc(1));
myIdx = findIdx(yVec, mLoc(2));
frameIdx = 1;
%----- display of num sol monitor --------
figure(4)
for tIdx=1:nt
if (tIdx == 20) % change this condition as appropriate for snapshot
    surf(xVec, yVec, potFDM);
    xlabel('x');
    ylabel('y');
    zlabel('voltage');
    title('T = 400');
    xlim([0,L]);
    ylim([0,L]);
    zlim([-1 1]);
    colorbar
    caxis([-0.001 0.001]);
    M(tIdx) = getframe;
    M(frameIdx) = getframe;
    frameIdx = frameIdx + 1;
end
    mVecFDM(tIdx) = potFDM(myIdx, mxIdx);
    for xIdx=1:nx
        for yIdx=1:ny
            if (xIdx == 1)
                xVal = potFDM(yIdx, xIdx+1) - potFDM(yIdx, xIdx);
            else if (xIdx == nx)
                    xVal = -potFDM(yIdx, xIdx) + potFDM(yIdx, xIdx-1);
                else
                    xVal = potFDM(yIdx, xIdx+1)-2*potFDM(yIdx, xIdx) +
potFDM(yIdx, xIdx-1);
                end
            end

            if (yIdx == 1)
                yVal = potFDM(yIdx+1, xIdx) - potFDM(yIdx, xIdx);
            else if (yIdx == ny)
                    yVal = -potFDM(yIdx, xIdx) + potFDM(yIdx-1, xIdx);
                else
                    yVal = potFDM(yIdx+1, xIdx)-2*potFDM(yIdx, xIdx) +
potFDM(yIdx-1, xIdx);
                end
            end
            potFDM_nxt(yIdx,xIdx) = (1-C*dT)*potFDM(yIdx,xIdx) + eX*xVal +
eY*yVal;
        end
    end
    potFDM = potFDM_nxt;
end

% -- Comparing FDM vs Analytical Solution for Figure 7
figure (5)
```

```
plot(tVec, mVecFDM, tVec, mVecAn);
legend('Euler FDM','Analytical');
title('analytical vs numerical method');
xlabel('Time (second)');
ylabel('Vm (volt)');
ylim([-0.003 0.003]);

end

function out = findIdx(vec, value)
nv = length(vec);
out = nv;
for i=1:nv-1
    if(vec(i)<=value & value < vec(i+1))
        out = i;
        return;
    end
end
end

function out = u(x,y,t, L, C, D, in, sSize)
out = 0;
for m=0:sSize
    for n=0:sSize
        tVal = exp((-C-D*((m*pi/L)^2+(n*pi/L)^2))*t);
        xZ = in(2,1)*cos(n*(pi/L)*in(1,1));
        yZ = in(2,2)*cos(m*(pi/L)*in(1,2));
        xL = cos(m*pi)*in(2,3)*cos(n*(pi/L)*in(1,3));
        yL = cos(n*pi)*in(2,4)*cos(m*(pi/L)*in(1,4));
        xyVal = (xZ + xL + yZ + yL)*cos(m*pi*x/L)*cos(n*pi*y/L);
        if (m==0)
            if (n==0)
                area = L^2;
            else
                area = L^2/2;
            end
        else
            if (n==0)
                area = L^2/2;
            else
                area = L^2/4;
            end
        end
        out = out + exp(-C*t)*xyVal*tVal/area;
    end
end
end
```

8.5. Appendix E: MATLAB PDE Toolbox Code

```
function pdemodel
[pde_fig,ax]=pdeinit;
pdetool('appl_cb',10);
set(ax,'DataAspectRatio',[1 1 1]);
set(ax,'PlotBoxAspectRatio',[870.40000000000009 580.26666666666677
725.33333333333337]);
```

```
set(ax,'XLimMode','auto');
set(ax,'YLim',[0 2]);
set(ax,'XTickMode','auto');
set(ax,'YTickMode','auto');

% Geometry description:
pderect([-0 1 1 0],'R1');
set(findobj(get(pde_fig,'Children'),'Tag','PDEEval'),'String','R1')

% Boundary conditions:
pdetool('changemode',0)
pdesetbd(4,...
'neu',...
1,...
'0',...
'0')
pdesetbd(3,...
'neu',...
1,...
'0',...
'0')
pdesetbd(2,...
'neu',...
1,...
'0',...
'0')
pdesetbd(1,...
'neu',...
1,...
'0',...
'0')

% Mesh generation:
setappdata(pde_fig,'Hgrad',1.3);
setappdata(pde_fig,'refinemethod','regular');
setappdata(pde_fig,'jiggle',char('on','mean',''));
pdetool('initmesh')
pdetool('refine')
pdetool('jiggle')

% PDE coefficients:
pdeseteq(2,...
'.0007',...
'0.0',...
'0',...
'1.0',...
'linspace(0,400,100)',...
' 0.001*(x==0).*(y==0.2) - 0.002*(x==0.7).*(y==0) + 0.005*(x==1).*(y==0.5)
- 0.003*(x==0.6).*(y==1)',...
'0.0',...
'[0 100]')
setappdata(pde_fig,'currparam',...
['.0007';...
'0    '])

% Solve parameters:
setappdata(pde_fig,'solveparam',...
str2mat('0','1872','10','pdeadworst',...
'0.5','longest','0','1E-4','','fixed','Inf'))

% Plotflags and user data strings:
setappdata(pde_fig,'plotflags',[1 1 1 1 1 1 7 0 1 1 0 100 1 0 1 0 0 1]);
setappdata(pde_fig,'colstring','');
setappdata(pde_fig,'arrowstring','');
setappdata(pde_fig,'deformstring','');
setappdata(pde_fig,'heightstring','');

% Solve PDE:
pdetool('solve')
```