**BIOMEDE 517**

**Name: Diksha Anoop Kumar Zutshi**
**UMID:34050139**
**Date: 02/09/2024**

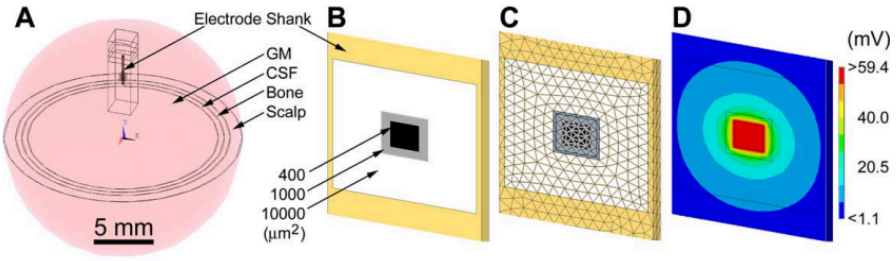
## Lab 3&4
## Introduction:

In this report, we delve into the realm of electrostatic models, focusing on their pivotal role in deep brain stimulation (DBS) research. By drawing upon historical experiment data, such as resistivity, conductivity, and neuronal structure, we aim to elucidate the intricate relationship between externally applied voltages and the internal currents necessary for neuronal activation. This exploration is further enriched by our construction of deep brain simulation models within the COMSOL framework, alongside the simulation of electrode data both in the presence and absence of noise. A significant portion of our analysis is dedicated to understanding the K matrix, which represents the voltage distribution along a neuron's axis, and its correlation with external voltages generated by internal currents. Through our comprehensive study, we strive to pinpoint the minimal internal current required for neuronal activation via external stimulation and to map the effective distance range for such activation, thereby offering novel insights into the optimization of DBS strategies.

## Methods:
### Lab-3:

Moffit & McIntyre (2005), assumed that the tissue surrounding the electrode behaves as a purely resistive and linear medium for electric currents, following Ohm's law. The models are static and do not account for the dynamic changes in tissue properties or electrode-tissue interfaces over time. Despite using high-impedance measurement equipment, the model assumes the electrode-electrolyte interfacial impedance's effect on the recording or stimulation is negligible.

To model the electrical properties of the head, concentric spheres are used, which simplifies the geometric and compositional complexities of the head and brain with current source and sinks in the brain and skull as the ground.




The rat head was modeled as 4 concentric spheres representing the brain, cerebrospinal fluid (CSF), skull, and scalp with radii of 8000, 8500, 9000, and 10,000 mm, respectively. The figures were 10x for the human brain.As we can observe the aim is to create an electric mesh.

**Lab-4:**

Warman and Durand (1992), made certain assumptions for predicting axonal excitation due to electric fields simplifying biological reality to make computations feasible. Key simplifications include treating myelin as a perfect insulator, assuming a homogeneous isotropic extracellular medium, and using linear cable models for subthreshold behavior. The paper also leverages the response of a passive cable to predict excitation. The axon is modeled as an electrical network with distinct nodes and internodal spacing, with myelin assumed to be a perfect insulator. Axonal excitation is determined by the extracellular potentials, which are themselves assumed not to be significantly affected by the fiber response. The use of linear cable models for subthreshold behavior and the approximation of membrane activation based on induced polarization exceeding a certain transmembrane voltage.

We calculated the effect the internal currents of a neuron from all its compartments, at a certain distance from the electrode, will have on the external potential at the electrode. When obtaining the signal all the neurons in the vicinity of the electrode will have contributions to the resultant signal. We can decide the firing rate for each signal from individual neurons and combine. After adding noise to the resultant recording, the simulation of the neural signal looks as good as the original. It's better to discard the signal from neurons which are in 25 um vicinity from the neuron, as it's a dead zone.
We calculated the external voltage from internal currents and K matrix from COMSOL, by interpolating the voltages at the coordinates of our present neuron.
We calculated the internal currents of the neuron from the stimulus current and electrode distance from the neuron by using the predefined equations from Warman & Durand (1992), it loops over all the neuron compartments multiplying the conductivity and the second difference of voltage at that compartment.

In the neural recording model, various elements were altered to examine their effects on recordings. These modifications included adding multiple neurons to simulate a more complex neural network, adjusting cell sizes to reflect differences between large and small neurons, incorporating noise to mimic real-world recording conditions, and utilizing a K matrix for modeling variable conductivity environments. Each change aimed to enhance the model's realism and investigate how such factors influence neural signal recordings, providing insights into the complexities of electrophysiological recordings and deep brain stimulation.

**Results:**

**Lab-3:**

The decreasing trend can be observed in figure1, in the voltage magnitude as the distance from the point source increases.The voltage drops sharply close to the current source, reflecting the strong electric field near the point of injection.The rate of decrease in voltage should become less steep as the distance increases, indicating the diffusive spread of the electric field through the head model's conductive media.In the figure2, we can observe the model as a mesh of electric field.

In the figure3, we can observe a peak at (0,0,0) as there's a current source of 5mA. We can also observe two dips because of current sinks at +3 mm and -3mm in y direction, the superior point with 1.25 mA current sink

and the inferior point a 3.75 mA current sink. In figure4, we can observe the presence of two current sinks in the bright regions.In figure 5, we can observe the points of current source and sinks.

In the figure 6, we can observe the second spatial derivative of the voltage,it portrays how the curvature of the voltage distribution changes as you move away from the active DBS contact, perpendicular to the probe.

**Lab-4:**

Figure 7 & 8 consists of external voltage plots at the electrode at distances of 50, 100, 200, 300 um from the neuron, with internal currents as Big currents and small currents respectively.

Figure 9 consists of a visualization of multiple neurons passing through the cube making sure that their axon hillocks are inside the cube. With the electrode at the origin of the cube (0,0,0).

Figure 10 consists of the combined signals from all neuronal voltage contributions with and without the dropped neurons from the dead zone, which is 25um from the electrode."Dead zone" refers to an area immediately surrounding the electrode where neuronal signals are not effectively recorded or stimulated. This can be due to tissue damage from electrode insertion, changes in tissue conductivity, or the physical separation between neurons and electrode surfaces. The dead zone can significantly affect the efficacy of neural interfaces by limiting the spatial resolution of recordings and reducing the precision of targeted stimulation. Recognizing and accounting for the dead zone is crucial in designing and interpreting neural interface experiments.

Figure 11 consists of the combined signals from all neuronal voltage contributions with added noise, which makes the simulated signal more realistic to the actual electrode recordings.

Figure 12 consists of the modeled voltage waveform from the interpolated voltage matrices from COMSOLS and treating them as K matrix and along with internal currents, finding the external voltage at the electrode.

Minimum current for action potential in an axon that is 20 um in diameter and 5mm in length is 40pA. (NEURON MODEL)

Minimum external current from from 1mm distance to activate the axon is >1A

I(internal)= ga*second difference of voltage at the particular node.

I(n)= ga*[Ve(n-1)-2Ve(n)+Ve(n+1)]

Increasing model complexity in neural recording and deep brain stimulation simulations can significantly enhance the accuracy and realism of the predictions. More complex models can account for the heterogeneous and anisotropic nature of neural tissue, interactions between multiple neurons, and the impact of various noise sources on signal fidelity. This leads to better understanding and prediction of how neural interfaces interact with biological tissue, allowing for the optimization of electrode designs and stimulation protocols. However, increased complexity also demands more computational resources and sophisticated modeling techniques, potentially making simulations more challenging to execute and interpret.

**Discussions:**

**Lab-3:**

The curvature of the second spatial derivative of voltage provides insight into how sharply or smoothly the voltage changes in space, which is crucial for understanding the spatial specificity of DBS effects: High Magnitude of Derivative: Sharp changes in voltage suggest that DBS effects are highly localized around the active contact. High peaks in the second derivative indicate regions with rapid changes in the electric field, potentially corresponding to areas of strong neural activation or inhibition. Low Magnitude of Derivative: Smoother curves or smaller magnitudes suggest a more gradual spread of the DBS effect, indicating that the stimulation might affect a broader area with less specificity.

The electric field generated by DBS electrodes and to understand its spatial distribution within the brain allows for the optimization of electrode placement and stimulation parameters, ensuring maximum efficacy while minimizing side effects. This approach could lead to the development of patient-specific DBS therapies, where treatment parameters are tailored based on the individual's unique anatomical and physiological characteristics, thereby enhancing therapeutic outcomes.

**Lab-4:**

From Warman and Durand (1992), the equation to find the internal current essentially calculates the second spatial derivative of the external voltage along the axon, which corresponds to the activating function. This function plays a critical role in determining the sites of action potential initiation and how the external electric field influences the neuron's internal current. We can also determine the stimulus current needed from a certain distance, to generate the action potential in the cell.

Looking forward, the models developed in this lab can serve as a foundational tool for advancing neural interface technologies. By accurately simulating how neurons respond to external stimuli, these models can be used to design more effective neural prosthetics, optimize deep brain stimulation (DBS) parameters for therapeutic applications, and enhance the resolution of brain-machine interfaces (BMIs). Furthermore, the ability to model complex conductivity environments and account for noise in neural recordings can lead to the development of more sensitive and selective recording devices, enabling the detection of subtler neural signals amidst the noise. These advancements could significantly improve the quality of life for individuals with neurological disorders, offer new insights into brain function, and pave the way for novel neurotechnological applications.

**Appendix:**

**Figures & Tables:**

**Lab 3:**

Figure 1

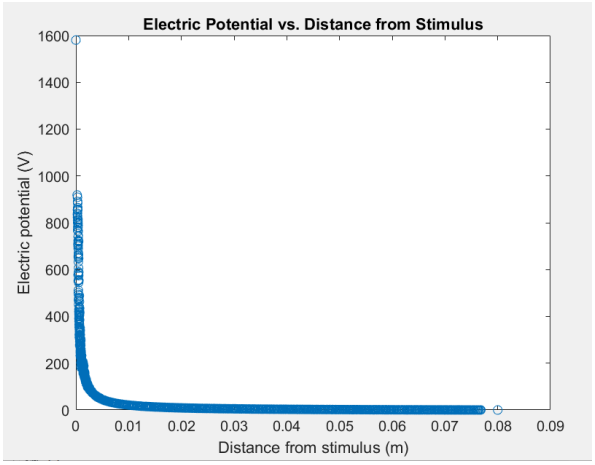



Figure 2

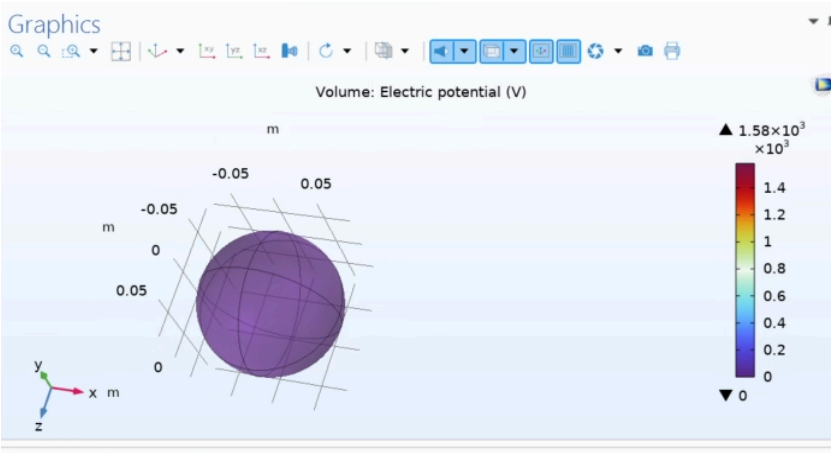



Figure 3

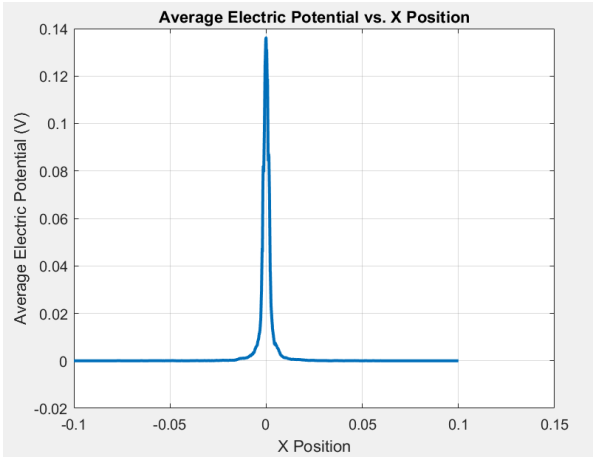



Figure 4

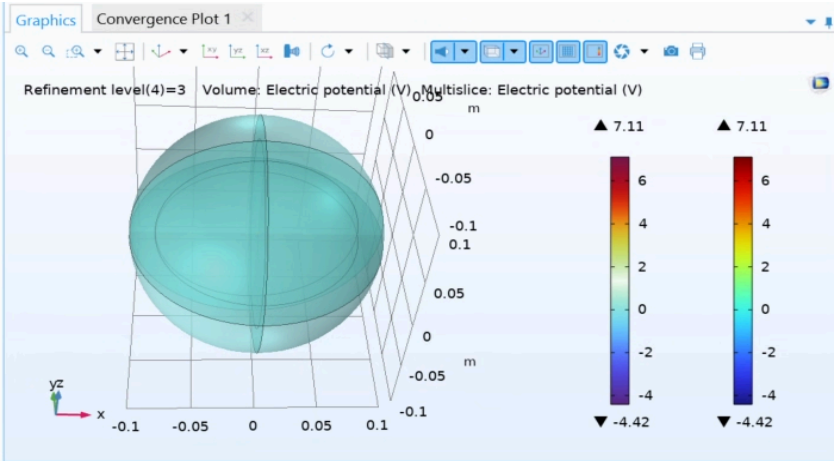



Figure 5

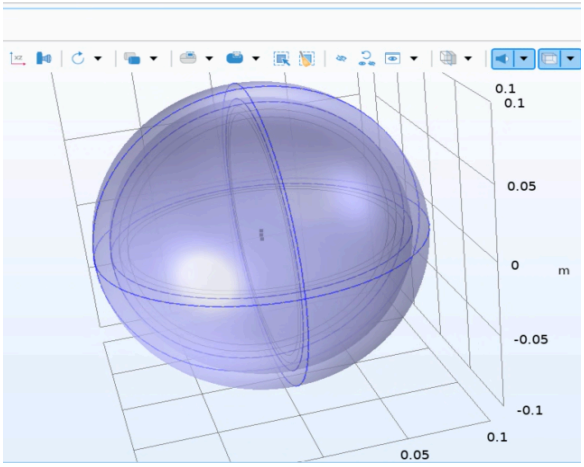
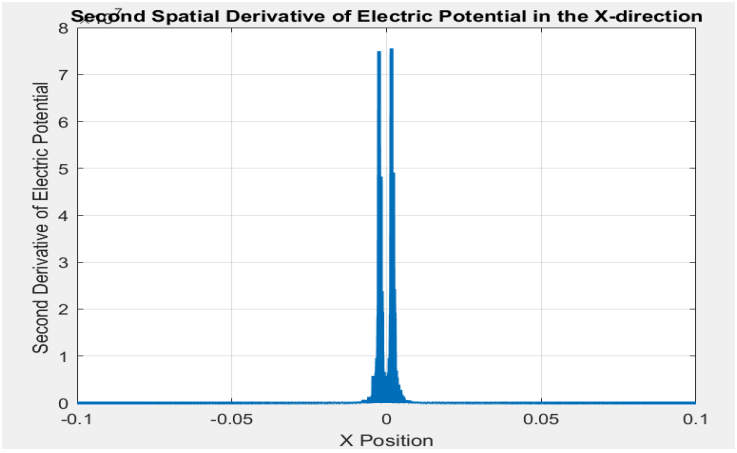



Figure 6



**Lab-4:**

Figure 7

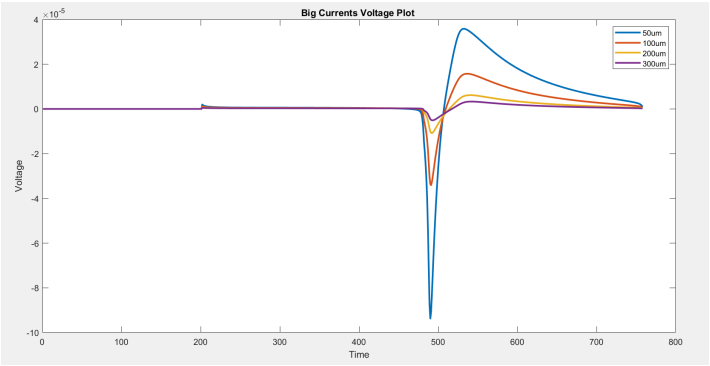



Figure 8

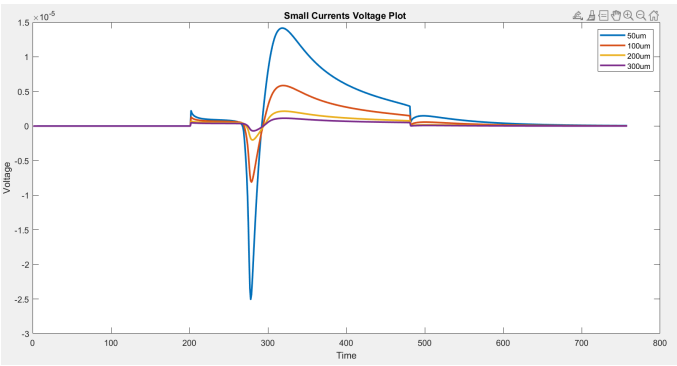



Figure 9

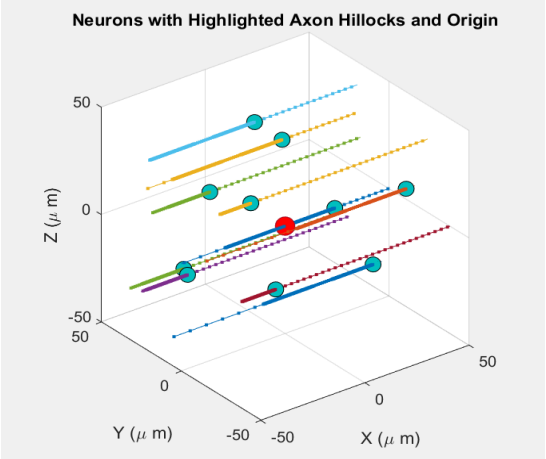



Figure 10

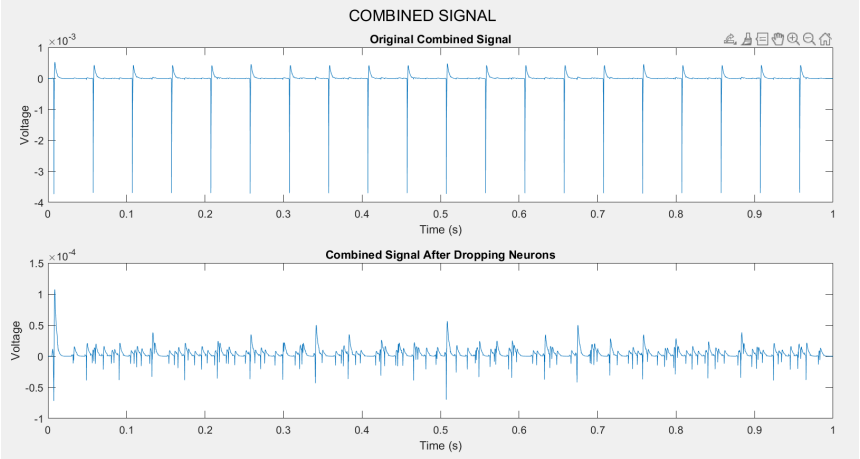



Figure 11

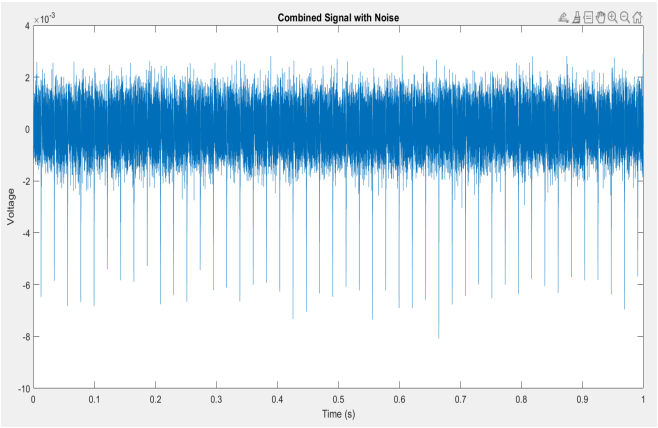
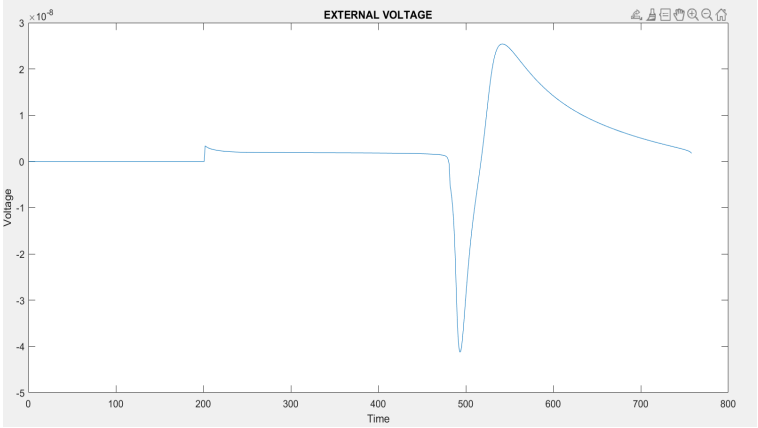



Figure 12



**Code:**

**Lab 3:**

**Part 2:**

```
numHeaderLines = 9; % Adjust this number based on the actual number of header lines in your
file.
% Read the file into a table, skipping the header lines
data = readtable('multislice.txt', 'HeaderLines', numHeaderLines);
x_pos=data.Var1;
y_pos=data.Var2;
z_pos=data.Var3;
electric_potential = data.Var4;
% Calculate the distance from the stimulus for each data point
distances = sqrt(x_pos.^2 + y_pos.^2 + z_pos.^2);
% Create a plot of electric potential vs. distance
figure;
plot(distances, electric_potential, 'o');
xlabel('Distance from stimulus (m)');
ylabel('Electric potential (V)');
title('Electric Potential vs. Distance from Stimulus');
```

**Part3:**

```
% Specify the number of header lines in the file
numHeaderLines = 8;
% Read the file into a table, skipping the header lines
data = readtable('brain_part3.txt', 'HeaderLines', numHeaderLines);
% Extract positions and electric potential
x_pos = data.Var1;
y_pos = data.Var2;
z_pos = data.Var3;
electric_potential = data.Var4;
% Calculate the distance from the stimulus for each data point
distances = sqrt(x_pos.^2 + y_pos.^2 + z_pos.^2);
% Plot electric potential vs. distance
figure;
plot(electric_potential, distances, 'o', 'MarkerSize', 3);
ylabel('Distance from stimulus (units)');
xlabel('Electric potential (V)');
title('Electric Potential vs. Distance from Stimulus');
% Define a new grid for x positions
x_grid = linspace(min(x_pos), max(x_pos), 30000);
% Initialize the array for the calculated potentials
custom_avg_V = zeros(size(x_grid));
for i = 1:length(x_grid)
   % Find indices and distances for points on the negative and positive sides
   neg_side_indices = find(x_pos < x_grid(i));
    pos_side_indices = find(x_pos >= x_grid(i));  % Use >= to include the point exactly at
x_grid(i) if exists
```

```
    % Initialize averages for negative and positive sides
    avg_neg = NaN;  % Use NaN to handle cases where we don't have enough data points
    avg_pos = NaN;

    % Calculate for negative side if there are enough points
    if length(neg_side_indices) >= 4
        neg_distances = abs(x_pos(neg_side_indices) - x_grid(i));
        [~, neg_indices] = mink(neg_distances, 4);
        direct_avg_neg = mean(electric_potential(neg_side_indices(neg_indices(1:2))));
        weights_neg = 1 ./ neg_distances(neg_indices(3:4));
        normalized_weights_neg = weights_neg / sum(weights_neg);
            weighted_avg_neg  =  sum(electric_potential(neg_side_indices(neg_indices(3:4)))  .*
normalized_weights_neg);
        avg_neg = mean([direct_avg_neg, weighted_avg_neg]);
    end

    % Calculate for positive side if there are enough points
    if length(pos_side_indices) >= 4
        pos_distances = abs(x_pos(pos_side_indices) - x_grid(i));
        [~, pos_indices] = mink(pos_distances, 4);
        direct_avg_pos = mean(electric_potential(pos_side_indices(pos_indices(1:2))));
        weights_pos = 1 ./ pos_distances(pos_indices(3:4));
        normalized_weights_pos = weights_pos / sum(weights_pos);
            weighted_avg_pos  =  sum(electric_potential(pos_side_indices(pos_indices(3:4)))  .*
normalized_weights_pos);
        avg_pos = mean([direct_avg_pos, weighted_avg_pos]);
    end

    % Combine the averages from both sides, ignoring NaN values
    custom_avg_V(i) = mean([avg_neg, avg_pos], 'omitnan');
end
% Plot the custom average electric potential vs. x_grid
windowSize=400;
smoothV1 = smoothdata(custom_avg_V, 'movmean', windowSize);
figure;
plot(x_grid, smoothV1, 'LineWidth', 2);
xlabel('X Position');
ylabel('Average Electric Potential (V)');
title('Average Electric Potential vs. X Position');
grid on;
% Calculate the second derivative using finite differences on the custom averages
h = x_grid(2) - x_grid(1); % Assuming uniform spacing in x_grid
second_derivative_custom = diff(custom_avg_V, 2) / h^2;
% Adjust x_grid to match the size of the second_derivative_custom array after diff operation
x_grid_adjusted_for_custom = x_grid(2:end-1);
% Plot the second derivative of electric potential based on the custom averages
windowSize=600;
smoothV = smoothdata(second_derivative_custom, 'movmean', windowSize);
smoothV=abs(smoothV);
```

```
figure;
plot(x_grid_adjusted_for_custom, smoothV, '-', 'LineWidth', 2);
xlabel('X Position');
ylabel('Second Derivative of Electric Potential ');
title('Second Spatial Derivative of Electric Potential in the X-direction');
grid on;
```

**Lab-4:**

**Part 1:**

```
%%%%%%%%%%%%%%%%%%%%%%%%%NEURON 1
%lest keep origin as the first value of XYZ.
%we need to obtain a point 50um in perpendicular direction to that..y axis.
origin=XYZ(200,:);
current_origin=currents(200,:);
sigma=3.33*10^-7;%um     0.3333S/m
distance1=50;%um
radius= 20;%um
POI=[0.5,50,0];
%calculate Vext for that for origin
Vext_50_origin=current_origin/(4*3.14*sigma*distance1);
%Calculate Vext for every point along the neuron
%voltages from everypoint is added
%sqrt(XYZ-POI)
distances=sqrt((XYZ(:,1)-POI(:,1)).^2+(XYZ(:,2)-POI(:,2)).^2+(XYZ(:,3)-POI(:,3).^2));
Vext_50_ALL=(currents./(4*3.14*sigma.*distances));
Vext_50_ALL=sum(Vext_50_ALL,1);
%plot(Vext_50_ALL(500,:))
%Now we need to repeat this process for points 100um,200um,300um from the axon hilloc: Vext
for all
%%100um
POI_100=[0.5,100,0];
%calculate Vext for that for origin
Vext_100_origin=current_origin/(4*3.14*sigma*100);
%Calculate Vext for every point along the neuron
%sqrt(XYZ-POI)
distances100=sqrt((XYZ(:,1)-POI_100(:,1)).^2+(XYZ(:,2)-POI_100(:,2)).^2+(XYZ(:,3)-POI_100(:,
3).^2));
Vext_100_ALL=sum(currents./(4*3.14*sigma.*distances100),1);
Vext_100_ALL=sum(Vext_100_ALL,1);
%%200um
POI_200=[0.5,200,0];
%calculate Vext for that for origin
Vext_200_origin=current_origin/(4*3.14*sigma*200);
%Calculate Vext for every point along the neuron
%sqrt(XYZ-POI)
distances200=sqrt((XYZ(:,1)-POI_200(:,1)).^2+(XYZ(:,2)-POI_200(:,2)).^2+(XYZ(:,3)-POI_200(:,
3).^2));
Vext_200_ALL=sum(currents./(4*3.14*sigma.*distances200),1);
Vext_200_ALL=sum(Vext_200_ALL,1);
%%300um
POI_300=[0.5,300,0];
%calculate Vext for that for origin
Vext_300_origin=current_origin/(4*3.14*sigma*300);
%Calculate Vext for every point along the neuron
%sqrt(XYZ-POI)
distances300=sqrt((XYZ(:,1)-POI_300(:,1)).^2+(XYZ(:,2)-POI_300(:,2)).^2+(XYZ(:,3)-POI_300(:,
3).^2));
```

```
Vext_300_ALL=sum(currents./(4*3.14*sigma.*distances300),1);
Vext_300_ALL=sum(Vext_300_ALL,1);
%distance at which the AP won't be detectable above a 10uV peakk to peak noise floor?
V_noise_floor = 10 ;%uV
% Function to compare voltage against noise floor
is_detectable = @(voltage, V_noise_floor) voltage > V_noise_floor;
% Check detectability at each distance
detectable_at_50_um = is_detectable(Vext_50_ALL, V_noise_floor);
detectable_at_100_um = is_detectable(Vext_100_ALL, V_noise_floor);
detectable_at_200_um = is_detectable(Vext_200_ALL, V_noise_floor);
detectable_at_300_um = is_detectable(Vext_300_ALL, V_noise_floor);
%%Plotting together
figure;
plot(Vext_50_ALL, 'LineWidth', 2);
hold on;
plot(Vext_100_ALL, 'LineWidth', 2);
hold on;
plot(Vext_200_ALL, 'LineWidth', 2);
hold on;
plot(Vext_300_ALL, 'LineWidth', 2);
title('Small Currents Voltage Plot ')
legend('50um','100um','200um','300um')
xlabel('Time');
ylabel('Voltage')
hold off;
```

**Part 2:**

```
%ques1
%%distance of origin of cube from neuron 1 comp
origin=[0,0,0];
neuron1=[XYZ(:,1),XYZ(:,2),XYZ(:,3)];
num_neurons=10;
hilloc_pos=neuron1(200,:);
neurons = zeros(size(neuron1, 1), 3, num_neurons);
neurons(:,:,1) = neuron1;
relative_positions = neuron1 - hilloc_pos;
% Cube boundaries for the axon hillocks
cube_min = -50;
cube_max = 50;
hillock_positions = zeros(num_neurons, 3); % 10 neurons, 3 coordinates (x, y, z)
hillock_positions(1, :) = hilloc_pos;
%%randon hilloc pos and other coordinates
for i = 2:num_neurons
   % Random position for the axon hillock within the cube
   random_hillock_position = cube_min + (cube_max - cube_min) .* rand(1, 3);
   hillock_positions(i, :) = random_hillock_position;
   neurons(:,:,i) = relative_positions + random_hillock_position;
end
% Plotting each neuron
```

```
figure; hold on; % Create a figure and hold it for multiple plots
% Plotting each neuron and highlighting axon hillocks
for i = 1:10 % Assuming 10 neurons
   neuron_positions = neurons(:,:,i); % Extract the ith neuron's positions
    plot3(neuron_positions(:,1), neuron_positions(:,2), neuron_positions(:,3), '.-'); % Plot
neuron compartments
   % Highlighting the axon hillock for this neuron
     scatter3(hillock_positions(i,1), hillock_positions(i,2), hillock_positions(i,3), 100,
'filled', 'MarkerEdgeColor', 'k', 'MarkerFaceColor', [0 .75 .75]);

end
scatter3(0, 0, 0, 150, 'MarkerFaceColor', 'r');
xlabel('X (\mu m)');
ylabel('Y (\mu m)');
zlabel('Z (\mu m)');
title('Neurons with Highlighted Axon Hillocks and Origin');
axis equal; grid on; % Equal aspect ratio and grid on
xlim([-50 50]); ylim([-50 50]); zlim([-50 50]); %cube boundaries
view(3); % 3D view
hold off;
%%distances from origin of cube:
distances = zeros(10, 531);
neurons_keep = false(num_neurons, 1); % Initialize as logical array
for i = 1:10
   % Extracting the positions for the i-th neuron (531x3)
   neuron_positions = neurons(:,:,i);
   % Calculating the distances for all compartments of the i-th neuron
   distances_i = sqrt(sum((origin-neuron_positions).^2, 2));
   distances(i,:) = distances_i';
    if distances_i > 25
       neurons_keep(i) = true;
   end
end
%%%%ques2
%RANDOM SPIKE TIMES
simulation_duration = 1; % Duration in ms
dt=2.5*10^-5;%0.025ms=>s
spike_duration_ms = 2*10^-4 ;%0.18ms>s;
refractory_period = 0.001; % 1 ms => s, example refractory period
neuron_spike_times = cell(num_neurons, 1); % Cell array to store spike times for each neuron
for i = 1:num_neurons
   % Randomly pick the number of spikes for the i-th neuron
   num_spikes = randi([2, 50]);
   % Calculating the total duration that spikes occupy plus refractory periods
      total_duration_needed = (num_spikes * spike_duration_ms) + ((num_spikes - 1) *
refractory_period);
   % Calculating the available time for intervals between spikes
   available_time = simulation_duration - total_duration_needed;
   % Adjust num_spikes if not enough time
```

```
   while available_time < 0 && num_spikes > 2
       num_spikes = num_spikes - 1;
           total_duration_needed = (num_spikes * spike_duration_ms) + ((num_spikes - 1) *
refractory_period);
       available_time = simulation_duration - total_duration_needed;
   end

   % Calculating evenly distributed intervals between spikes
   interval_between_spikes = available_time / (num_spikes - 1) + refractory_period;
   % Generating spike times starting with a random offset to stagger spikes between neurons
   random_offset = rand() * refractory_period; % Stagger start within one refractory period
    spike_times = cumsum([random_offset, repmat(spike_duration_ms + interval_between_spikes,
1, num_spikes - 1)]);
   neuron_spike_times{i} = spike_times;
end
%making sure none of the spike trails are overlapping
figure; hold on;
scatter(neuron_spike_times{1}, ones(size(neuron_spike_times{1})), 'filled', 'DisplayName',
'Neuron 1', 'SizeData', 10);
scatter(neuron_spike_times{2}, ones(size(neuron_spike_times{2})), 'filled', 'DisplayName',
'Neuron 2', 'SizeData', 10);
scatter(neuron_spike_times{3}, ones(size(neuron_spike_times{3})), 'filled', 'DisplayName',
'Neuron 3', 'SizeData', 10);
scatter(neuron_spike_times{4}, ones(size(neuron_spike_times{4})), 'filled', 'DisplayName',
'Neuron 4', 'SizeData', 10);
scatter(neuron_spike_times{5}, ones(size(neuron_spike_times{5})), 'filled', 'DisplayName',
'Neuron 5', 'SizeData', 10);
scatter(neuron_spike_times{6}, ones(size(neuron_spike_times{6})), 'filled', 'DisplayName',
'Neuron 6', 'SizeData', 10);
scatter(neuron_spike_times{7}, ones(size(neuron_spike_times{7})), 'filled', 'DisplayName',
'Neuron 7', 'SizeData', 10);
scatter(neuron_spike_times{8}, ones(size(neuron_spike_times{8})), 'filled', 'DisplayName',
'Neuron 8', 'SizeData', 10);
scatter(neuron_spike_times{9}, ones(size(neuron_spike_times{9})), 'filled', 'DisplayName',
'Neuron 9', 'SizeData', 10);
scatter(neuron_spike_times{10}, ones(size(neuron_spike_times{10})), 'filled', 'DisplayName',
'Neuron 10', 'SizeData', 10);
hold off;
xlabel('Time (s)');
ylabel('Spike');
title(sprintf('Spiking Pattern for 10 Neurons %d'));
xlim([0 1]);
%%dropped spikes times
neurons_droppped=neurons;
neurons_drop = find(~neurons_keep);
neurons_droppped(:,:,neurons_drop) = [];
neuron_dropped_times=neuron_spike_times;
neuron_dropped_times(neurons_drop) = [];
%%%ques3
```

```
sigma=3.33*10^-7;%um     0.3333S/m
Voltage_neuron1=sum((currents./(4*3.14*sigma.*distances(1,:)')),1);
Voltage_neuron2=sum((currents./(4*3.14*sigma.*distances(2,:)')),1);
Voltage_neuron3=sum((currents./(4*3.14*sigma.*distances(3,:)')),1);
Voltage_neuron4=sum((currents./(4*3.14*sigma.*distances(4,:)')),1);
Voltage_neuron5=sum((currents./(4*3.14*sigma.*distances(5,:)')),1);
Voltage_neuron6=sum((currents./(4*3.14*sigma.*distances(6,:)')),1);
Voltage_neuron7=sum((currents./(4*3.14*sigma.*distances(7,:)')),1);
Voltage_neuron8=sum((currents./(4*3.14*sigma.*distances(8,:)')),1);
Voltage_neuron9=sum((currents./(4*3.14*sigma.*distances(9,:)')),1);
Voltage_neuron10=sum((currents./(4*3.14*sigma.*distances(10,:)')),1);
voltage_contribution=[Voltage_neuron1;Voltage_neuron2;Voltage_neuron3;Voltage_neuron4;Voltag
e_neuron5;Voltage_neuron6;Voltage_neuron7;Voltage_neuron8;Voltage_neuron9;Voltage_neuron10];
voltage_contribution_dropped=[Voltage_neuron2;Voltage_neuron4;Voltage_neuron6;Voltage_neuron
7;Voltage_neuron8;Voltage_neuron9;Voltage_neuron10];
%%combined signal
simulation_duration = 1; % Simulation duration in seconds
dt = 2.5e-5;
% Generate the time vector
time_vector = 0:dt:simulation_duration-dt;
combined_signal = zeros(1, length(time_vector));
for i = 1:10 % For each of the 10 neurons
   voltage_trace = voltage_contribution(i, :);
   for spike_time = neuron_spike_times{i}
       spike_index = round((spike_time / dt)) + 1;
       % Determining the range of indices in combined_signal where the voltage trace will be
added
       start_idx = spike_index;
       end_idx = min(spike_index + length(voltage_trace) - 1, length(combined_signal));
           % Adjust the length of the voltage trace if it extends beyond the end of
combined_signal
       trace_length = end_idx - start_idx + 1;
                combined_signal(start_idx:end_idx)  =  combined_signal(start_idx:end_idx)  +
voltage_trace(1:trace_length);
   end
end
%%combined signal with dropped
combined_signal_dropped = zeros(1, length(time_vector));
for i = 1:7 % For each of the 10 neurons
   voltage_trace = voltage_contribution_dropped(i, :);
   for spike_time = neuron_dropped_times{i}
       spike_index = round((spike_time / dt)) + 1;
       % Determining the range of indices in combined_signal where the voltage trace will be
added
       start_idx = spike_index;
                       end_idx   =   min(spike_index   +   length(voltage_trace)   -   1,
length(combined_signal_dropped));
           % Adjust the length of the voltage trace if it extends beyond the end of
combined_signal
```

```
        trace_length = end_idx - start_idx + 1;
                                                combined_signal_dropped(start_idx:end_idx)      =
combined_signal_dropped(start_idx:end_idx) + voltage_trace(1:trace_length);
    end
end
%%%%specifically focusing on the reduction of larger spikes
%due to the exclusion of neurons within 25um of the electrode
figure;
subplot(2, 1, 1);
plot(time_vector, combined_signal);
title('Original Combined Signal');
xlabel('Time (s)');
ylabel('Voltage');
subplot(2, 1, 2);
plot(time_vector, combined_signal_dropped);
title('Combined Signal After Dropping Neurons');
xlabel('Time (s)');
ylabel('Voltage');
sgtitle('COMBINED SIGNAL');
%%ques4
%%adding noise to the combined signal
numSamples = round(simulation_duration / dt);
recordedNoise = randn(1, numSamples);
rangeVext = max(combined_signal) - min(combined_signal);
scaledNoise = recordedNoise * (0.1 * rangeVext) / std(recordedNoise);
combined_signal_noise = combined_signal + scaledNoise;
% Plot the combined signal with noise
plot(time_vector,combined_signal_noise);
xlabel('Time (s)');
ylabel('Voltage');
title('Combined Signal with Noise');
```

## Part 3

```
origin=[0,0,0];
neuron1=[XYZ(:,1),XYZ(:,2),XYZ(:,3)];
hilloc_pos=neuron1(200,:);
nwpos=neuron1-[0,50,0];
%%Kmatrix with 1A current
numHeaderLines = 8; % Adjust this number based on the actual number of header lines in your
file.
% Read the file into a table, skipping the header lines
data = readtable('brain_part3.txt', 'HeaderLines', numHeaderLines);
%%units: volatge from comsol is in V, xyz is in m
%%units from current_big: voltage is in     , xyz is in um
x_pos=(data.Var1)*10^6;%um
y_pos=(data.Var2)*10^6;%um
z_pos=(data.Var3)*10^6;%um
electric_potential = data.Var4;
%%we need to interpolate the x,y,z coordonates to match our neuron's
```

```
%%coordinate system to use V
%%we need V for our neuron's coordinates
% Interpolate K matrix data to neuron positions
%  V_interpolated  =  griddata(x_pos,  y_pos,  z_pos,  electric_potential,  neuron1(:,1),
neuron1(:,2), neuron1(:,3), 'cubic');
% V_interpolated(isnan(V_interpolated)) = 0; % Setting NaNs to 0
F = scatteredInterpolant(x_pos, y_pos, z_pos, electric_potential, 'nearest', 'none');
% Use the interpolant to find values at neuron positions
K = F(nwpos(:,1), nwpos(:,2), nwpos(:,3));
K(isnan(K)) = 0; % Setting NaNs to 0
Vext=currents.*K;
%%total voltage at the electrode
Vext_elec=sum(Vext,1);
plot(Vext_elec);
xlabel('Time');
ylabel('Voltage');
title('EXTERNAL VOLTAGE');
```

## Part 4

```
axonLength = 5; % mm
numCompartments = 200; % Number of compartments
current = 1000; %mA
sigma = 0.00033; % S/mm,
radius=10; %um
x_coordinates = linspace(-axonLength / 2, axonLength / 2, numCompartments);
y_coordinates=zeros(1,numCompartments);
z_coordinates=zeros(1,numCompartments);
axon_pos=[x_coordinates;y_coordinates;z_coordinates]';
axon_hilloc=axon_pos(100,:);
%%electrode position is 1mm away from ceneter
electrode=axon_hilloc+[0,1,0];
distances=sqrt((electrode(1)-axon_pos(:,1).^2)+(electrode(2)-axon_pos(:,2).^2)+
(electrode(3)-axon_pos(:,3).^2));
ga=3*10^-5; %S
Vext=current/4*3.14*sigma*distances; %mV
Iint = zeros(size(Vext));
for n = 2:length(Vext) - 1
   Iint(n) = ga * (Vext(n-1) - 2*Vext(n) + Vext(n+1));
end
% Handling boundary conditions for n = 1 and n = 200
Iint(1) = ga * (-2*Vext(1) + Vext(2));
Iint(end) = ga * (Vext(end-1) - 2*Vext(end));
Iint=(Iint);
plot(Iint)
%%ques1 : 40pA to cross nthe threshold
%%ques4:  mA>pA >>> 10^15pA
%>>1A
```