USER MANUAL

# MATLAB CODE

## Bipolar\_Auto.m

The main function to call is ‘**Bipolar\_Auto.m**’ in MATLAB. The code is divided into six sections, with the last five sections nested in a for-loop.

The first section is to load the gridded data from COMSOL, specify the current amplitude and ratio of the anodic current when bipolar configuration is used. The ranges of the coordinates in the regular grid are as follows: X=-2500:20:2500 µm, Y=-750:5:750 µm and Z=-750:5:750 µm. neg\_el is a handle for the grid of the cathodic electrode, whereas pos\_el is for the anodic electrode. There are 12 electrodes, hence 12 gridded data files, named **‘circularONx.txt’** (x=A to L; for the position of each electrode see **‘ELECTRODE CENTROID.xls’**). currents takes a vector of current amplitudes in µA, and neg\_ratio and pos\_ratio are factors that are multiplied with the current amplitude to define the total injected current for each electrode. The combinations of the elements in currents and pos\_ratio will be simulated in loop in the next part.

neg\_el=load('circularONJ.txt');

pos\_el=load('CircularONI.txt');

currents=[15];

% %Prescribe ratio

neg\_ratio=[1];

pos\_ratio=[0 0.2 0.5 1];

The second section is the multiplication and superposition (in the case of bipolar configuration) of the loaded potential grids. The potential values from each grid (in mV) are multiplied with the current specified and ratio. The superposed grid is called comb\_mult.

The third section is to determine the initial sampling area. Firstly, the code finds the location where the absolute potential value from the superposed grid is (el\_loc), and the initial sampling resolution d in µm is defined. d is 40 in most of the spatial probability maps that have been generated. In the next step, el\_loc becomes the centre point for the initial sampling area calculation. A while-loop is called, where the function ‘**makebox.m**’ computes eight points around el\_loc, forming a square. The distance of these points from the centre point are determined by the value n. Then, the firing probability of these points are computed by calling the function ‘**testfunct\_three.m**’, which contains the instruction to run the NEURON .hoc code. In the case where the point falls outside the grid, the probability is assumed to be 0. If the probability values of the eight points are all 0, the while-loop stops running. If not, n value is increased, so that a bigger square is formed and the probability values are computed for the new points. The output of this process is Yinit and Zinit, containing the initial Y and Z coordinates that enclose the initial sampling area. The variable coordgives a table with all the Y-Z coordinate pairs where the probability is to be computed. Yinit and Zinit are saved to the disk.

file\_name=strcat(string(int16(100\*neg\_ratio(iter))),'\_',string(int16(100\*neg\_ratio(iter))),'\_', string(int16(current)));

Yinit=Yinit';

fname=strcat('YinitJI ',file\_name);

fname=char(fname);

save(fname,'Yinit');

Zinit=Zinit';

fname=strcat('ZinitJI ',file\_name);

fname=char(fname);

save(fname,'Zinit');

The fourth section generates the map with the initial sampling resolution (for most of the maps it is 40µmx40µm), evaluated at the Y-Z coordinates defined in coord. As MATLAB sometimes crashes while running the code, checkpoints are used to save the probability values to a column matrix called axonProbabilitytemp. The matrix is saved at 30%, 50%, 70%, 90% and 100% of points in coord have been computed.

Next, the probability values in axonProbabilitytemp are then reshaped into a 2D matrix that corresponds to the initial sampling coordinates in coord. The new matrix is called axonProbability, and it is saved to the disk. Another matrix called combine contains the coordinates of the points that have been computed so far, together with the probability values. It is also saved to the disk.

The fifth section finds the coordinates where the points in the axonProbability matrix have to be refined, based on the difference of a point’s probability with that of the neighbouring points. When the difference is above 70%, 4 new sampling points will be added around that centre point, with a distance of d/2. The output is the matrix newPointswhich contains the points that have to be calculated for the refinement step. The probability value at the coordinates specified in newPoints are computed by calling **‘testfunct\_three.m’** again. Checkpoints are used in the computation of the new points, and they are saved in an intermediate matrix called axprobtemp, and renamed to axprobwhen finished.

The sixth section is the interpolation steps to get the final map. First, the matrix axonProbability is interpolated to double its resolution to d/2, the resulting matrix is called vm2.

x5=min(Yinit):d/2:max(Yinit);

x6=min(Zinit):d/2:max(Zinit);

vm2=zeros(numel(x6),numel(x5));

for k=1:numel(x6)

for l=1:numel(x5)

vm2(k,l)=interpn(Zinit,Yinit,axonProbability,x6(k),x5(l),'linear');

end

end

Then, the additional points that have been computed in the refinement process (axprob) replace the interpolated value in the vm2 matrix. It is then saved to the disk.

for k=1:length(axprob)

vm2(floor((newZ(k)-min(Zinit))/(d/2))+1,floor((newY(k)-min(Yinit))/(d/2))+1)=axprob(k);

end

Lastly, the final interpolation to 1 µm x1µm pixel size is done, and the matrix is padded with zero to give a uniform size of the potential map when necessary. The points that are located outside of the optic nerve are cut out by using *roipoly* function, where the coordinates of the boundary to cut out are obtained by loading **‘xpos.mat’** and **‘ypos.mat’**. The final map should have a dimension of 1501 x 1501. The final map is saved to the disk with variable name vmnew.

# File Naming

There are several intermediate matrices that this code generate, and a naming convention for the matrix files is used. The file name consists of: variable name, cathodic electrode code, anodic electrode code, cathodic current ratio, anodic current ratio, current amplitude. For example, the variable Yinit for electrodes J (cathodic) and I (anodic), with 100% cathodic current and 50% anodic current at 15 µA is saved as **’YinitJI100\_50\_15.mat’**.

## Testfunct\_three.m

This function is divided into two sections, the first one calls a neuron .hoc files called main3 and the second one calls another .hoc file main2. Two optimisation steps for NEURON computation are implemented, one in this .m code, and another one in main2.

The fibre parameters, including the fibre diameters and the fixed node length are specified in this code. Then, the remaining parameters are calculated based on the fibre diameters using **fiberParameters** function. The dimension of the optic nerve are defined here (longAxis, shortAxis, totalLength). The vector percentage\_shift defines the longitudinal shift tested (0-0.5). The values in this vector will be multiplied with the node-to-node distance to define the distance of the nodes of Ranvier from the electrode in the longitudinal axis. The fibre parameters (fibre diameter, node-to-node distance and node diameter) that have been computed are saved to .txt files, which will be accessed by the NEURON code.

save par\_fibDiam.txt fiberDiameter -ascii

save par\_ntnlengths.txt nodeToNodeDistance -ascii

save par\_ndiam.txt nodeDiameter -ascii

The fiber diameter distribution data should be defined. The variable is called fiberProb.

The next step is to collect all the potential values at the given Y-Z coordinate, at all available X-coordinates from the potential grid. This is done by calling the function getFiberValues. After that, a nested loop to run the NEURON simulation for each fiber geometry and longitudinal shift is added. An optimisation step is done, by testing only the smallest and largest fiber diameters (defined as i=1,12), at 0 and 0.5 shift (n=1,6). The .hoc code run is main3,which runs a NEURON computation to solve the cable equation for one fibre geometry at one shift value. For each fibre diameter, the locations of the nodes of Ranvier and the number of nodes are computed. The number of nodes are also saved in a text file called **‘par\_nnodes.txt’**. The fibre distribution data for each fibre diameter is also given here, in a variable called fiberProb. This distribution data is taken from Vaney (1976).

fiberProb = [1.56,13.2,27.5,21.4,13.125,8.61,5.43,3.51,1.91,1.28,0.59,0.48] ;

Interpolation of potential values are done to get the potential at the specified locations of the nodes of Ranvier using linear interpolation. Then, main3 is called to generate run.dat file. The value of run is 1 if the axon is activated and 0 otherwise. This value is then collected in an M x N matrix tempProb, where M is the number of fibre diameters and N is the number of shifts.

Based on the first four combinations tested, the code determines whether or not to continue the computation. If all four combinations result in the same value (either all 0 or all 1) the computation is stopped, and all the other combinations are assumed to have the same value. If this is not the case, the code looks at the two combinations at the same shift. If these two values are equal, all the remaining combinations in that specific shift is assumed to have the same value. If not, they will have to be computed with the remaining combinations.

To solve for the remaining combinations, the function main2 is used. This function is similar to main3, but with an optimisation step already implemented in the .hoc code. This function will return a vector containing the computation results for all the fibre geometries in one shift. Similar to the optimisation step using main3, main2 tests the first (smallest), then the last (largest) fibre diameter. When the two produce different run values, the second fibre diameter is simulated. The computation is continued for the third, fourth, … diameter until the run value changes. After the change in run value is found, the remaining fibre diameters are not simulated, but assumed to produce the same value as the last simulation.

## Mainx.hoc

This code is the main .hoc code to run. To plot the membrane voltage, run this function from File Explorer. From the NEURON main menu, click Graph- Voltage axis. To run it from the NEURON GUI, main32.hoc needs the command batchrun(simulation time, stimulus amplitude, fibre number), while main22.hoc only needs batchrun(simulation time, stimulus amplitude), as it will compute for all the fibre diameters, albeit with optimisation step. As the current amplitude has been applied in the **Bipolar\_Auto.m** function, always keep the stimulus amplitude when running batchrun to 1. It is also important to note that in hoc the indexing starts from 0. Hence, to plot the voltage of the first fibre in NEURON GUI, fibre number should be specified as 0 instead of 1.

Mainx.hoc calls several functions, including:

Param\_reading.hoc : Load the fibre parameters that have been computed in MATLAB to build the nerve fibre

Geometry.hoc: Build the axon geometry using the fibre parameters specified

Axon\_template.hoc: The template of the axon biophysical and geometrical properties. Ion channels are defined in separate .mod files, but they are inserted to the axon in this code.

Setup.hoc: Set the time step, simulation time, etc. It also calls the function stimulation.hoc.

Stimulation.hoc: Apply the extracellular potential from COMSOL to the nodes of Ranvier. The stimulus waveform can be defined here. Right now, we have different versions of stimulation.hoc code. stimulation.hoc has a monophasic cathodic pulse, and stimulationX.hoc, where X=1,2,3 or 4, has biphasic pulse, with X denoting the number of pulses in the stimulus train. This can be compacted to one code that takes X as a local variable to define the number of pulses, rather than having separate codes for each number of stimulus pulse.

To change the parameters of the stimulus, change the parameters DEL (delay of stimulus onset), DUR (duration of the cathodic pulse) and DURP (duration of the anodic pulse). These parameters are located in mainx.hoc.