12-2018

Modeling and simulation methodologies for spinal cord stimulation.

Saliya Kumara Kirigeeganage
University of Louisville

MODELING AND SIMULATION METHODOLOGIES FOR SPINAL CORD STIMULATION

By
Saliya Kumara Kirigeeganage
B.Sc., University of Peradeniya, Sri Lanka, 2011
M.Sc., University of Louisville, Kentucky, USA, 2013

A Dissertation
Submitted to the Faculty of the
J. B. Speed School of Engineering of the University of Louisville
in Partial Fulfillment of the Requirements
for the Degree of

Doctor of Philosophy
in Electrical Engineering

Department of Electrical and Computer Engineering
University of Louisville
Louisville, Kentucky

December 2018

MODELING AND SIMULATION METHODOLOGIES FOR SPINAL CORD STIMULATION

By

Saliya Kumara Kirigeeganage
B.Sc., University of Peradeniya, Sri Lanka, 2011
M.Sc., University of Louisville, Kentucky, USA, 2013

Dissertation approved on

November 28, 2018

by the following dissertation Committee:

Dissertation Director
Dr. John Naber

Dr. Robert Keynton

Dr. Kevin Walsh

Dr. Jacek Zurada

Dr. Thomas Roussel

To my parents & Shash

ACKNOWLEDGMENTS

Nothing in life comes easy, without hurdles. So was this dissertation. However, there were many people around me who made this difficult task possible.

First and foremost, I would like to thank Dr. John Naber for being my mentor and making me a part of his team where I got unparalleled opportunities to work with different technologies and products. Thank you for spending countless hours reading and commenting on drafts of my work. Your guidance and support throughout the last few years not only made this dissertation possible, but also gave me the opportunity to work on outstanding electronic projects. I was able to gain a lot experiences in embedded system design and software applications, because of you.

I am also grateful for my colleague Doug Jackson for all his valuable comments and support over the last five years. I am fortunate enough to have worked with you side by side. You are one of the smartest people I have ever met in my life. Your knowledge about everything and your experience in engineering, inspire me every day. Thank you so much for being so willing to share your knowledge. I will always remember the conversations we had. I will miss working with you the most.

Thank you Dr. Kevin Walsh for lending me some of your computer equipment to run my simulations. If it was not for your generous support, I might still be running simulations. I am grateful for you Dr. Jacek Zurada for making room for me in your busy schedule to read my drafts and give me -constructive comments to improve my work. Thank you Dr. Robert Keynton and Dr. Thomas Roussel for your invaluable comments and all the corrections you had made on my writing. A huge shout out to Dr. Indika Wijayasinghe for giving me new ideas and pointing me toward machine-

learning. I am grateful for you for showing me the way when I was lost in the world of optimization.

Lastly, I could not have achieved my goals without the support of my wonderful family. I thank my parents for teaching me the value of education and hard work. Without all your sweat and tears, I could not have been able to travel across the world to live my dream. I am especially thankful for my loving wife Shash, for sticking by me and sacrificing so much, so that I could achieve this goal. You have always been there for me every step of the way giving me the strength I needed.

ABSTRACT

MODELING AND SIMULATION METHODOLOGIES FOR SPINAL CORD STIMULATION

Saliya Kumara Kirigeeganage

November 28, 2018

The use of neural prostheses to improve health of paraplegics has been a prime interest of neuroscientists over the last few decades. Scientists have performed experiments with spinal cord stimulation (SCS) to enable voluntary motor function of paralyzed patients. However, the experimentation on the human spinal cord is not a trivial task. Therefore, modeling and simulation techniques play a significant role in understanding the underlying concepts and mechanics of the spinal cord stimulation. In this work, simulation and modeling techniques related to spinal cord stimulation were investigated.

The initial work was intended to visualize the electric field distribution patterns in the spinal cord. A system consisting a set of stimulating electrodes with a model of the spinal cord was built and simulations were performed by applying different stimuli to the electrodes. Depending on the complexity of the model, the simulation times can be varying. However, the system demonstrated the ability to visualize the field distributions inside the spinal cord for static and transient stimuli. This data can be used to aid experimental studies and to better understand the results of the spinal cord stimulation by mapping known experimental results to the simulation results.

The stimulation of biological tissue raises safety concerns. Therefore, a methodology utilizes in electrical stimulation of biological systems known as charge balancing was studied. The importance of charge balancing and the effects of using charge balancing was studied with simulation studies. The simulation results showed extensive charge accumulation in the biological tissue, around the stimulation sites, when charge balancing was not utilized. Hence, the studies showed the importance of using charge balancing and the importance of avoiding pulse collisions. Stimulation sequences that utilizes different stimuli with different frequencies results in a phenomenon known as pulse collision. Pulse collisions cause complications in charge balancing and should be avoided or reduced in stimulation procedures. An algorithm and a software based on the algorithm was developed to reduce pulse collisions by optimizing pulse positioning. The tool showed significant reduction in pulse collisions for simulation sequences that were used in experimental settings. The current version of the software is capable of optimizing up to five different pulses.

Coupling the electric fields and action potentials generated in neurons due to the electric fields can lead to significant discoveries on the mechanics of the SCS. Therefore, the investigation of bi-domain models has valuable implications. Bi-domain models were built by combining different neural models with electric field results. The simulations proved the possibility of visualizing the action potentials in neurons, when the spinal cord was stimulated using electrical pulses. Having the optimized neural models that simulate the behavior of the motor neurons can aid the experimental studies by easily identifying the proper stimulation sequences to activate motor neurons.

The final section of this work was focused on using machine learning in neuroscience for optimization purposes. Well known Hodgkin-Huxley (HH) model and a recently published Izhikevich mathematical model were used for this task. The HH model is a widely used, biophysically meaningful model that can simulate the action

potentials in the nerve axons. It is mainly used to simulate the type 2 behavior of the axon firing. However, other types of neuron spiking and bursting have been observed in the literature. This work demonstrates the optimization of the HH model parameters to simulate neuron bursting behavior. The results of the study demonstrate that it is possible to extend the HH model beyond its intended type 2 behavior and can be modified to simulate more complex neuron firing patterns including neuron bursting. The optimized HH model was able to generate bursting patterns corresponding to the two target bursting patterns used in this study with error values $< \mathbf{18}\%$ for phasic bursting and $< \mathbf{6}\%$ for tonic bursting.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Background

For decades man and machine shared a symbiotic relationship for the betterment of each other. This union is becoming more and more common as the technology advances. For the most part, machines have been assisting humans to improve their lifestyle and quality of life. However, improving health and curing illnesses using machines is an area of active research. Today, having machines implanted inside the human body to heal an injury or to cure an illness is not fiction.

Neural prosthetics is an area of research that focuses solely on creating interfaces to open communication with machines and the nervous system. Neural prostheses[1] have been used extensively in the last five decades to restore some neural functions by selectively stimulating neurons. The first known implant, a cochlear implant was made in 1957. Other pioneering inventions in this area include the development of the first internal pacemaker[2] in 1958 and the first motor prosthesis for foot drop in a hemiplegic in 1961. Further, the research conducted in the late 80's and early 90's proved that the functional electrical stimulation (FES)[3] allowed paraplegics to walk and the lumbar anterior root implant facilitated standing [1].

[1] Devices that are capable of restoring functions lost due to an injury or neural damage.

[2] "A pacemaker is a small device that's placed in the chest or abdomen to help control abnormal heart rhythms." `https://www.nhlbi.nih.gov/health-topics/pacemakers`

[3] Application of small electrical pulses to paralyzed muscles to restore or improve their function

Figure 1.1. Typical neural prosthesis used in SCS. On *left* a spinal cord stimulator from Medtronic© and on *right* a set of electrode arrays from Boston Scientific©.

In 2001, it was recorded that since 1963 over 40000 neural prostheses have been implanted [2, 3]. These prostheses can either be spinal cord implants (SCI) or deep brain implants (DBI). A typical neural prosthesis used in spinal cord stimulation (SCS)[4] is shown in Fig. 1.1. Common research that utilize these prostheses are in the areas of restoration of hearing, improvement of bladder and bowel functions and relieving of chronic pain [4, 5, 6, 7].

Another active field of research that has a great importance in health care is the treatment of spinal cord injuries. Spinal cord injury is a significant cause of paralysis. Out of 5.6 million cases of paralysis recorded in 2009, 23% are attributed to spinal cord injuries (*refer* Fig. 1.2). Records show that there are more than 1.2 million people with spinal cord injuries in the United States [8]. Patients with spinal cord injuries confront complications when performing basic life activities. Depending on the severity of the injury these patients could loose the ability to walk, stand and move their arms. These injuries could often lead to secondary complications in bowl, bladder and sexual functions.

Recent studies have shown that SCS can be utilized to enable voluntary motor

[4]A therapy that provides pain relief by blocking pain signals before they reach the brain

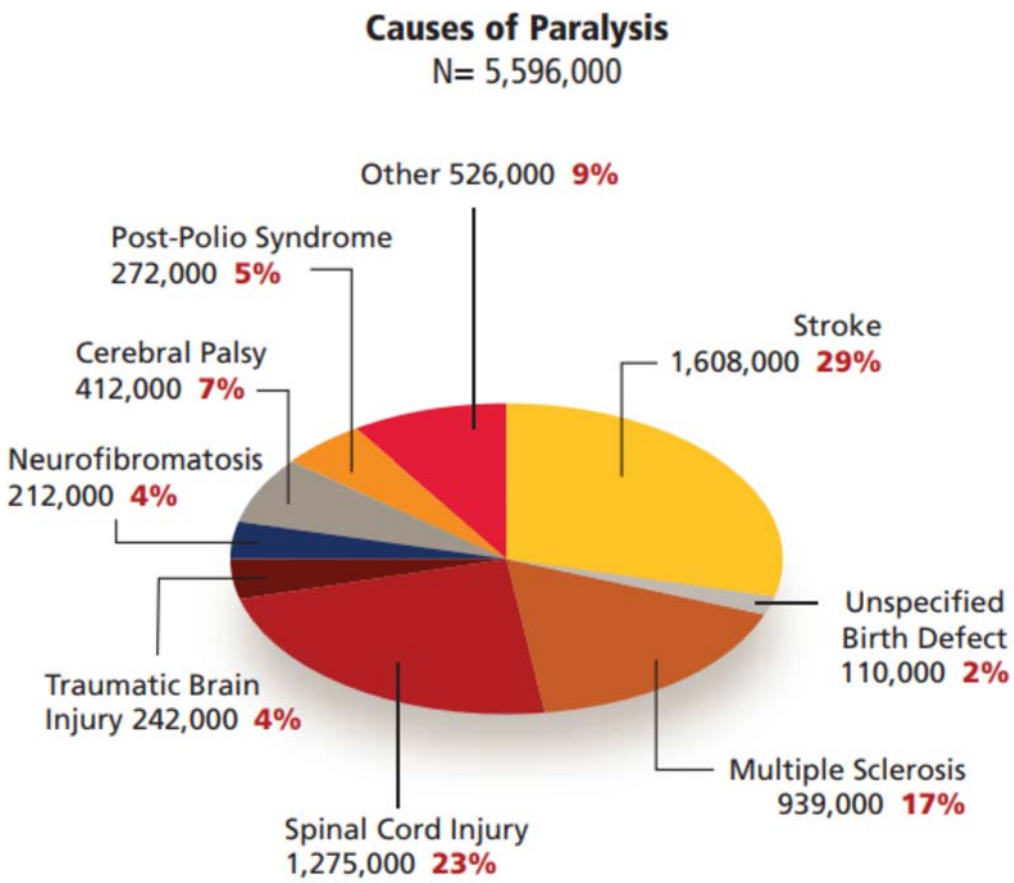


Figure 1.2. Causes of paralysis according to a study conducted by Reeve Foundation, in a cooperative agreement with the U.S. Centers for Disease Control and Prevention (CDC)[8].

functions in patients with spinal cord injuries [9, 10]. In a study performed at the Frazier rehab center at the University of Louisville's Kentucky Spinal Cord Injury Research Center, four individuals received SCS and all four regained the ability to independently stand and acquire some voluntary control of the toes, ankles, knees and hips [11, 12].

Even though there has been breakthroughs and improvements in the way of using SCIs to treat patients, the underlying mechanisms of the SCS is not well understood. For instance, it is known that the SCS can enable voluntary motor functions in paraplegics. However, a certain stimuli that enabled or restored a certain functionality in an individual might not do the same for a second individual. Therefore, the efficacy

Figure 1.3. The first four individuals who received SCS at the Frazier rehab center at the University of Louisville's Kentucky Spinal Cord Injury Research Center. Left to right: Andrew Meas, Dustin Shillcox, Kent Stephenson and Rob Summers. *Photo: University of Louisville, KY.*

of SCS when treating paraplegic or hemiplegic patients rely heavily on trial and error approaches. These approaches can be time consuming and unreliable at times.

Due to some of the drawbacks of SCS and the insufficient knowledge of underlying mechanisms, doctors and researchers are actively trying to better understand the science of SCS. One of the ways to do this is via simulation studies.

1.2 Possible Simulation and Modeling Studies

Electrical stimulation of the spinal cord and the central nervous system in general has been used for decades with clinical perspectives as well as for fundamental research [13, 14, 15]. Despite the widespread use and the groundbreaking results the physics and the underlying mechanisms of the electrical stimulation is far form being under-

stood. To overcome experimental limitations and better understand the electrical stimulation, modeling approaches have been used since the 80's [16, 17]. Simulation studies can model various components of SCS system and help understand the connections between the external stimuli and the internal mechanisms of the body. Therefore, various aspects of SCS can be identified for the purposes of simulations. The simulations can include electric field distribution studies, ion and charge distribution studies under stimulation and cellular level simulations to study the behavior of the neurons. Details of these studies are described in the following chapters.

CHAPTER 2

PRELIMINARY SIMULATION STUDIES

As mentioned in the previous chapter, a variety of simulation studies can be performed to simulate different aspects of the SCS.

2.1 Electric Field Simulation

In the early stages of this study, one of the main objectives was to observe the behavior of the electric fields in the spinal cord when it was stimulated with external electrical stimuli. In experiments performed on patients with spinal cord injuries, the external stimuli were applied to the spinal cord using a 16 electrode array[1] manufactured by Medtronic[2]. The goal was to find stimulus patterns and threshold intensities that would be used to stimulate certain parts of the spinal cord. By stimulating specific regions of the spinal cord, the deactivated or damaged neural paths could be activated to re-enable the communication between the brain and the different parts of the human body [9]. The same experimental procedure was needed to be simulated in a computational environment to replicate the experimental results and better understand the physics and the chemistry behind SCS.

Since the simulations involved solving mathematical equations in 3D space, it was determined to use a specialized tool to perform the simulations. Therefore it

[1]Electrode array which was used in the patient trials mentioned in chapter 1
[2]Medtronic is one of the world's largest medical device manufacturing companies[18]

was decided to use the commercially available COMOSL Multiphysics® [3] simulation software to perform this study. Over the course of this study, version 4.4, 5.0 and 5.1 of COMSOL have been used. COMSOL is based on the computational technique finite element method (FEM). FEM utilizes a piecewise approximation technique where the bounded domain of interest is divided in to finite number of non-overlapping elements. Depending on the number of spatial dimensions of the study, these elements can take different shapes. Usually they take triangular or rectangular shapes for 2D studies and tetrahedral or hexahedral for 3D studies. Once these elements are defined, the functions that are of interest to the study are approximated by trial functions. Usually the trial functions are chosen to be polynomials. The boundary conditions are applied along the local and global boundaries. The result is an approximation of the variational form of the system by a linear combination of a finite set of trial functions.

COMSOL requires a model setup before the system can be solved to extract the desired results. Typical work flow of a COMSOL simulation is as follows.

1. Creating the model
2. Applying materials
3. Applying Physics
4. Generating a mesh
5. Solving the problem
6. Extracting the results

The following sections summarize the information on above steps that were used to solve the electric field distributions in the spinal cord.

[3] One of the leading software tools, that is used in finite element analysis and multi-physics simulations

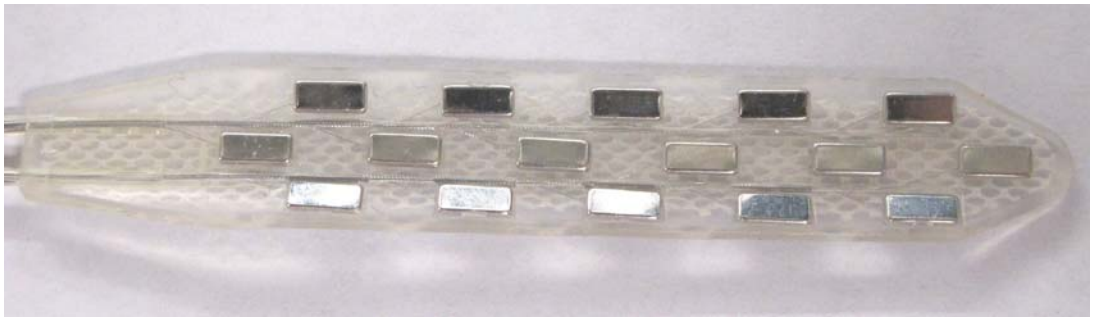


Figure 2.1. Medtronic 16 electrode array

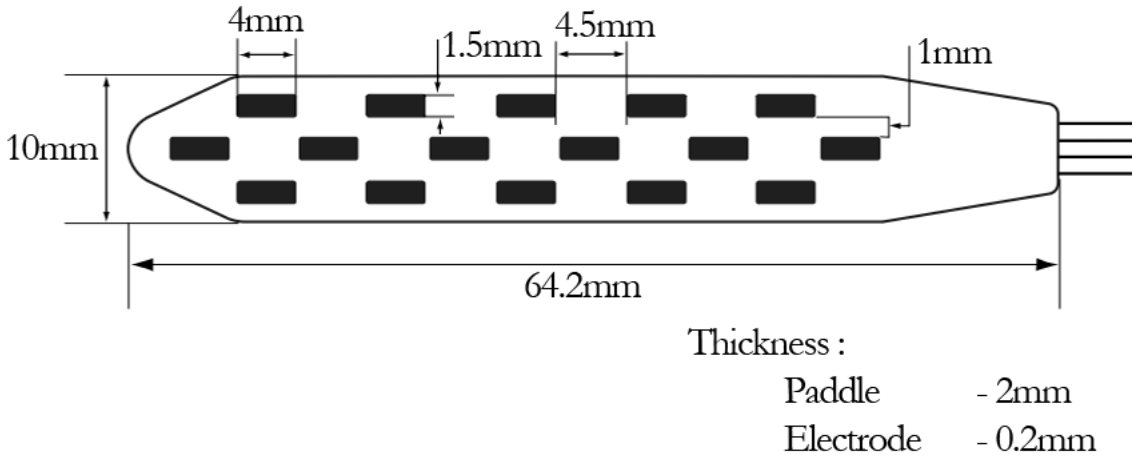


Figure 2.2. Medtronic 16 electrode array dimensions

2.1.1 Creating the Model Geometry

The Medtronic 16 electrode array (fig. 2.1) consists of 16 Platinum - Iridium electrodes arranged in 3 rows (5 - 6 - 5 arrangement) and a Silicone rubber paddle on which the electrodes are embedded. To support the structure of the paddle, there exists a polyester mesh inside the Silicone rubber. Due to the complexity of the structure, it was decided to create the model using SolidWorks rather than the built-in model builder inside COMSOL.

Materials used and the dimensions of the array were taken from the Medtronic implant manual[19]. Dimensions of the arrays are shown in the figure 2.2.

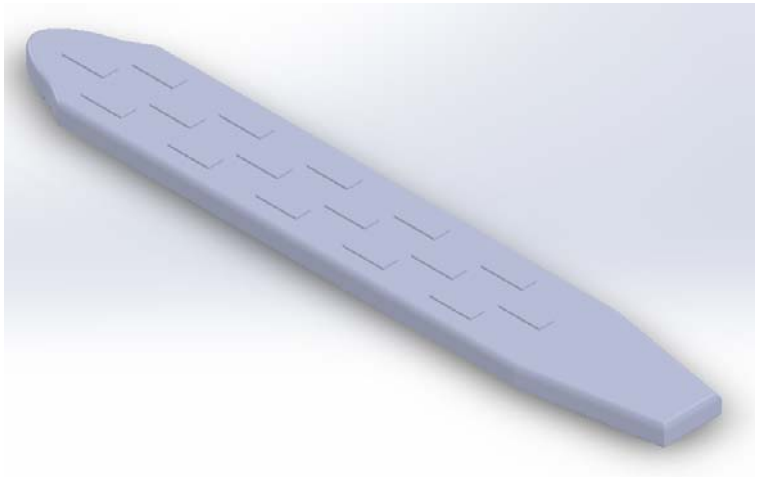
Using the actual dimensions obtained from the Medtronic implant manual, models with the exact scale were built in SolidWorks. Actual electrodes in the array have soft

and curved edges and the paddle contains a slight bend along the center electrode row. These features were accounted for in the model making. However, the soft edges on the electrodes and on the paddle causes the mesh[4] to be extremely fine when it comes to mesh the model inside COMSOL. Although an extremely fine mesh can be generated in COMSOL, the resulting mesh causes the computational time to be increased significantly. Therefore, it was decided to remove the soft edges in the model and to keep rough edges in the final model to decrease the complexity of the mesh and to reduce the computational time. Furthermore, the paddle plays a key role in retaining the structure of the actual array. However, it does not affect[5] the simulation results. Therefore the paddle can also be removed from the final model. By having less geometry the simulation time can be decreased further. Different variations of the model that were created in SolidWorks are shown in the figure 2.3

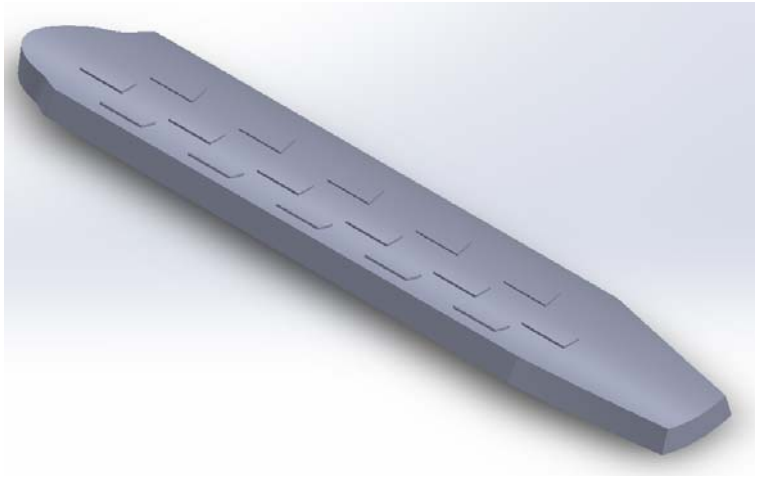
For the reasons mentioned in the previous paragraph, it was decided to use the model that contains only the electrodes in the final simulations (Figure 2.3c). The model of the electrode array is only a part of the final model that would be used in the simulations. A piece of geometry that would represent the spinal cord is also needed. The spinal cord is a complex structure that is made up of different sections that possess different material properties. However for initial electric field simulations, it was decided to use a simple tubular structure to represent the spinal cord. The final model that consists of the electrode array and the tubular structure to represent the spinal cord is shown in figure 2.4.

[4]Meshing is the process of dividing the domain of interest in to a finite number of non-overlapping elements as described in the previous section

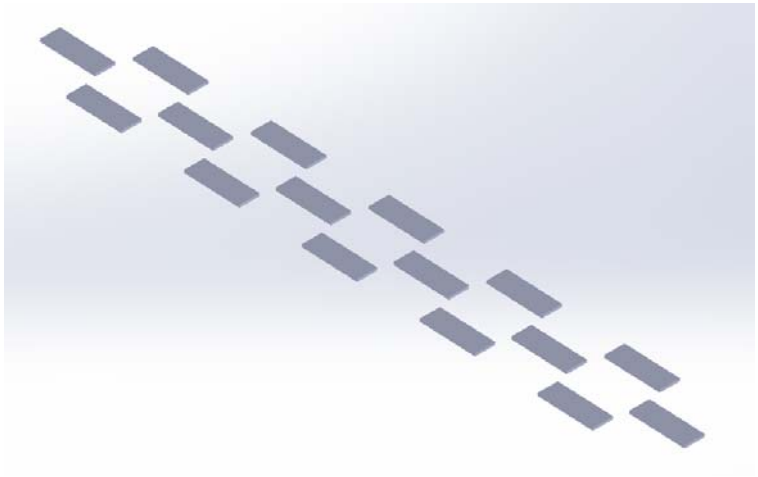
[5]Although the Silicone rubber in the array has a different permittivity value from the other materials in the model and causes the electric fields to change, we are interested in the electric fields above the electrodes rather than the electric fields in the direction of the paddle



(a) Model with soft edges in the electrodes and the paddle



(b) Model with the curved paddle



(c) Model that contains only electrodes

Figure 2.3. Different 3D models of the Medtronic array created in SolidWorks

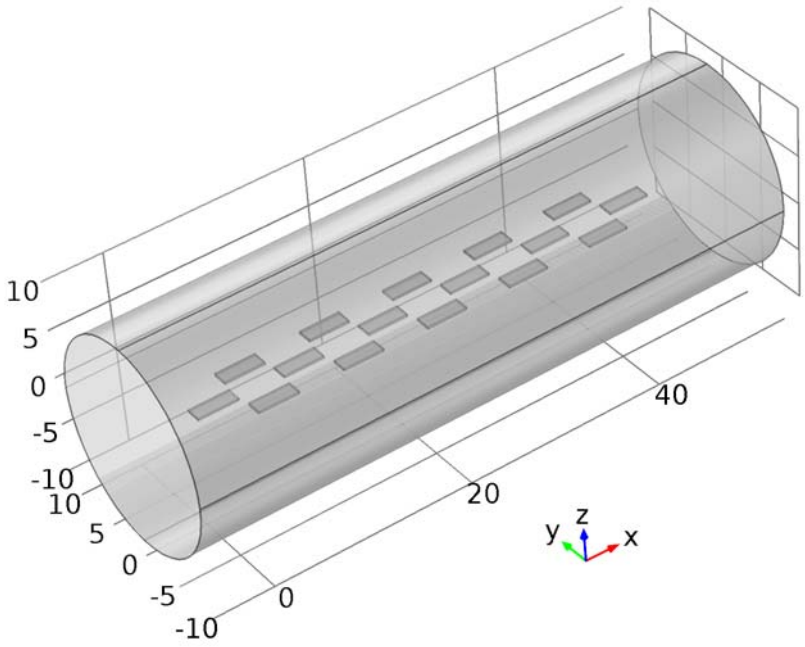


Figure 2.4. Final model used for the simulation

2.1.2 Materials, Physics and Mesh

Once a model geometry is created for the simulation study, materials must be applied to the geometry to provide the necessary material parameters that would be used in solving the equations of the system.

Since the model was simplified to improve the computational efficiency by reducing the computational time, the final model used in the simulation contains only two types of materials. One material that represents the electrodes and one material that represents the environment that surrounds the electrode array.

According to the Medtronic implant manual, the electrodes are coated with an alloy made out of Platinum and Iridium. The material properties of this alloy varies depending on the Platinum to Iridium ratio used for the alloy. The exact ratio of the alloy was not provided and the the values of the material properties for this alloy was not found in the literature. Further, the environment in which the electrodes were embedded was simplified to a single material, the exact material properties of the spinal cord could not be used. In the laboratory experiments performed with the

electrode array, typically this environment was a Saline solution. However, for the purposes of this simulation the properties of this environment can be any value that is biologically meaningful and plausible. Therefore, for the simulation, the properties of stainless steel and the properties of blood were used for the electrodes and the surrounding environment respectively. Specific material properties that were used for the electromagnetic simulations are listed in the table 2.1.

Table 2.1. Material Properties of Stainless Steel and Blood[20]

	Permittivity, ϵ	Conductivity, σ (S/m)
Stainless Steel	1×10^{15}	1.35×10^{6}
Blood	1.2×10^{3}	0.7

Once the materials were assigned to the model geometry, the next step is to add the necessary physics modules to the model. Physics modules inside COMSOL must be chosen according to the type of the simulation that is required to obtain the desired results. For instance, in this case, the requirement is to obtain the electric filed distribution patterns inside the spinal cord or the environment in which the electrode array is embedded. For this specific case, COMSOL provides two physics modules, Electrostatics (es) and Electric Currents (ec). Once these modules were employed in the model, necessary boundary conditions must be applied. In this study, simulations were performed such that some electrodes were stimulated with voltage pulses while some electrodes were grounded and some electrodes remain floating (high impedance)

COMSOL provides variety of presets to create the mesh for the model. However, depending on the complexity of the model making a custom mesh may be beneficial in cases. Although the geometry is not very complex in this study a custom generated mesh was used because of the size variation of the model geometry. A very fine mesh was used for the electrodes and a coarse mesh was used for the tubular surrounding environment. Figure 2.5 shows the mesh that was generated in this study.

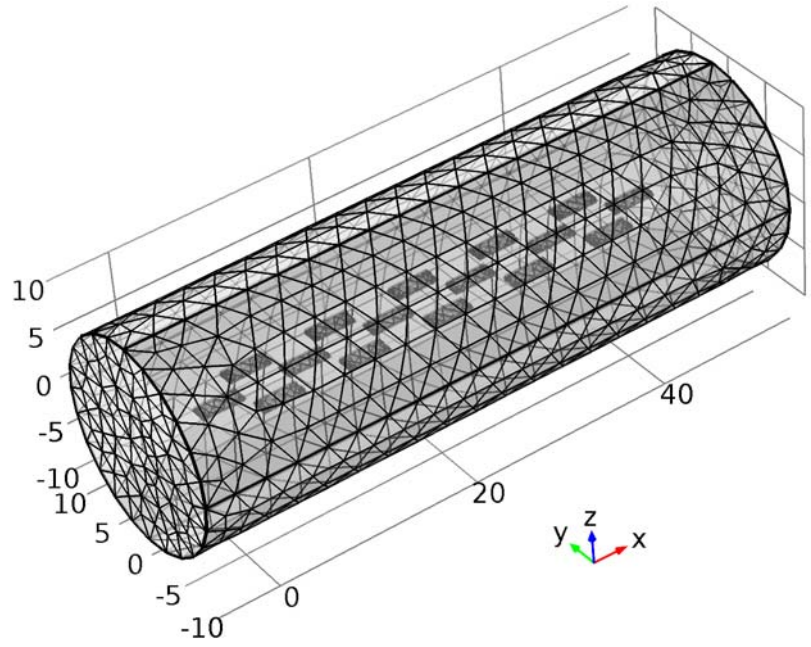


Figure 2.5. Mesh of the Final model

2.1.3 Simulating the Model

2.1.3.1 Simulating Using Test Pulses

The system can be simulated once the materials and physics were set and the model geometry was meshed. The duration for the simulations can vary depending on various factors. Therefore it is always a good idea to simulate the system for a simplistic case before simulating using the actual simulation parameters. Performing a test simulation is also useful in verifying the setup of the simulation as well as estimating the simulation time. For this purpose, two test pulses were first defined. COMSOL provides different mathematical functions to create stimuli with different shapes. The built-in function `flc2hs` was used to create the test pulses. The table 2.2 shows the exact expressions used to define this test pulses under COMSOL's global definitions section.

Once the pulses have been defined, they were applied to two electrode configurations identified as the 'maximum current configurations' and two separate simulations

Table 2.2. Expressions used to generate pulses for test simulation

Pulse	Expression
Anode	$2 * (flc2hs(t + 0.1, 0.01) - flc2hs(t - 0.2, 0.01) - exp(-5 * (t)) * flc2hs(t - 0.4, 0.01) * 8)$
Cathode	$-2 * (flc2hs(t + 0.1, 0.01) - flc2hs(t - 0.2, 0.01) - exp(-5 * (t)) * flc2hs(t - 0.4, 0.01) * 8)$

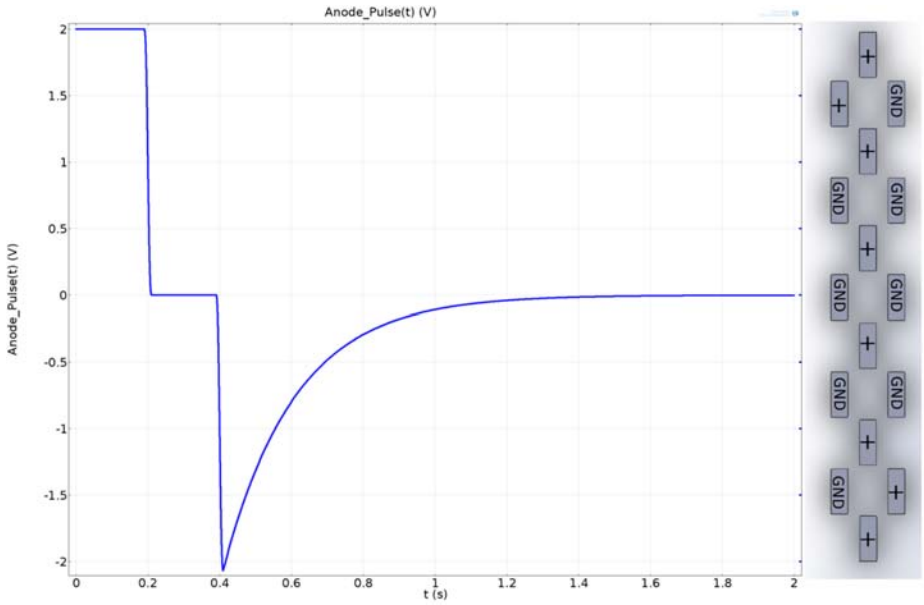
were performed. Figure 2.6 shows the two pulses and the electrode configurations to which they were applied.

The boundary conditions for these two simulations were set such that the electrodes marked as GND were grounded and the electrodes marked as + and − were stimulated using the Anode and Cathode pulses respectively.

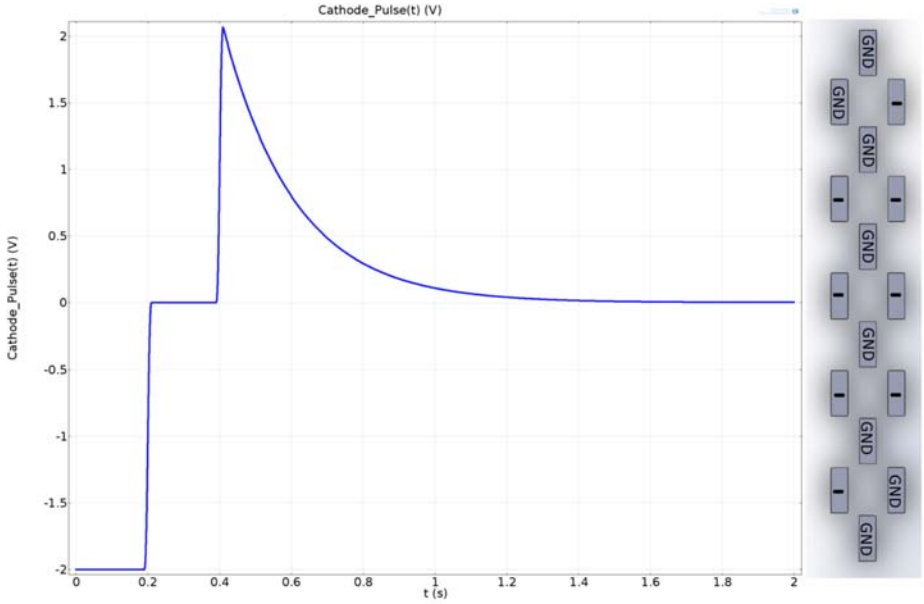
2.1.3.2 Simulating Using Experimentally Used Pulses

Once the model setup is verified using the test simulation, the second simulation was performed using the data from the experimentally used simulation pulses. In this case, rather than applying a single pulse to multiple electrodes, a combination of pulses, high impedance and ground conditions were applied to electrodes according to a scheme used in the actual experimental SCS studies. The scheme used in this simulation is one of many schemes that are used in experiments and is identified as a "program". A *program* defines the transient boundary conditions applied to each electrode in the 16 electrode array at certain time intervals. The *program* used in this simulation is shown in the figure 2.7. The R in the figure indicates the return or the ground (GND) boundary condition while the X indicates high impedance or floating boundary condition.

To better understand the boundary conditions applied to a single electrode, consider boundary conditions applied to electrode 1 (Figure.2.7a). At $t = 0$ the electrode must be stimulated with $2.5V$ pulse. The pulse contains a positive voltage for a certain amount of time and a negative voltage for different amount of time. This type of

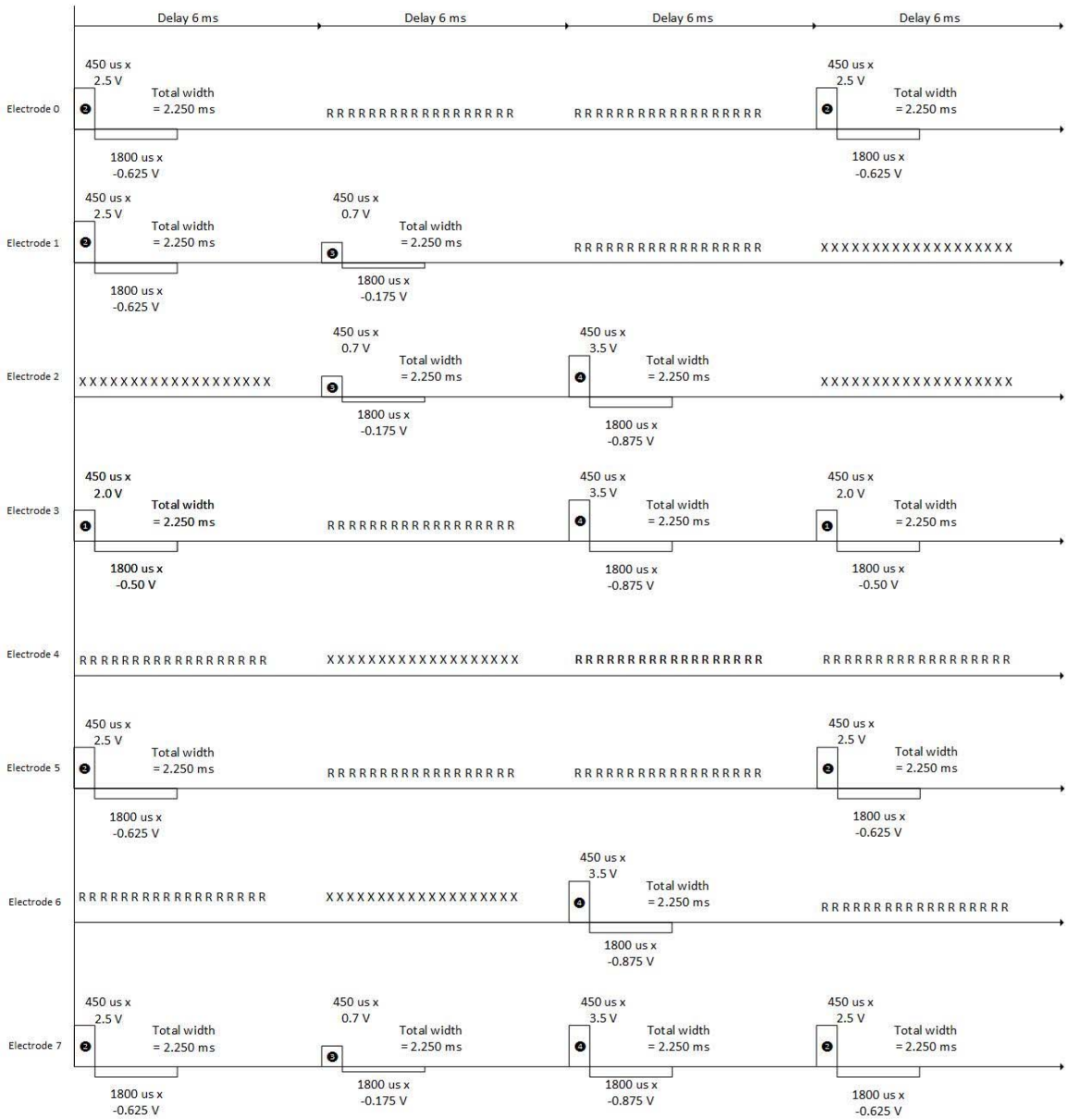


(a) Anode pulse and the used electrode configuration

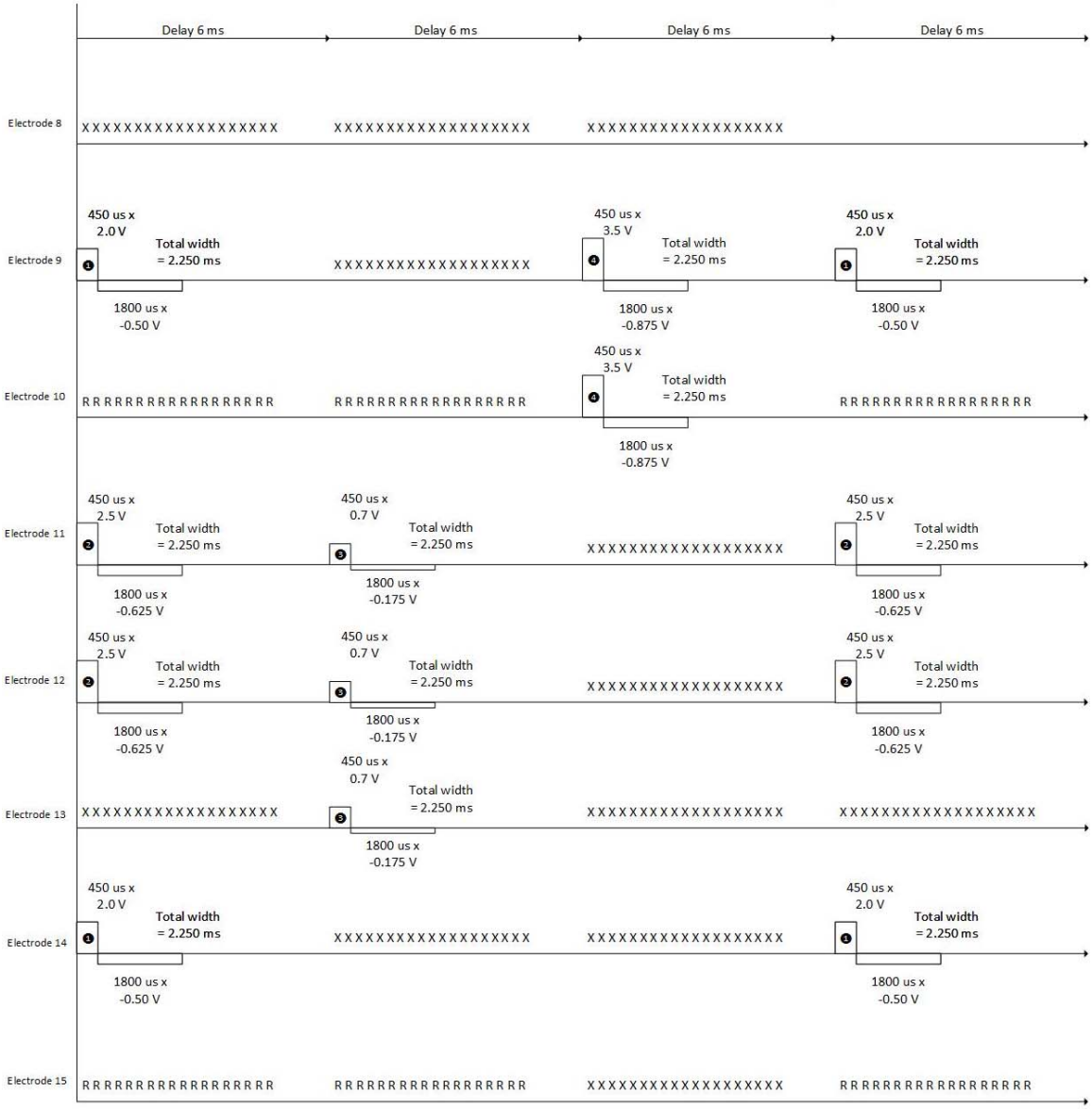


(b) cathode pulse and the used electrode configuration

Figure 2.6. Two pulses and the electrode configurations



(a) Waveforms used for electrodes 0 to 7



(b) Waveforms used for electrodes 8 to 15

Figure 2.7. The *program* used in the simulation

a pulse is called a charge balanced voltage pulse[6]. After a $6ms$ has passed since the application of the fist pulse a $0.7V$ pulse must be applied. Finally, the boundary conditions of the electrode must be set to ground(GND) and floating mode respectively at the given time markers.

The variations in the boundary conditions for a given electrode raised a question in simulating the entire simulation range ($24ms$) as a whole, because COMSOL did not have the capability to handle a voltage boundary condition and a floating boundary condition for a given domain in a single simulation setup. Therefore it was decided to break the simulation into four sets where each set simulates a period of $6ms$ of the total simulation. This way each electrode can take either a voltage boundary condition or a floating boundary condition according the figure 2.7 within the selected set.

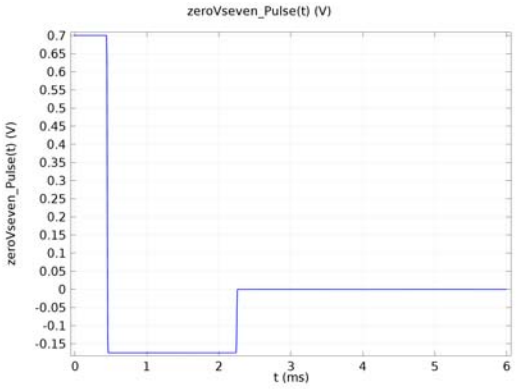
It can be seen by observing the figure 2.7 that there are 6 different boundary conditions. These include four voltage pulses ($0.7V$ pulse, $2.0V$ pulse, $2.5V$ pulse and $3.5V$ pulse) a ground and a floating condition. The four voltage pulses were defined under COMSOL's global definitions section and added to the corresponding model domains using the `Terminal Boundary Condition` option in COMSOL. The expressions used to generate four different voltage pulses are shown in the table 2.3. The voltage pulses generated using the expressions shown in the table 2.3 are shown in the figure 2.8.

Using the pulse boundary conditions shown in table 2.3 as well as ground and floating boundary conditions, 4 simulations were performed. Since the four separate simulations represented a breakdown of a single large simulation, the continuity between individual simulations had to be preserved. In order to do this, the last set of results of the first transient simulation was used as the initial condition of the second transient simulation. The same approach was taken for the 3rd and 4th simulation

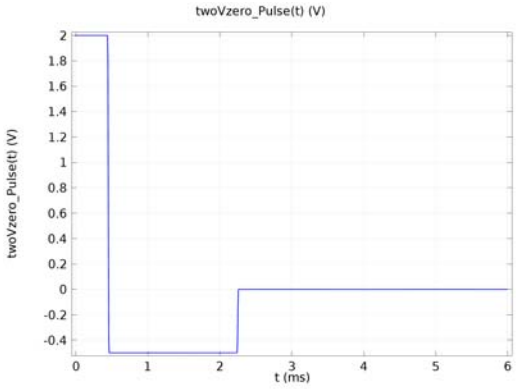
[6]More on charge balancing and charged balanced pulses are described in the section 2.2

Table 2.3. Expressions used to generate four voltage pulses

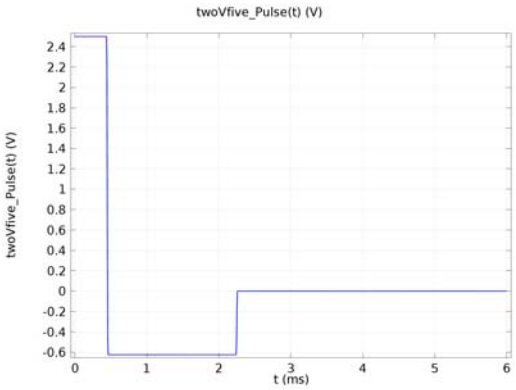
Pulse	Expression
$0.7V$	$0.7*flc2hs(t+0.01, 0.01)-0.875*flc2hs(t-0.45, 0.01)+0.175*flc2hs(t-2.25, 0.01)$
$2.0V$	$2.0*flc2hs(t+0.01, 0.01)-2.500*flc2hs(t-0.45, 0.01)+0.500*flc2hs(t-2.25, 0.01)$
$2.5V$	$2.5*flc2hs(t+0.01, 0.01)-3.125*flc2hs(t-0.45, 0.01)+0.625*flc2hs(t-2.25, 0.01)$
$3.5V$	$3.5*flc2hs(t+0.01, 0.01)-4.375*flc2hs(t-0.45, 0.01)+0.875*flc2hs(t-2.25, 0.01)$



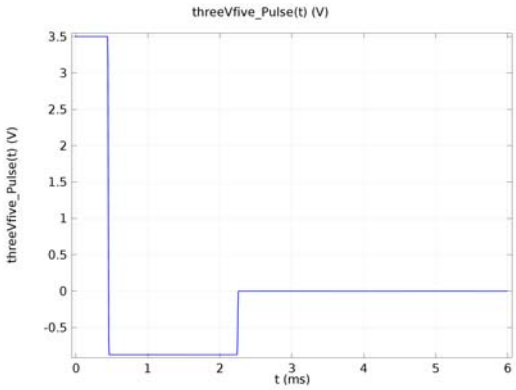
(a) $0.7V$ pulse



(b) $2.0V$ pulse



(c) $2.5V$ pulse



(d) $3.5V$ pulse

Figure 2.8. Voltage pulses generated using the expressions in the table 2.3

sets as shown in equation (2.1).

$$t1_{initial} = 0 \tag{2.1a}$$

$$t2_{initial} = t1_{end} \tag{2.1b}$$

$$t3_{initial} = t2_{end} \tag{2.1c}$$

$$t4_{initial} = t3_{end} \tag{2.1d}$$

2.1.4 Results

Both of the simulations mentioned in the previous section were transient simulations. This implies that the results from the simulations were functions of time. Because the resolution of the each simulation was set to about 4 frames per millisecond, a large number of result frames were extracted at specific time intervals. Therefore, it is not convenient to illustrate the results from each time frame, but only the most relevant results are discussed in this chapter.

2.1.4.1 Results - Simulation of a Test Pulse

Simulations were performed for an anode pulse with a positive voltage and a negative charge balancing section and for a cathode pulse with a negative voltage and a positive charge balancing section. Although charge balancing is crucial in stimulating a biological system it is not intended to affect the results of the stimulation. Therefore, only the results obtained in the stimulation section of the pulses are illustrated here.

Figure 2.9 shows the voltage distribution along the surface of the electrodes. Although the actual magnitudes of the potentials are different, the potential difference remains the same between anode and cathode pulse simulations. This indicates that there are two possible ways to stimulate a system to obtain the same exact result. Figure 2.10 shows the magnitudes and directions of electric fields $1mm$ above and parallel to the surface of the electrodes. Slight differences in the magnitudes of the

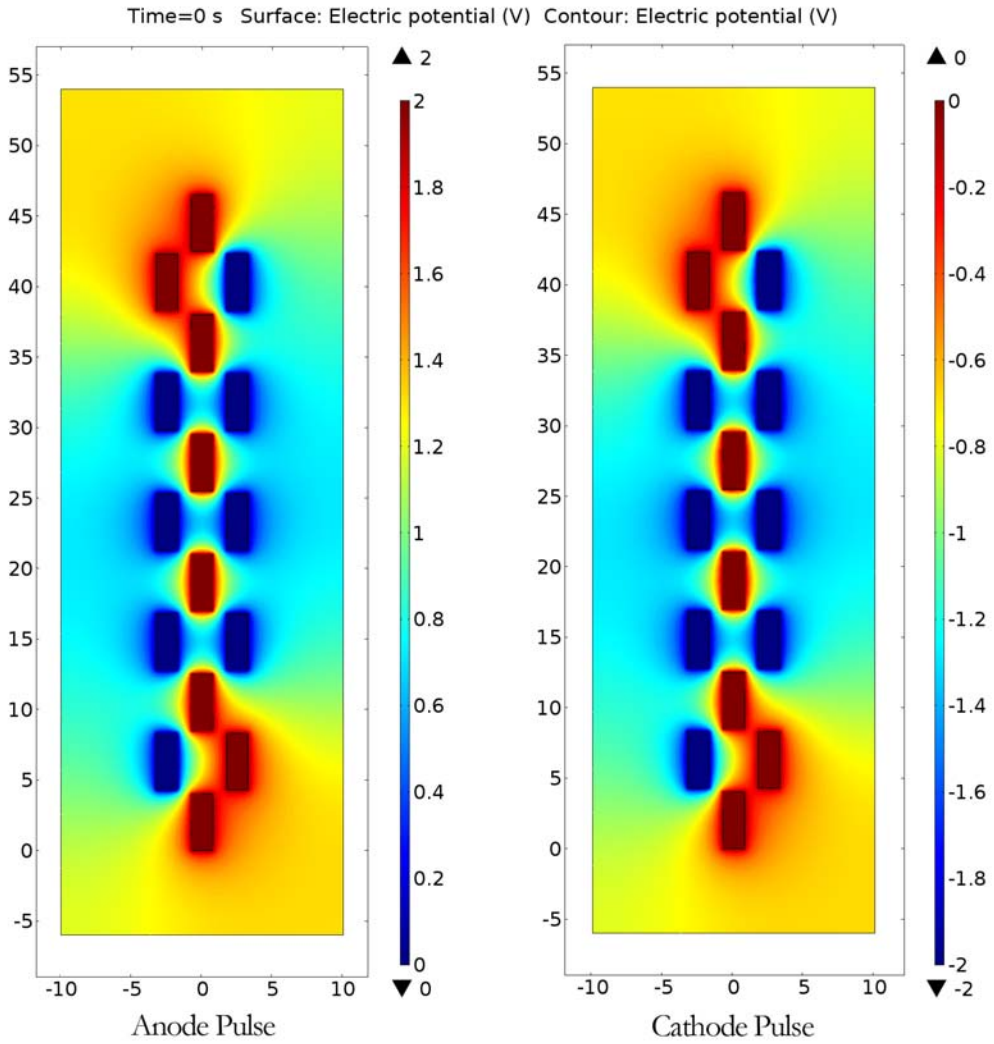


Figure 2.9. Potential distribution along the surface of the electrodes

electric fields are observed, yet the directions of the electric fields are identical for both cases.

2.1.4.2 Results - Simulation of Experimentally Used Pulses

As discussed in the previous section, the actual stimuli considered in this simulation consisted of 16 different waveforms supplied to the electrodes in a period of $24ms$. To avoid the complications in setting up the boundary conditions of the simulation, the full sequence was divided in to four $6ms$ sections. Each of these four quarters

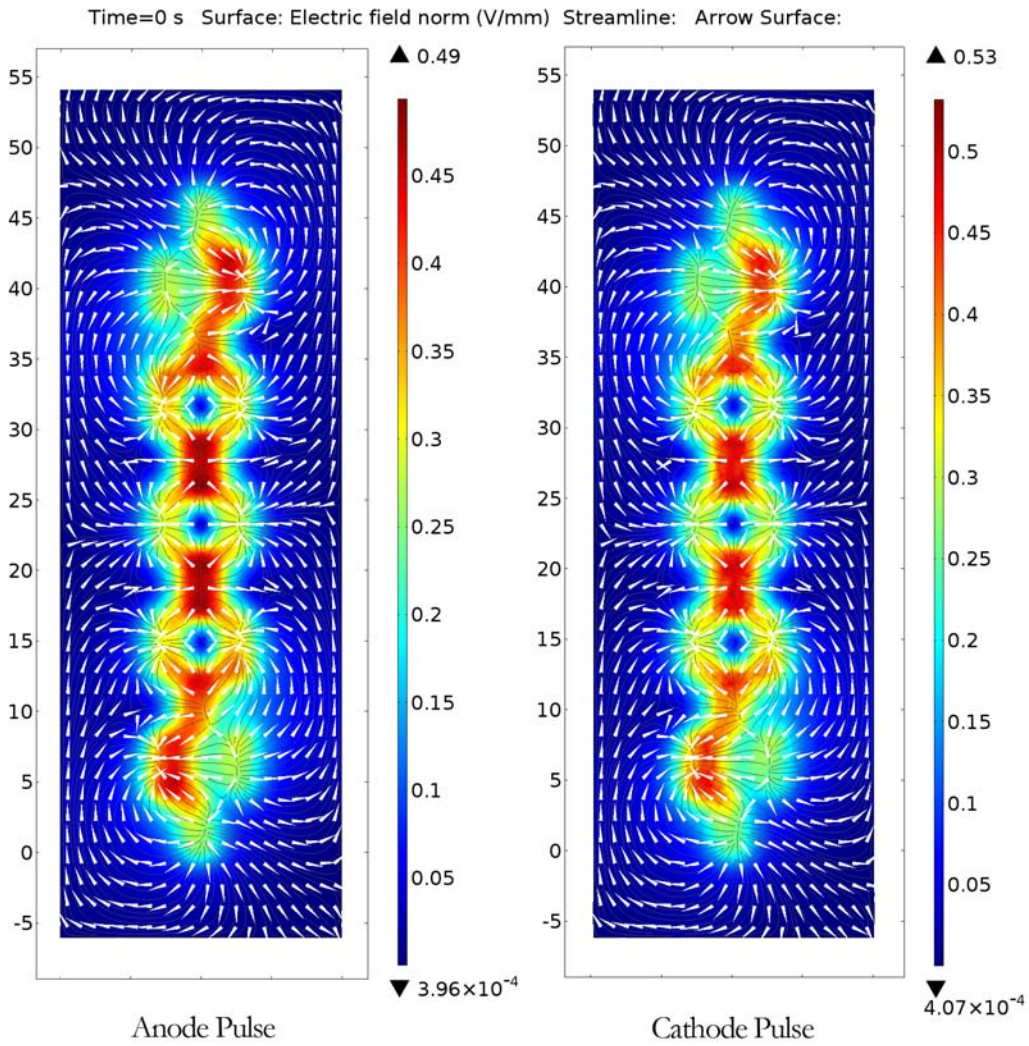


Figure 2.10. Magnitudes and directions of electric fields - 1mm above and parallel to the electrodes

contained 16 waveform signals intended for the 16 electrodes[7]. Considering the pulse boundary conditions applied to the system, the first $450\mu s$ of each quarter was the stimulation section while the next $2250\mu s$ was the charge balancing section. The remaining $3300\mu s$ was a relaxation period. Hence the most important section in each quarter was the first $450\mu s$. Therefore the results extracted in this $450\mu s$ period in each quarter are illustrated here.

The potential distributions and vertical cross sections of the electric fields are

[7]Review figure 2.7a and figure 2.7b

illustrated here. Each figure shows the results extracted at $t = 240\mu s$ in each quarter. All four plots in a given figure refers to the same scale shown in the right hand side of the figure while the two numbers on top and bottom of each plot denotes the maximum and minimum values recorded at each quarter. x and y axes in the figures denotes the spacial dimensions in millimeters.

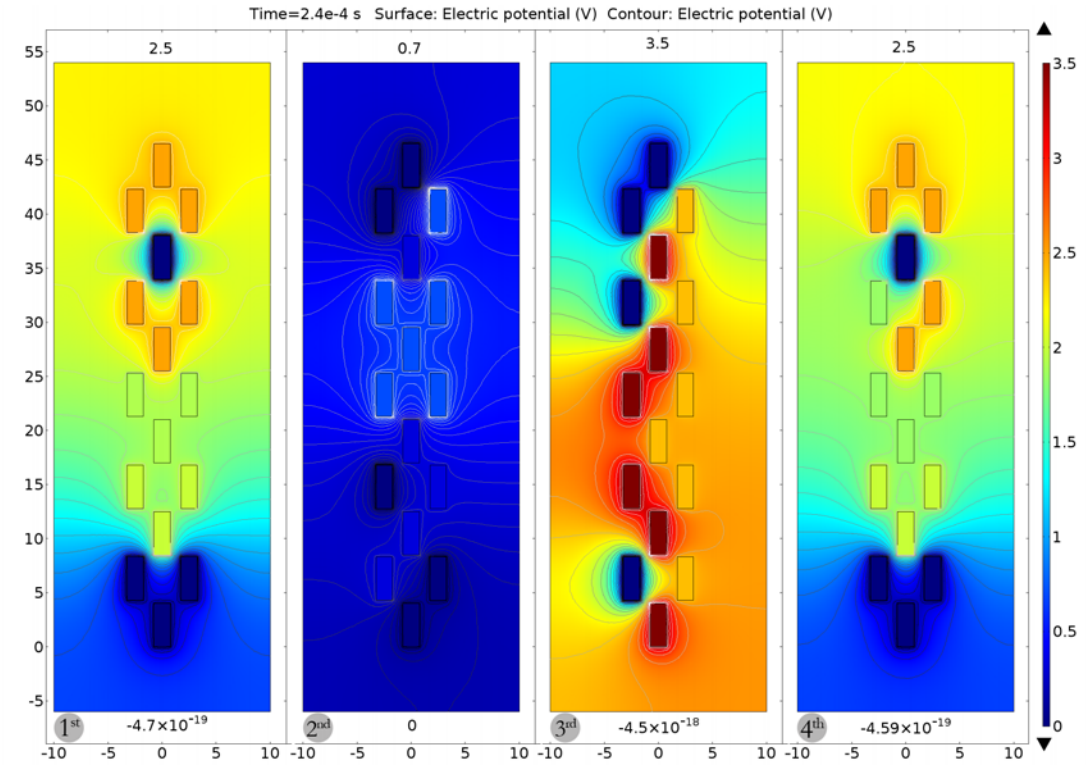


Figure 2.11. Potential distribution along the surface of the electrodes

Figure 2.11 shows the potential distributions along the surface of the electrodes. Minimum values in the 1st, 3rd and 4th quarters are shown to be a really small value instead of zero. However these values are practically zero. Figure 2.12 shows the directions and magnitudes of electric fields, $1mm$ above and parallel to the surface of the electrodes. Since all four plots refers to the same scale, the scale had to be adjusted in such a way that the results in all four quarters could be visualized properly. Due to this, the regions that contain values above the maximum value in the scale are shown in a highly saturated maroon color.

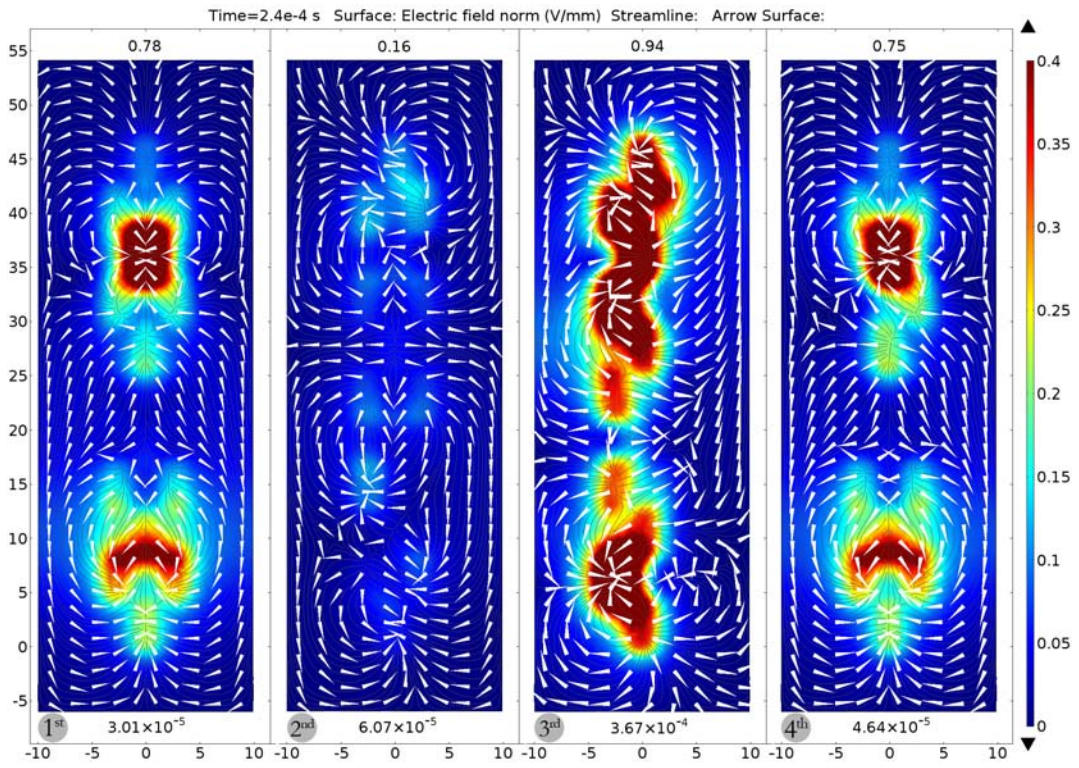


Figure 2.12. Magnitudes and directions of electric fields - 1mm above and parallel to the electrodes

2.2 Charge Balancing and Electro-Chemistry Simulations

2.2.1 Charge Balancing at the Electrode-Tissue Interface

The primary objective of neural stimulation is to activate excitable tissue where there is a neural dysfunction and is accomplished by transferring charge over a conducting electrode into nerve tissue. This process may induce oxidation reduction reactions at the electrode-tissue interface and could lead to electrode and tissue damage. Therefore it is important to prevent or reduce this damages by minimizing the charge accumulation around the stimulation interface.

There are two primary mechanisms of charge transfer at the electrode-tissue interface. One is the non-Faradaic reaction where no electrons are transferred between the electrode and the tissue. The other is the Faradaic reaction where electrons are transferred between the electrode and the tissue at the interface[21]. In the non-

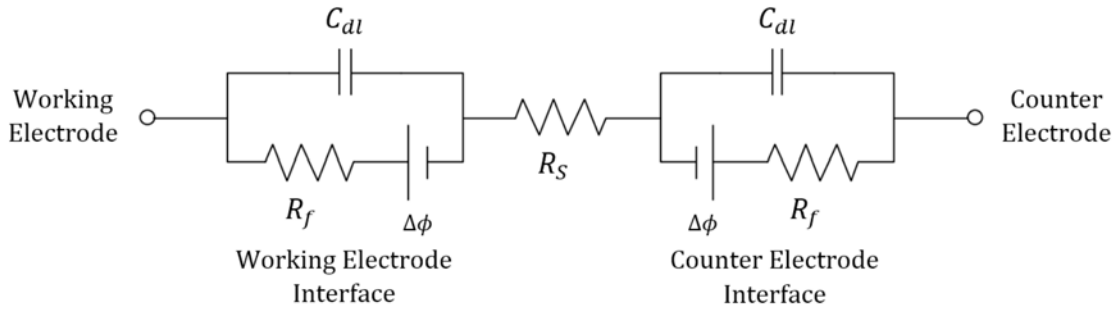


Figure 2.13. Two electrode model of the electrode-tissue interface[22]

Faradaic mechanism, the electrode that is stimulated with a positive voltage attracts negatively charged ions while the electrode that is stimulated with a negative voltage attracts positively charged ions. Therefore an electrical current is induced due to the movement of the ions. Because of this characteristic behavior the interface can be modeled as an electrical capacitor known as the double layer capacitor, C_{dl}. On the other hand, in Faradaic mechanism, electron removal and electron addition to the system occur at the positively driven and negatively driven electrodes respectively.

To successfully stimulate a biological system at least two electrodes are needed to complete a current path from one electrode to the other, where one electrode would stay at a relatively positive voltage while the other stay at a relatively negative voltage. These two electrodes and the electrode-tissue interface where non-Faradaic and Faradaic processes occur can be modeled in an electrical circuit as shown in figure 2.13. In the model shown in the figure 2.13 C_{dl} is the double layer capacitance which represents the non-Faradaic process, while R_f is the Faradaic resistance that represents the charge transfer between the electrode and the tissue. R_s is the resistance of the tissue and $\Delta\phi$ is defined as the equilibrium potential between the electrode and the tissue. The value of $\Delta\phi$ is typically small and can be neglected for simplicity.

Since there is no external charge injection to the system the non-Faradaic process is reversible and harmless. However, the Faradaic process can be harmless and reversible or harmful and non reversible. Therefore it is important to ensure that the

Faradaic process is reversible during the stimulation. There are several approaches that can be used to ensure the reversibility of the Faradaic process by preventing or reducing the charge accumulation at the electrode-tissue interface.

There are few common approaches when it comes to applying stimuli in the neural stimulation. The most common approach is to apply a constant stimulation current for a certain time. This approach is known as constant current stimulation (CCS). The other openings are voltage controlled stimulation (VCS) and constant charge or switched capacitor stimulation[23, 24, 25]. VCS method is the simplest design which is considered to be efficient in stimulation but lacks the control over the charge being injected to the tissue since it is dependent on the impedance of the electrode-tissue interface [26]. The impedance of the electrode-tissue interface has been shown to fluctuate during stimulation[27, 28, 29]. On the other hand the constant charge method has better controllability, however the capacitors that needs to be used are very costly, very large and would take a considerable amount of space in a circuit.

Regardless of the method used the Faradaic process must be minimized. The most common way of doing this is to maintain a zero potential difference across the C_{dl} capacitor after each stimulation cycle. The technique to achieve this task is known as charge balancing. There are various methods to achieve charge balancing in biological systems. One of the simplest ways of performing this is to utilize biphasic stimulation pulses as the stimuli to the electrodes. Different variations of biphasic current waveforms are possible and shown in the figure 2.14[30]. I_c, I_a, t_c, t_a and t_{ip} are the cathodic current, the anodic current, the cathodic half-phase period, the anodic half-phase period and the inter phase delay respectively.

In order to balance out the charge accumulation after stimulation, each of the pulses has to meet a certain criteria. For symmetric and asymmetric biphasic pulses this criterion is to satisfy the equation (2.2a) while for monophasic the equation that needs to be satisfied is (2.2b).

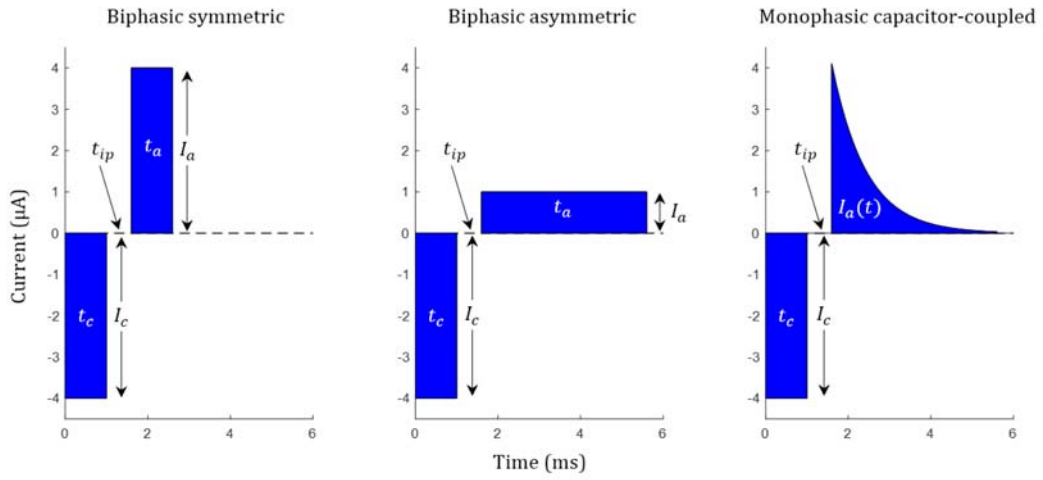


Figure 2.14. Different bi-phasic current pulses that can be used in neural stimulation. The parameters can vary widely depending on the application and the system

$$I_c \times t_c = I_a \times t_a \tag{2.2a}$$

$$I_c \times t_c = \int I_a(t)dt \tag{2.2b}$$

2.2.2 Simulation of Charge Balancing Using FEM

The fundamentals of the double layer capacitance and the charge balancing discussed in the previous section can be simulated using a FEM analysis. COMSOL multiphyics was used for this simulation as it provides an electro-chemistry module that supports the physics of double layer capacitance and charge balancing.

Since this was a proof-of-concept simulation to observe the charge transport through a medium when a stimulus was applied to the medium via an electrode, a simplified geometry was used. As it was described in section 2.2.1, a minimum of two electrodes are required to perform a stimulation by completing a current path. Therefore, a simple geometry with two steel electrodes stimulating in a saline solution was considered.

The physics of the simulation was to solve for the charge transport properties in the solution under the influences of both ionic concentration gradient created by

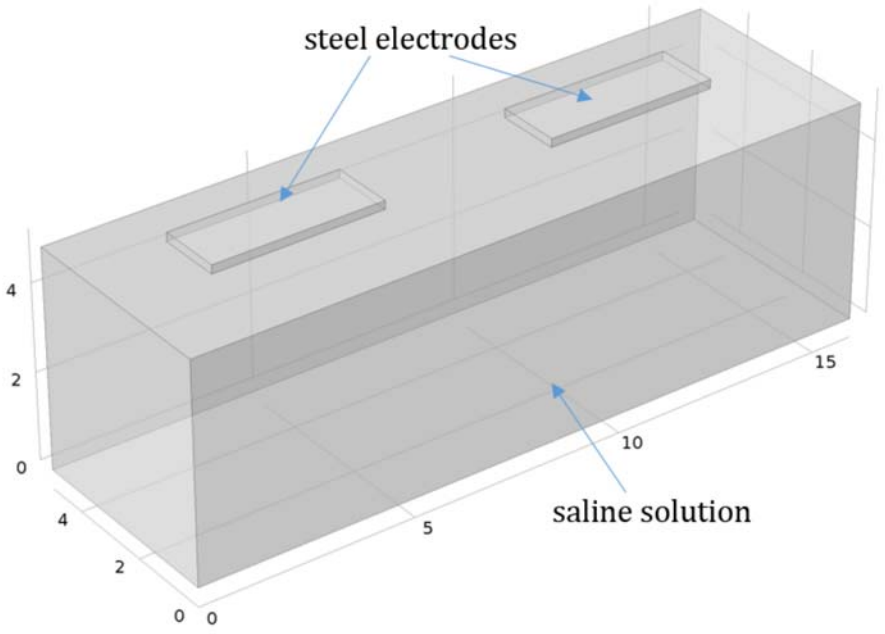


Figure 2.15. Geometry used for electro-chemistry and charge balancing simulation

the ions in the solution and an electric field created by the external stimulus applied through the two electrodes. Dilute species (ions with low concentrations) obey the Nernst-Planck equation (2.3) for mass transport[31, 32].

$$N_i = -D_i \nabla c_i - u_{m,i} z_i F c_i \nabla \phi \tag{2.3}$$

The Nernst-Planck equation (2.3) describes the fluxes (N_i, $mol \cdot m^{-2} s^{-1}$) of two ions with opposite charge in the solution where D_i ($m^2 s^{-1}$) is the diffusion coefficient, $u_{m,i}$ ($s \cdot mol \cdot Kg^{-1}$) is the mobility, z_i is the charge number, F ($C \cdot mol^{-1}$) is the Faraday's constant, c_i ($mol \cdot m^{-3}$) is the concentration and ϕ (V) is the potential.

Assuming that there exists no heterogeneous reactions in the solution, the governing equation for the two ions in the steady state can be written as (2.4).

$$\frac{\partial c_i}{\partial t} + \nabla \cdot N_i = 0 \tag{2.4}$$

The electric fields obey Gauss's law (2.5), where E ($V \cdot m^{-1}$) is the electric field,

ρ $(C \cdot m^{-3})$ is the space charge density, ϵ_s is the permittivity of the solution and ϵ is the permittivity of the free space.

$$\nabla \cdot E = \frac{\rho}{\epsilon_s \epsilon} \tag{2.5a}$$

$$\rho = F \sum_i z_i c_i \tag{2.5b}$$

When the two equations (2.3) and (2.5) are combined the final equation, the Nernst-Planck-Poisson equation (2.6) for the system can be obtained[33, 34].

$$\nabla^2 \phi + \frac{F}{\epsilon_s \epsilon} \sum_i z_i c_i = 0 \tag{2.6}$$

Sodium and Chloride was considered to be the two ions for this study. The diffusion coefficients for the ions in a saline solution depends on the salinity as well as the temperature of the solution. The diffusion coefficients of a solution with salinity, $S = 34.864$ and the temperature, $T = 25°C$ was considered (see Table 2.4). Table 2.5 shows the material properties used for the simulations.

Table 2.4. Diffusion coefficients of Na^+ and Cl^- ions in a saline solution with salinity, $S = 34.864$ and temperature, $T = 25°C$ [35].

	Cl^-	Na^+
D_i $(10^{-10}\ m^2 s^{-1})$	17.71	12.12

Table 2.5. Material properties for steel and saline used in the simulations

	Steel	Saline
Electrical conductivity $(S \cdot m^{-1})$	1.35×10^6	0.7
Relative permittivity	1×10^{15}	1.2×10^3
Electrolyte conductivity $(S \cdot m^{-1})$	N/A	0.7

Once the physics and the material properties were set, two simulations were performed where one of the electrodes was grounded while the other electrode was stimulated with a current pulse. In the first simulation, a charge balanced asymmetric

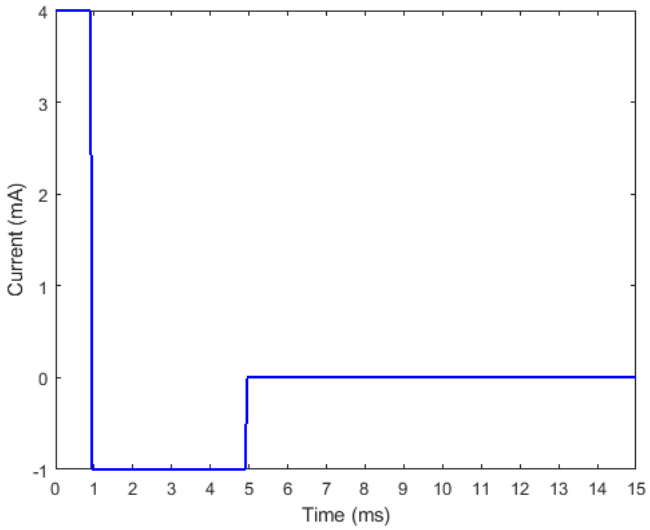
biphasic current pulse was used as the stimulus, while a charge imbalanced biphasic current pulse was used in the second simulation.

2.2.3 Results of the Charge Balancing Simulations

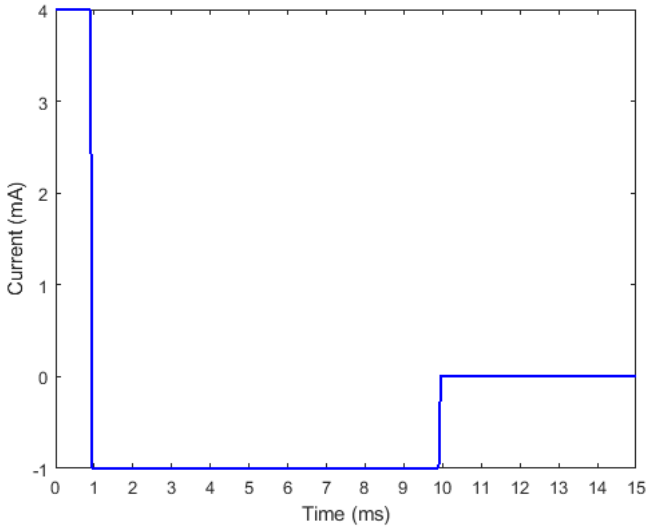
Two proof-of-concept simulations were performed to simulate the effects of charge distribution in a biological system under stimulation. First a simulation was performed to observe the effects when a charge balanced biphasic stimulus was applied to the system. The two stimuli used in the simulations are shown in the figure 2.16.

The simulation results extracted from the first simulation are shown in figures 2.17 and 2.18. Figure 2.17 shows the charge concentrations of Na^+ and Cl^- ions extracted over time at an edge of the stimulating electrode. It can be seen that at time $t = 0$ both Na^+ and Cl^- ions have their initial concentration which was set to be $300mol \cdot m^{-3}$ in the simulation. As the transient simulation progresses, the Na^+ concentration starts to drop while the Cl^- concentration starts to rise at the selected edge. This observation agrees with the theory because at the anodic phase of the pulse, the edge attracts the negative ions while repelling the positive ions. As the cathodic phase of the pulse begins, the concentrations start to reach back to their steady concentration level and at the end of the charge balanced biphasic pulse this concentration level is reached. Figure 2.18 shows the same phenomena in a $3D$ space where the charge concentration in the electrode-tissue interface is clearly shown.

The second simulation shows the distribution of the charges in the electrode-tissue interface when a charge unbalanced pulse is used as the stimuli. The simulation results extracted from the second simulation are shown in figures 2.19 and 2.20. Results of the second simulation is identical to the results of the first simulation until time $t = 5ms$ is reached where the balancing of the charges should have reached. However, as it is shown in figure 2.16b, the cathodic pulse continues for 5 more milliseconds after this point causing a charge imbalance to build up. It is evident from both figures 2.19



(a) Charge balanced biphasic pulse



(b) Charge unbalanced biphasic pulse

Figure 2.16. Two stimuli applied to one of the electrodes in the two simulations

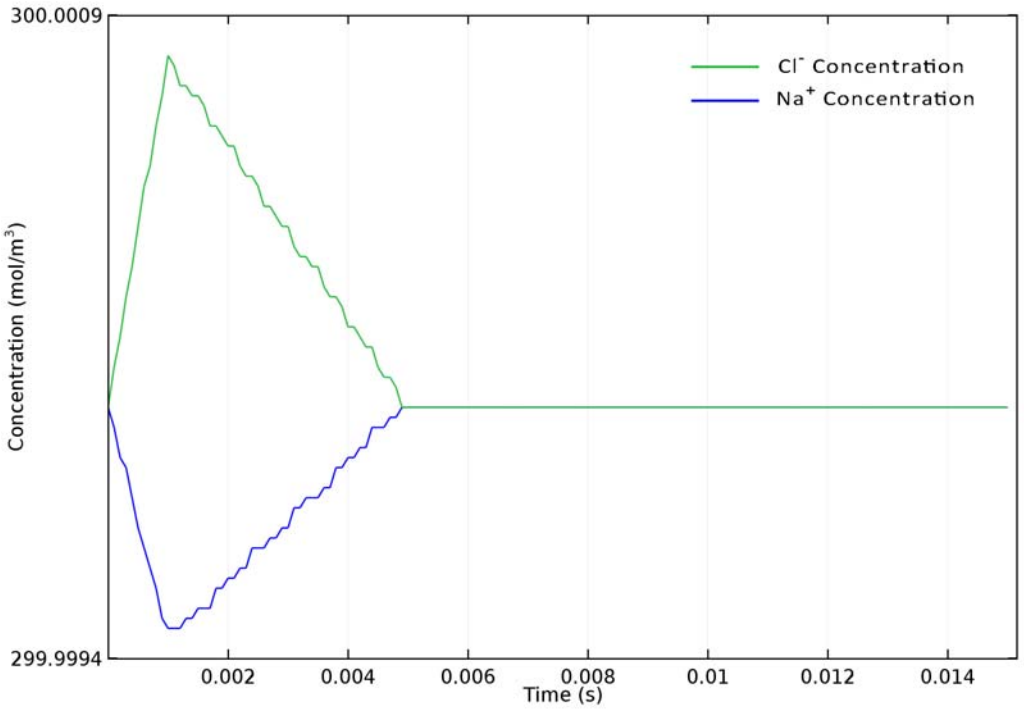
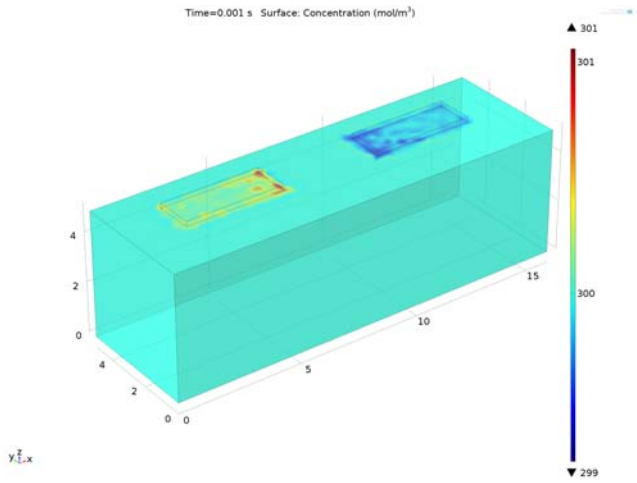


Figure 2.17. Balanced charge concentration in $mol \cdot m^{-3}$ at an edge of the electrode with respect to time

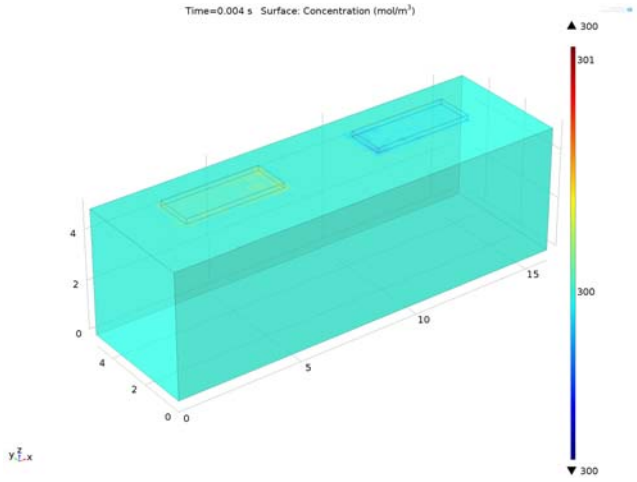
and 2.20 that an excess of charges are accumulated in the electrode-tissue interface at the end of the stimulation pulse.

It has been shown and discussed in the literature that therapeutic electrical stimulation can cause tissue damage[36, 37]. Multiple factors such as the duration of the stimulation and thresholds of the stimulation can contribute to the level of damage to the tissues. Regardless of the factor, a major cause of the tissue damage is the accumulation of charge in the electrode-tissue interface. Therefore, it is critical to minimize the charge accumulation during and after the stimulation.

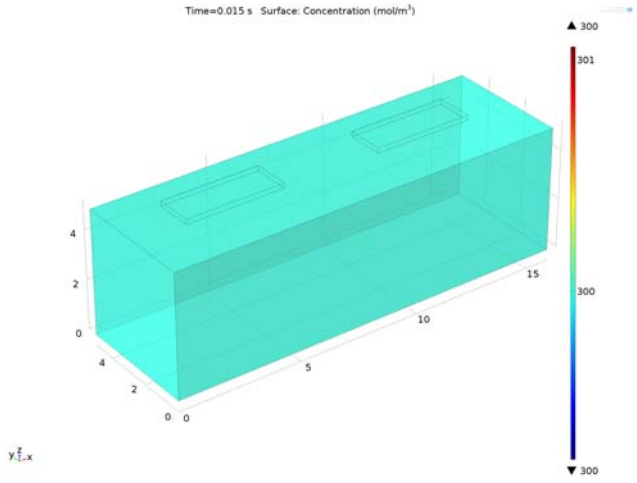
From the results of the two simulations it is apparent that using charge balanced biphasic pulses can prevent or reduce the charge accumulation in the electrode-tissue interface. However, it has been shown that due to the capacitive-resistive nature of the load, charge balancing may not be achieved even with a use of charge balanced biphasic pulses[22, 38]. Therefore, additional methods of charge balancing must be



(a)



(b)



(c)

Figure 2.18. Balanced Sodium ion concentration in $mol \cdot m^{-3}$ at the electrode-tissue interface with respect to time. (a) at the anodic phase of the pulse, (b) at the cathodic phase of the pulse, (c) at the end of the pulse

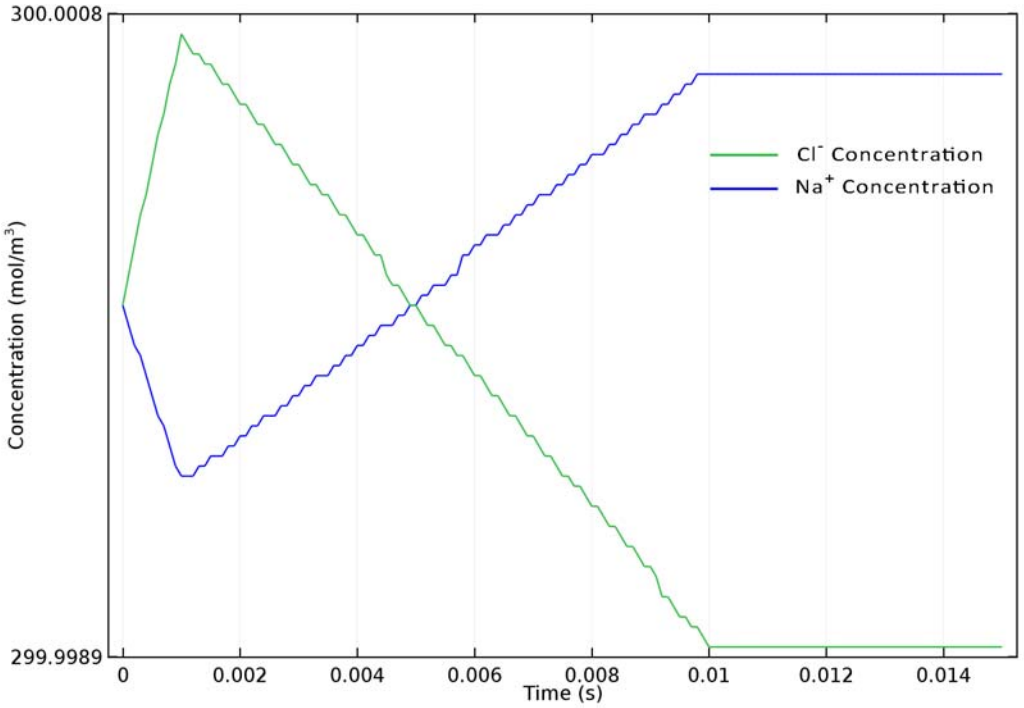
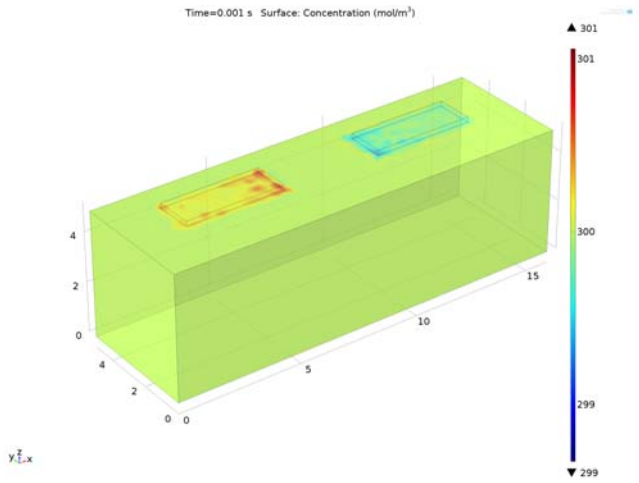


Figure 2.19. Unbalanced charge concentration in $mol \cdot m^{-3}$ at an edge of the electrode with respect to time

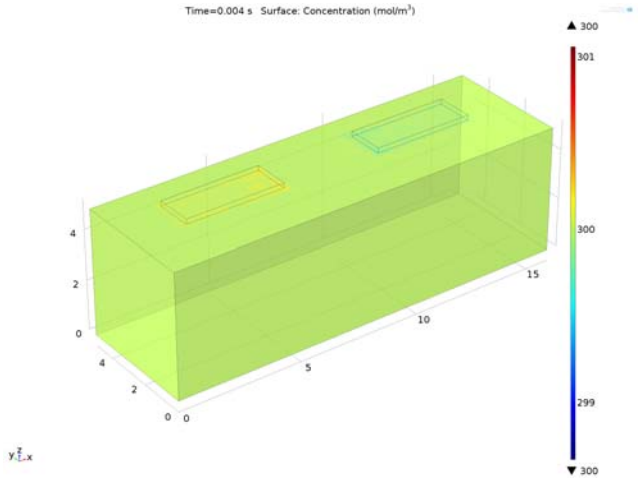
employed along with the use of charge balanced biphasic pulses[39, 40, 41].

2.3 Pulse Position Optimization

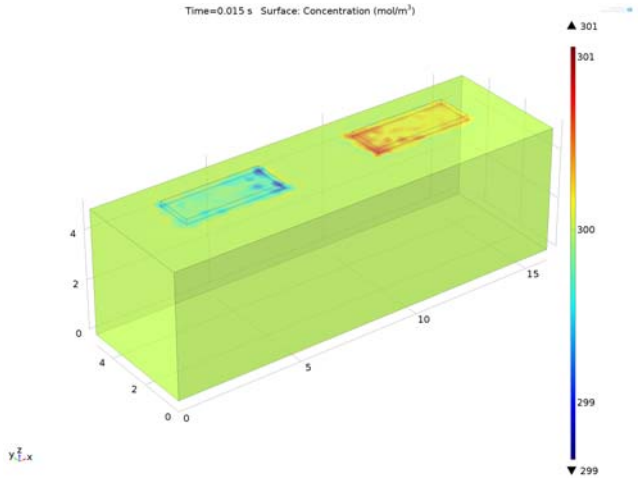
As it was described in the previous section 2.2, preventing the tissue damage during the spinal stimulation is critical. Therefore, different active and passive charge balancing methods are being utilized in not only spinal but any kind of a biological system stimulation. Regardless of the method used, biphasic pulses are used to minimize the charge accumulation at the electrode-tissue interface. To demonstrate the benefit of using biphasic pulses, a simple case of two electrodes with a single stimulus pulse was simulated and the results were discussed in the section 2.2. However, an actual stimulation scheme known as a "*program*", uses more than 2 electrodes and more than one stimulus pulse at a give point in time. Therefore, two or more electrodes can provide activation pulses to the biological system at once. This results



(a)



(b)



(c)

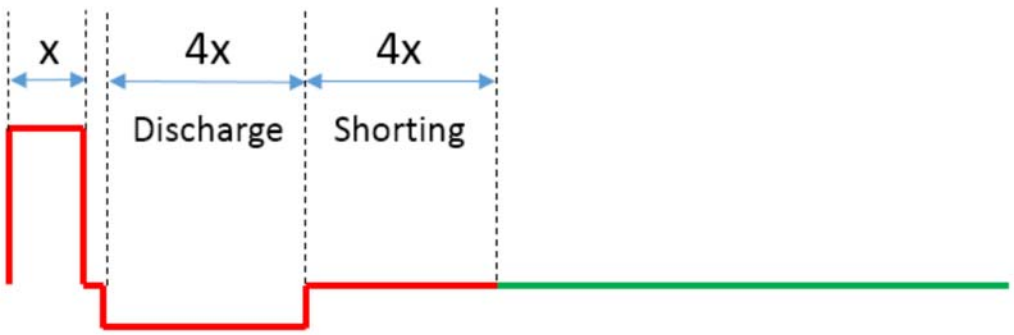
Figure 2.20. Unbalanced Sodium ion concentration in $mol \cdot m^{-3}$ at the electrode-tissue interface with respect to time. (a) at the anodic phase of the pulse, (b) at the cathodic phase of the pulse, (c) at the end of the pulse

in events known as pulse collisions or pulse overlaps. A pulse collision is where the anodic, cathodic or shorting phase of one stimulus pulse coincides with the anodic, cathodic or shorting phase of another stimulus pulse driving a second electrode. See fig. 2.21.

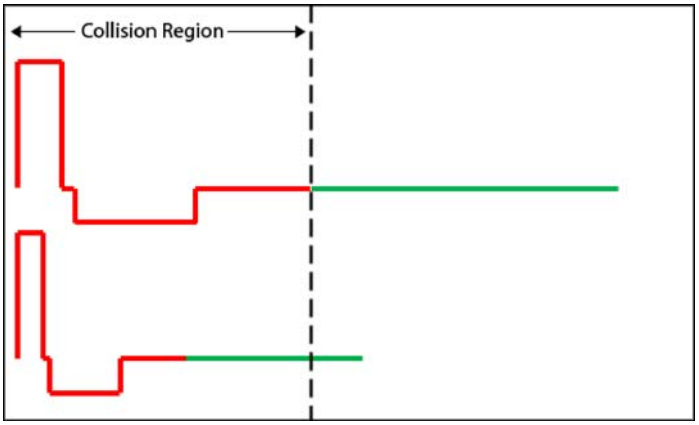
The pulse collision events are known to create perturbations in the shape of the biphasic pulses. This minimizes the ability to reduce charge accumulation after each stimulus pulse. Therefore, in an actual stimulation process, the pulse collision events must be eliminated or reduced as much as possible. In an actual stimulation program similar to the one shown in figure 2.7, multiple waveforms that have multiple frequencies are used. Therefore, mathematically it is not possible to completely remove the pulse collision events. However, the pulse collision events can be minimized. This minimization process can be done by varying the starting position (pulse shifting) of each stimulus pulse. Although the method to minimize the collisions appear to be straightforward, finding the optimum shifting positions can be laborious and time consuming. To simplify this task and to easily find the optimum shifting position, an algorithm was developed and a software program was written based on the algorithm.

The flowchart of the developed algorithm that can optimize three waveforms is shown in the figure 2.22. The same algorithm can be extended to optimize any number of simultaneous waveforms. At the time this algorithm and software were developed, five waveforms were the maximum number of simultaneous waveforms that were used in the experiments. Therefore, a software program was written to support and optimize up to five simultaneous waveforms. MATLAB code that was written to make the graphical user interface (GUI) as well as to minimize the collisions are shown in the *Appendix A*. The software has gone through several versions and the interface of the final version (`Version 4.3`) is shown in the figure 2.23.

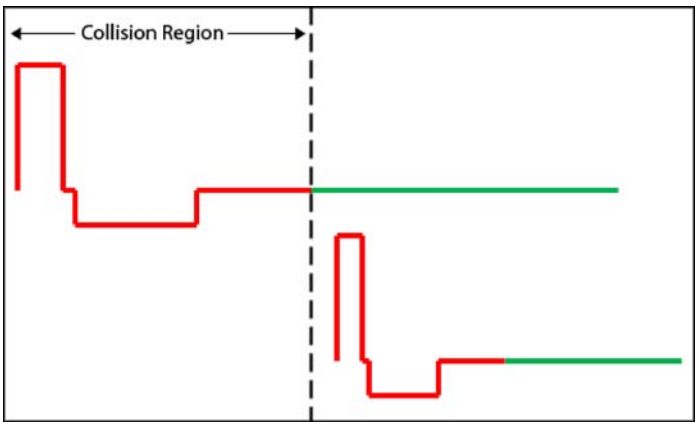
As it can be seen from the figure 2.23, the software provides the capability to perform the optimization of up to five different waveforms. Once the number of



(a)



(b)



(c)

Figure 2.21. Composition of a biphasic pulse and the definitions of a pulse collision. (a). Example composition of a biphasic pulse. (b). Example of a waveform collision. (c). Example of a non-collision.

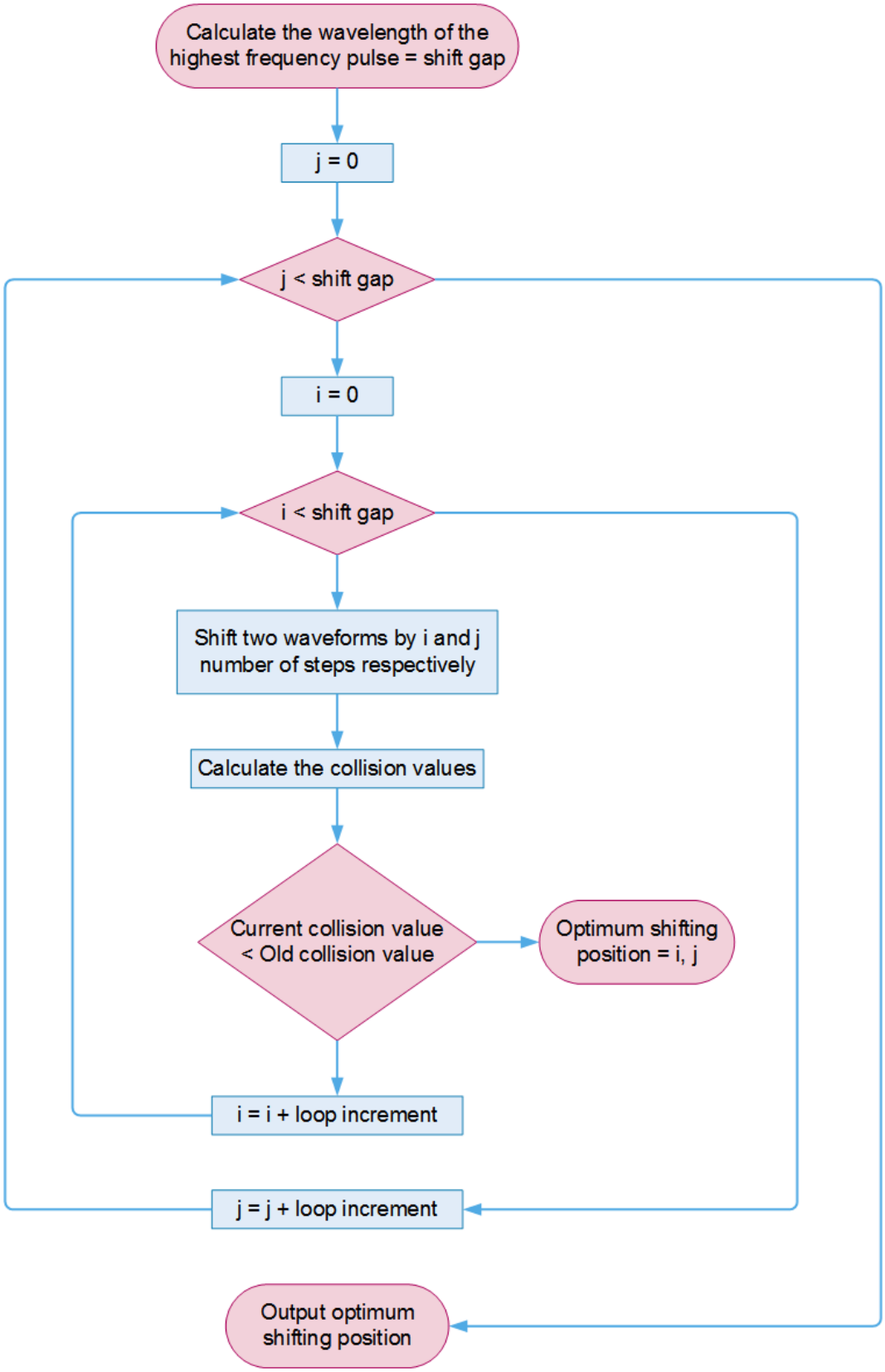


Figure 2.22. Flowchart of the PPO algorithm for three waveforms

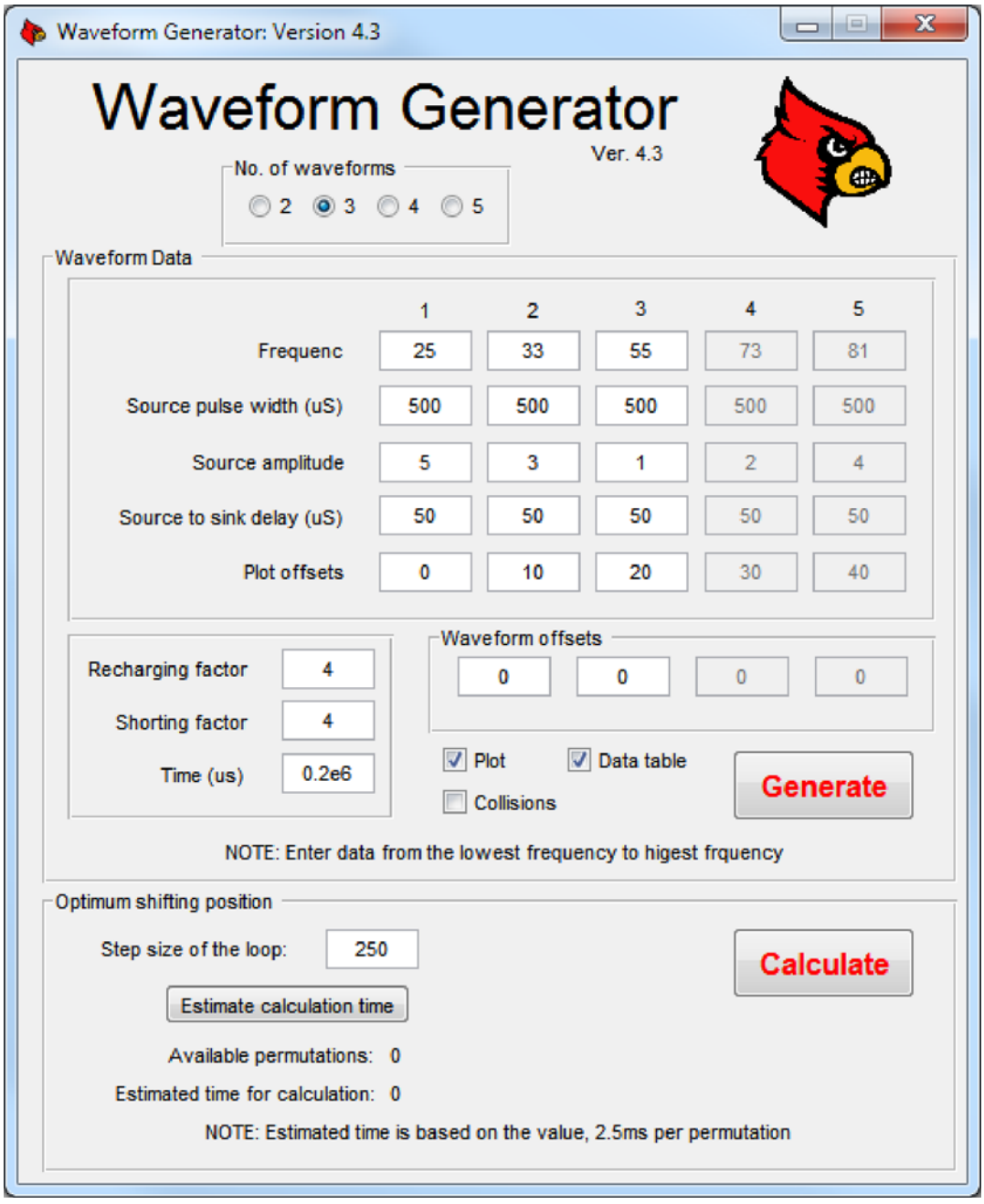
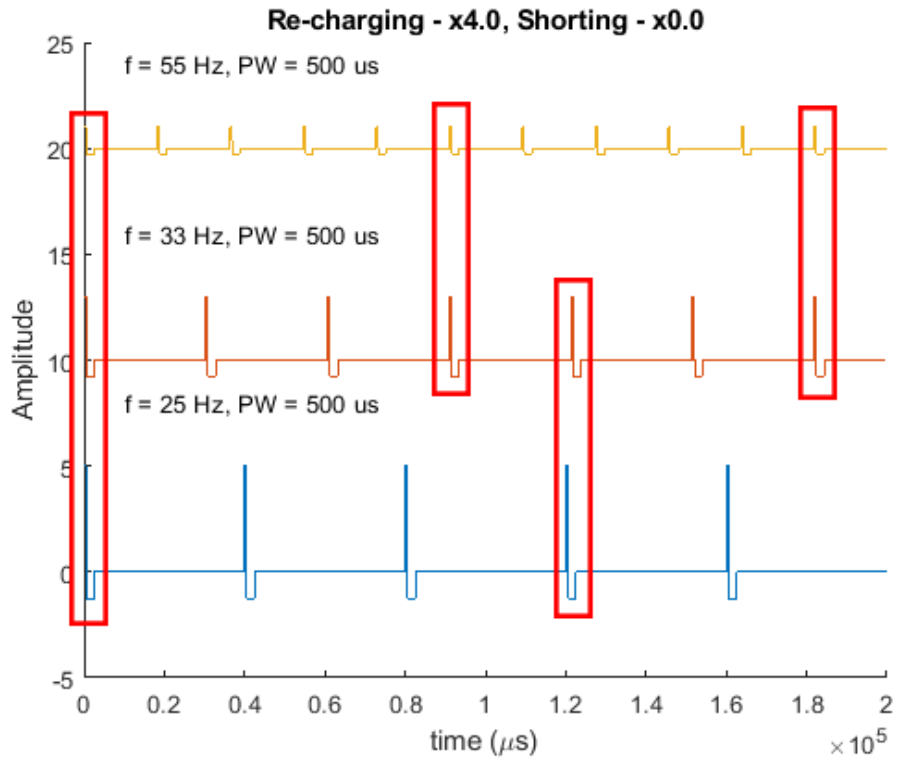


Figure 2.23. Interface of the PPO Software

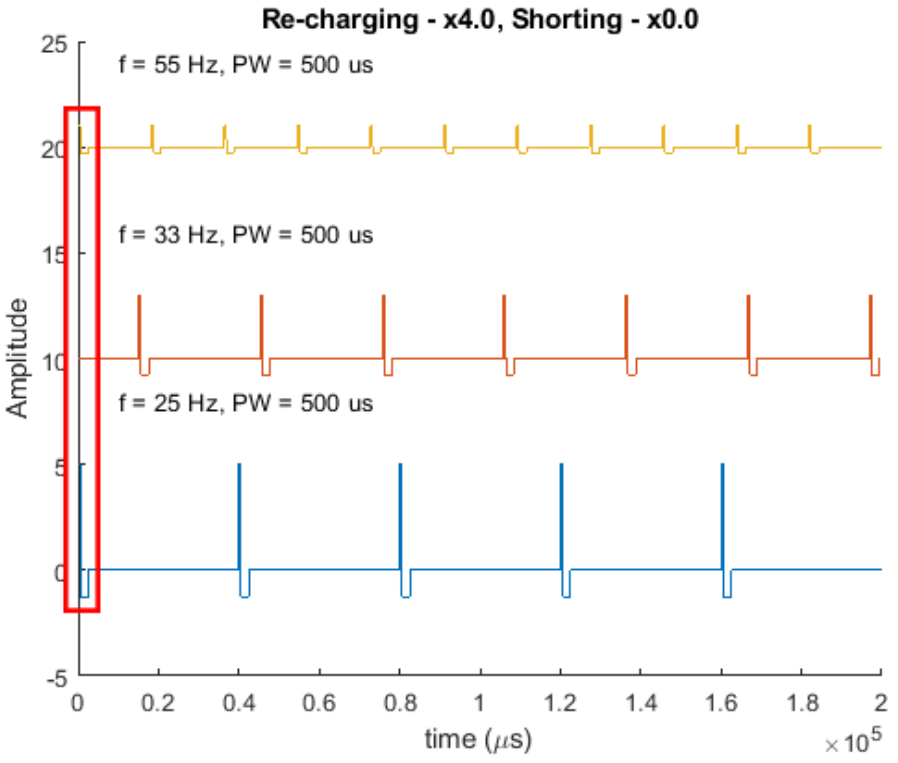
waveforms are set, the waveform information are entered in the second section of th GUI. The default values for the waveform offset is set to zero, indicating no waveform shifts are performed and all the waveforms start at the same time. Clicking on the generate button generates the waveforms for the provided information and a summary of the results. In order to optimize the pulse positions, the calculate button is clicked. Depending on the provided parameters, this can take a few seconds to several minutes to find the optimum waveform offsets. Once the software returns the optimum values, they can be entered in the waveform offsets section in the GUI and regenerate the waveforms that are placed at optimum positions.

Figures 2.24 and 2.25 show the results obtained from the PPO software. Results show the waveform data for three waveforms before and after the optimization. The Waveforms used here have the frequencies $25Hz$, $33Hz$ and $50Hz$. The figure 2.24a shows the waveform positions before the optimization and the figure 2.25a shows the corresponding collision data. It can be seen from the data that there are 7 collisions in the shown time range of $200ms$. The figure 2.24b shows the same waveforms after the positions have been optimized and the figure 2.25b shows the corresponding collision data table. It is evident that the optimization has reduced the number of collisions from 7 to 1.

Although it might not be mathematically possible to remove all the collisions further optimization is possible by changing some of the parameters of the algorithm. For instance, in the optimization settings there is a parameter called step size of the loop which is set to 250 by default. This indicates that waveforms are shifted by $25\mu s$ steps to find the optimum positions. However this number can be changed according to the needs and a more optimum result if exists can be found.



(a)



(b)

Figure 2.24. Waveforms generated via the PPO software. (Red rectangles show the collisions). (a). Waveforms without any pulse shifting. (b). Waveforms with pulse shifting applied.

	%No Pulse	# Collisions	% Collisions
25Hz, 33Hz	86.6957	2	40
25Hz, 55Hz	80.9510	1	20
33Hz, 55Hz	80.9210	3	45.4545
All Three	76.5062	1	15.1515

(a)

	%No Pulse	# Collisions	% Collisions
25Hz, 33Hz	84.7608	0	0
25Hz, 55Hz	80.9510	1	20
33Hz, 55Hz	77.1411	0	0
All Three	72.0614	0	0

(b)

Figure 2.25. Data tables corresponding to the plots in figure 2.24 obtained via the PPO software. (a). Collision results for waveforms without shifting. (b). Collision results for waveforms with shifting applied.

CHAPTER 3

THE SPINAL CORD

3.1 Introduction to the Spinal Cord

The spinal cord is a long, thin, fragile, tubular structure that facilitates a bundle of nerve tissue and other cells. The structure extends from the foramen magnum where it is continuous with medulla oblongata to the level of the first or the second lumbar vertebrae. Length and width of the spinal cord varies on number of factors including the gender of the person. However on average, the length of the spinal cord of a grown human is about $43cm - 45cm$. The width of the spinal cord varies along the length of the spinal cord, ranging about $13mm$ in the cervical and lumbar regions to about $6.5mm$ in the thoracic region. A diagram of the spinal cord and different regions of the spinal cord are shown in the figure 3.1.

The primary function of the spinal cord is to maintain the communication between the brain and the body. This includes the transmission of the nerve signals from the motor cortex to the brain and from the sensory neurons to the sensory cortex. As it was discussed in chapter 1.1, this communication gets interrupted in a spinal cord injury. Therefore, external stimuli are used to bridge the communication across the damaged area of the spinal cord.

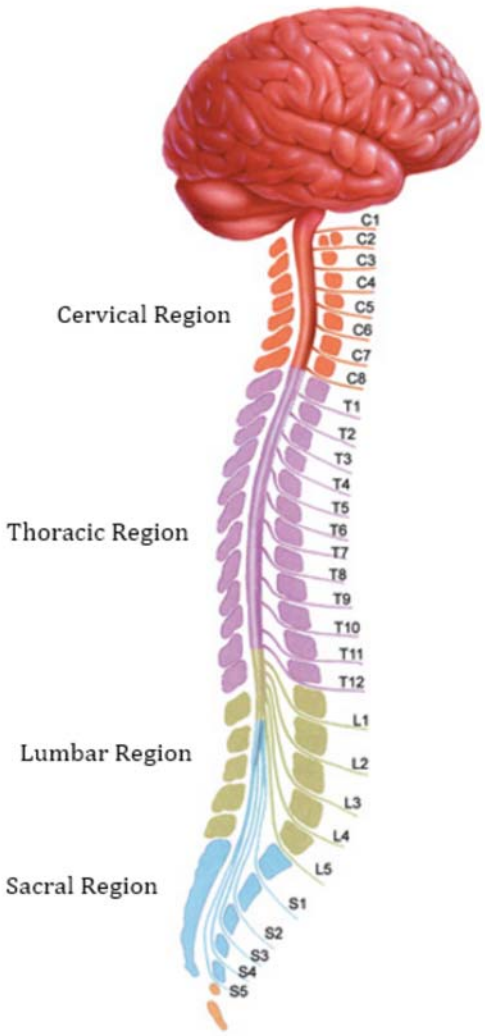


Figure 3.1. Spinal cord and different regions of the spinal cord

3.2 Modeling the Spinal Cord

In the chapter 2.1 it was shown, the methods and results extracted from the electric field simulations performed in a proof of concept study. The primary objective of the simulations was to gain an insight in to the electric field distribution patterns in a region of the spinal cord. This information is then used to optimize the stimuli intensities and patterns to alter the electric field distribution pattern as necessary. In parallel to the simulation studies a different group of researchers performed experiments, where they were trying to activate damaged neural communications in

a group of patients with spinal cord injuries. The focus of this experimental study was to activate the voluntary muscle movements. To optimize the parameters of the experimental study, an accurate representation of the electric field distributions in the spinal cord was needed. Therefore, the preliminary electric filed simulation study was extended to incorporate the actual geometry of the spinal cord and the electrode array.

A detailed model representation of the spinal cord needs to include different sections of the spinal cord as shown in the figure 3.2. In the experimental studies the electrode array was implanted on the surface of the dura matter of the spinal cord. Therefore, extra layers cerebrospinal fluid (CSF), dura matter and the epidural fat was included in the model geometry. As it was described in chapter 3.1 the dimensions of the spinal cord varies depending on different factors. Therefore, average measured dimensions of the lumbar region (L1-L5) of the spinal cord were used to create the model geometry[42].

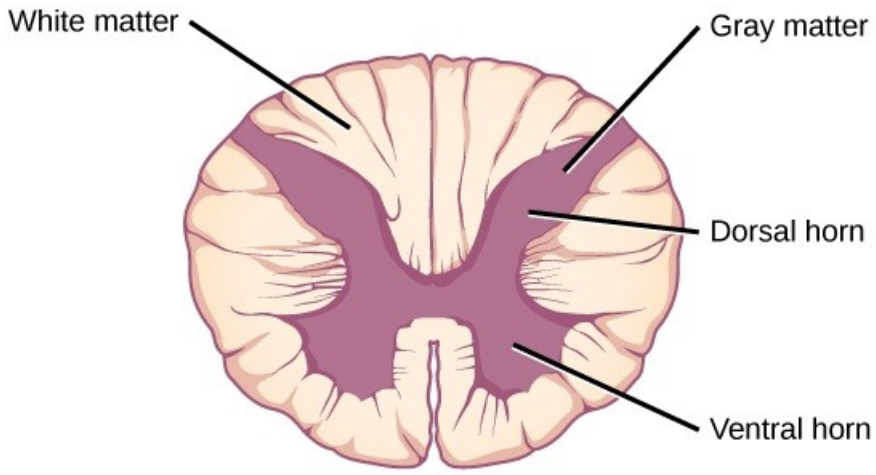


Figure 3.2. Cross section of the spinal cord

The detailed 3D model of spinal cord with 16 electrodes created in Solid Works is shown in the figure 3.3. For the same reasons explained in the chapter 2.1 the paddle of the electrode array was not included in the model.

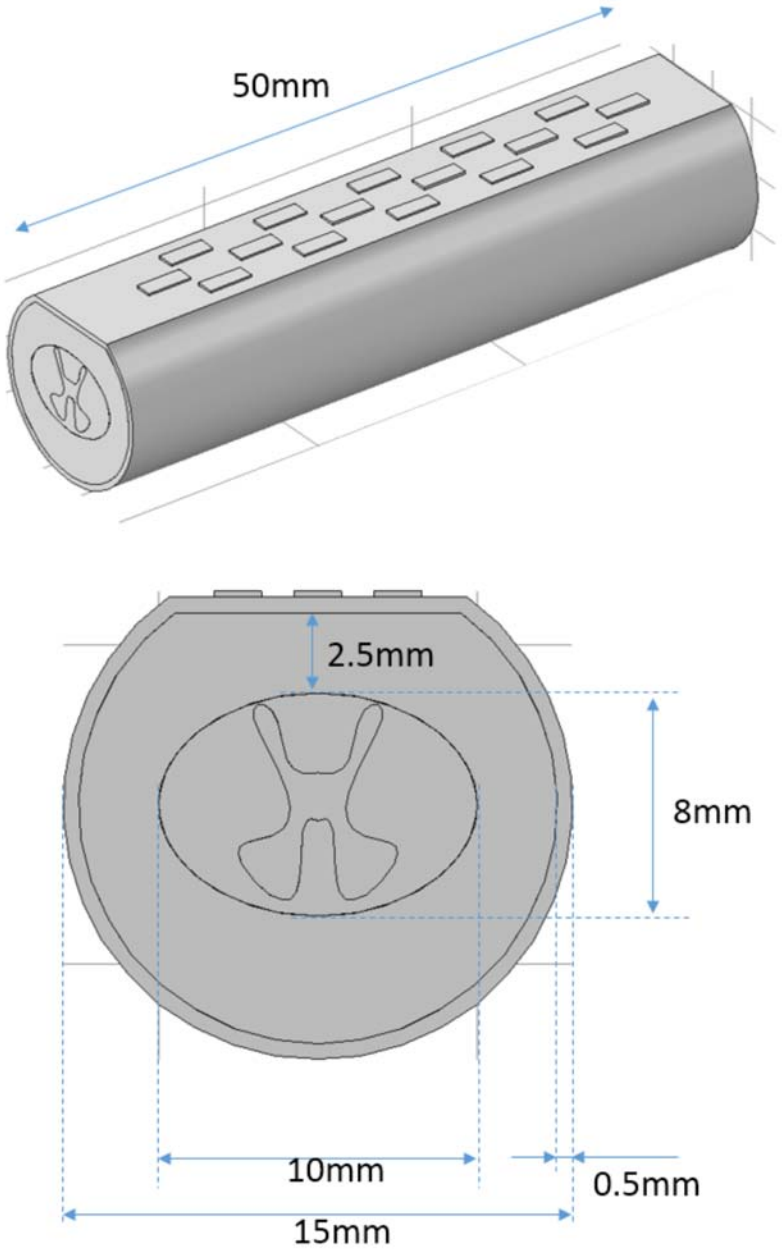


Figure 3.3. Detailed 3D model of the spinal cord and the electrodes created in Solid Works

Once the 3D model is prepared it was imported to COMSOL multiphysics. The physics of the simulation remains similar to the previous study since the goal here was the observation of the electric field distributions as before. However, a different section of the spinal cord had to be provided with the corresponding material properties.

The values of the dielectric properties of the biological tissues varies on the frequency. Studies have been conducted and models have been presented in the literature to calculate relevant dielectric properties at a desired frequency. In a series of papers Gabriel *et al.* published their experimental measurements of dielectric properties of biological tissues [43, 44, 45]. Further, Anderson and Rowley have published an Excel based calculator that calculates dielectric properties of biological tissue for a given frequency [46]. Other scholars who have conducted and published work similar to this have also provided the relevant material properties [47, 48, 49]. The dielectric properties used in this study are listed in the table 3.1.

Table 3.1. Dielectric properties for different biological tissue in the spinal cord

	Electrical Conductivity (Sm^{-1})	Permittivity
White matter (Longitudinal)	0.6	38.79
White matter (Transverse)	0.083	1846.05
Gray matter	0.23	458.89
CSF	2	108.89
Dura matter	0.6	141.25
Epidural Fat	0.04	38.72

Since the model geometry had evolved from the previous electric field simulation study, the computational cost also increased. Hence, the amount of time and computational resources needed to solve the problem had increased. Therefore, to test the simulation setup, a static simulation was performed instead of a transient one. Only two electrode pairs were used where one electrode in each pair stimulated the model with a static $5V$, while the other electrode was grounded. $5V$ was used because it was a typically used value in the actual human experiments. Figure 3.4 shows the

two pairs of electrodes used in the simulation.

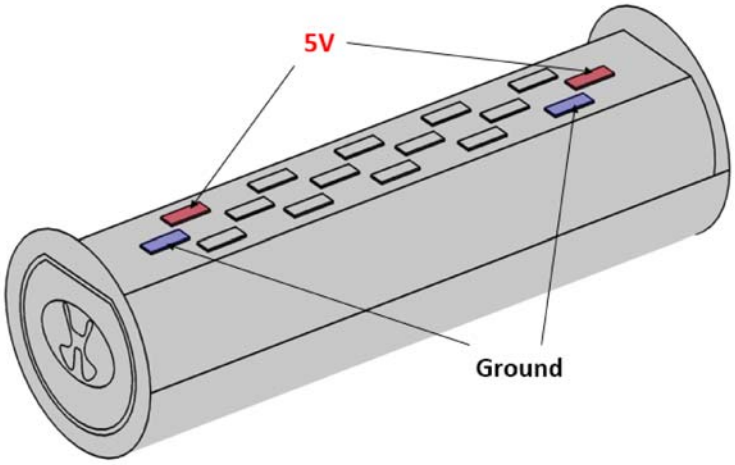
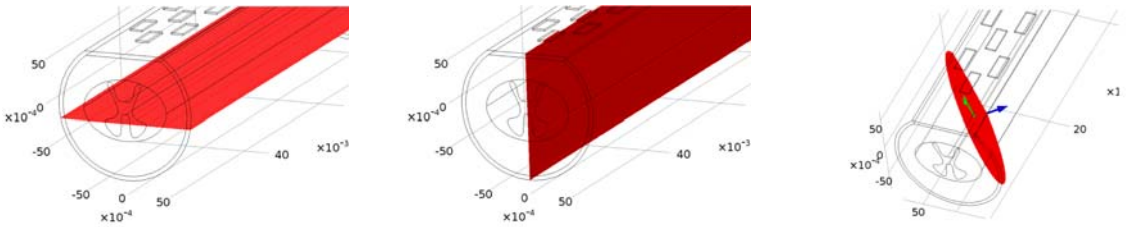


Figure 3.4. Two pairs of electrodes and the stimuli applied to them

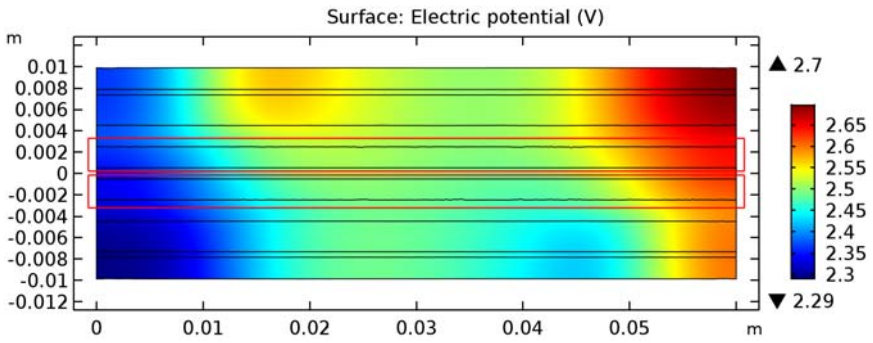
The goal of the experiments performed on the patients with a SCI was to enable the lost communication in the spinal cord sustained from their injury. The specific study was focusing on the motor functions of the patients and was trying to activate the communications in the motor neurons. Therefore, the simulation was targeted to explore the effects of the stimuli in the ventral horns which ares the regions of the spinal cord where the motor neurons are located.

Three planes were defined on which the results of the simulation were extracted. Defined three planes are shown in the figure 3.5. The results extracted on the defined planes are shown in the figures 3.6, 3.7 and 3.8.

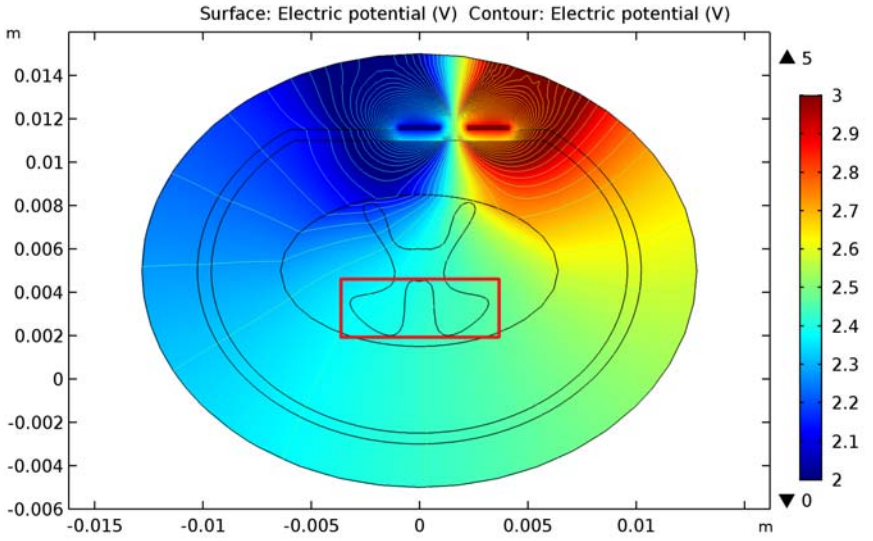


(a) **(b)** **(c)**

Figure 3.5. Three planes on which the results were extracted. (a). Horizontal plane, (b). Vertical plane, (c). Plane across a pair of active electrodes.



(a)



(b)

Figure 3.6. Potential distribution in the spinal cord from the stimuli shown in figure 3.4. Red rectangles encloses the regions where the motor neurons are located. (a). Potential distribution parallel to the electrodes, (b). Potential distribution in the plane that goes through two active electrodes.

The red rectangles in the figures 3.6, 3.7 and 3.8 shows the regions of interest where the motor neurons are located. From the potential distributions it can be seen that the potential at the ventral horns is about 50% different that at the surface of the active electrodes. On the other hand, the electric field distribution patterns indicate that the electric fields in the ventral horns are significantly lower than the electric fields in the vicinity of the active electrodes. The theoretical results could not be compared with the experimental observations, since it was not possible to map the potential distribution or the electric field distributions in an actual human

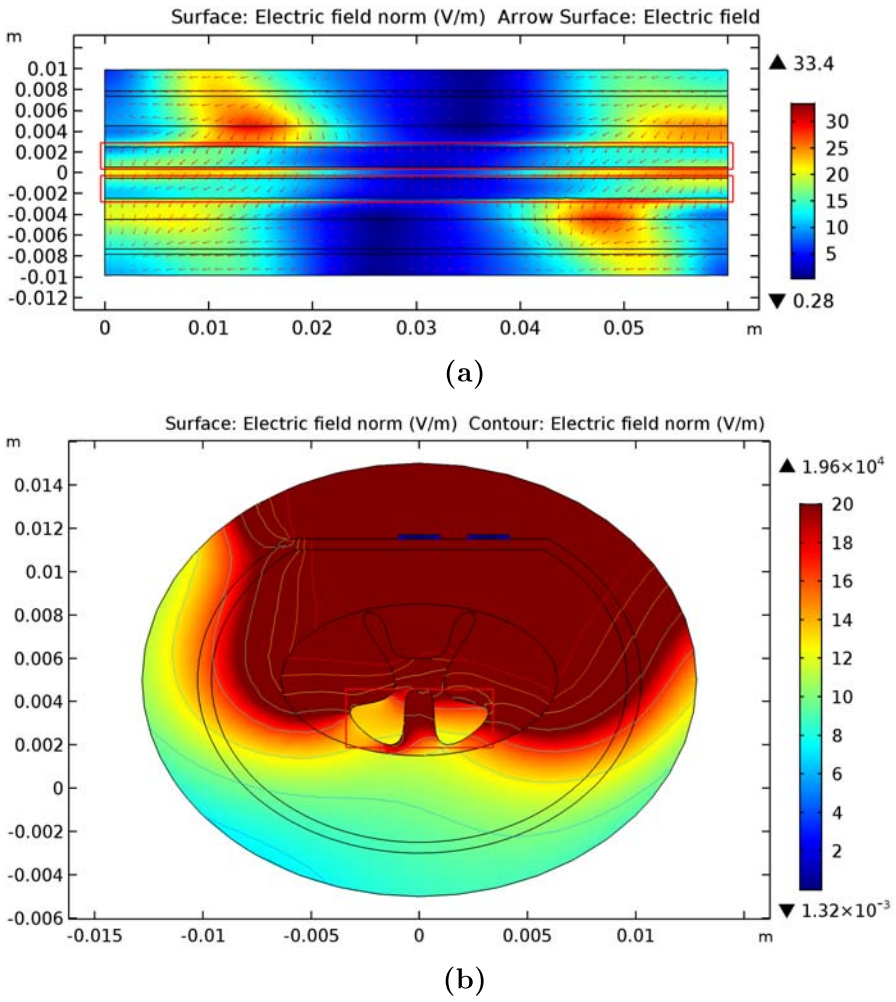
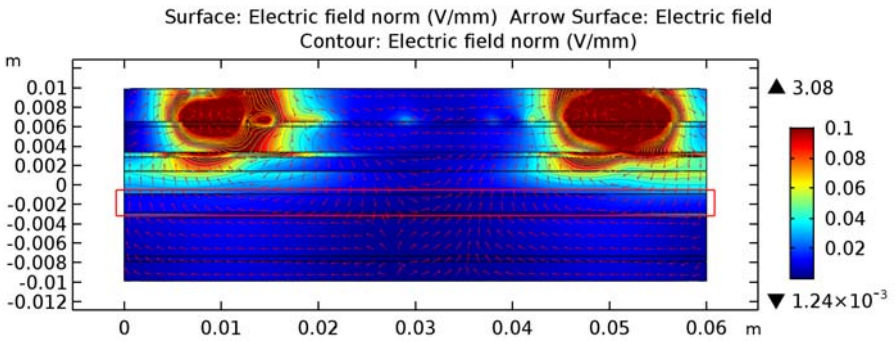


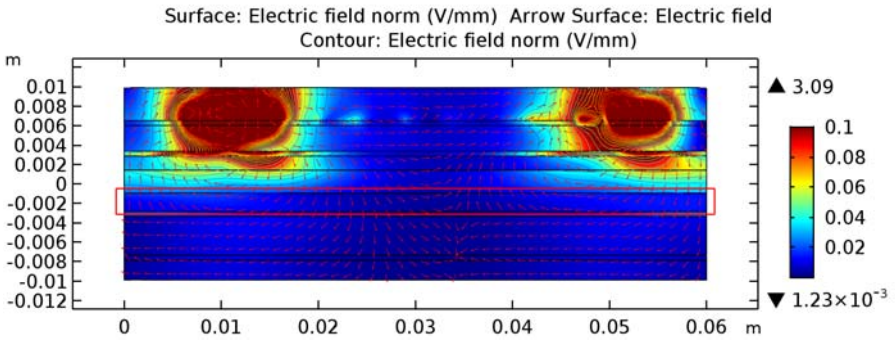
Figure 3.7. Electric field distribution in the spinal cord from the stimuli shown in figure 3.4. Red rectangles encloses the regions where the motor neurons are located. (a). Electric field distribution parallel to the electrodes (b). Electric field distribution in the plane that goes through two active electrodes.

spinal cord under stimulation. However, the results show the possibility of visualizing the electric fields and potential distributions inside the spinal cord using 3D FEM software. Given that the results show the approximate potential and electric field values in the spinal cord when stimulated, it can be deduced that the electric field values between $10V/mm - 20V/mm$ are reasonably sufficient to activate the disturbed communication in a damaged spinal cord.

The researchers who were involved in the human experimental studies explored the possibility to correlate the electric or potential distribution patterns in the spinal



(a)



(b)

Figure 3.8. Electric field distribution in the spinal cord from the stimuli shown in figure 3.4. Red rectangles encloses the regions where the motor neurons are located. (a). Electric field distribution perpendicular to the electrodes through the left ventral horn, (b). Electric field distribution perpendicular to the electrodes through the right ventral horn.

cord to a muscle movement they've observed in the patients[9, 10]. However, correlating one data set to another requires a large number of data samples which were not available at the time of this simulations. Further, one set of stimulation parameters that worked for one patient would not necessarily work for a second patient. This indicated that the observed muscle movements are correlated not only with the stimulation parameters but also with the properties of the spinal cord damage.

Due to the lack of resources and funding to find a correlation between the simulation and experimental results, it was decided to take a different approach to study the effects of stimulation. It was decided to explore the microscopic behavior of the spinal cord, instead of a macroscopic behavior. Primarily, the behavior of individual

neurons under different stimuli.

CHAPTER 4

NEURON LEVEL MODELING

All of the previous simulation studies on the spinal cord and spinal cord stimulation described the macroscopic behaviors of the spinal cord. However, as it was described in chapter 3.2, a study of microscopic behaviors of the spinal cord could lead to understanding a different perspective of the spinal cord stimulation and could possibly shed light on some of the unknowns of the internal mechanics of spinal cord stimulation.

4.1 Anatomy and the Physiology of a Neuron

Before delving in to the modeling and mechanics of a neuron, it is important to understand the anatomy and the physiology of the nervous system and neurons. The nervous system of an animal is a complex system that is responsible for all the sensations, actions as well as thoughts and emotions. This complex system is broken down into small parts as shown in the figure 4.1.

As shown in the figure 4.1, the main component is the Central Nervous System (CNS) which includes the brain and the spinal cord. Next is the Peripheral Nervous System (PNS), which includes the cranial nerves and spinal nerves that extend outward from the brain and the spinal cord. PNS controls the communication between the CNS and the rest of the body. PNS includes both the sensory division which conducts impulses from the receptors to the CNS and the motor division that conducts impulses from the CNS to the muscles and glands. The motor division is

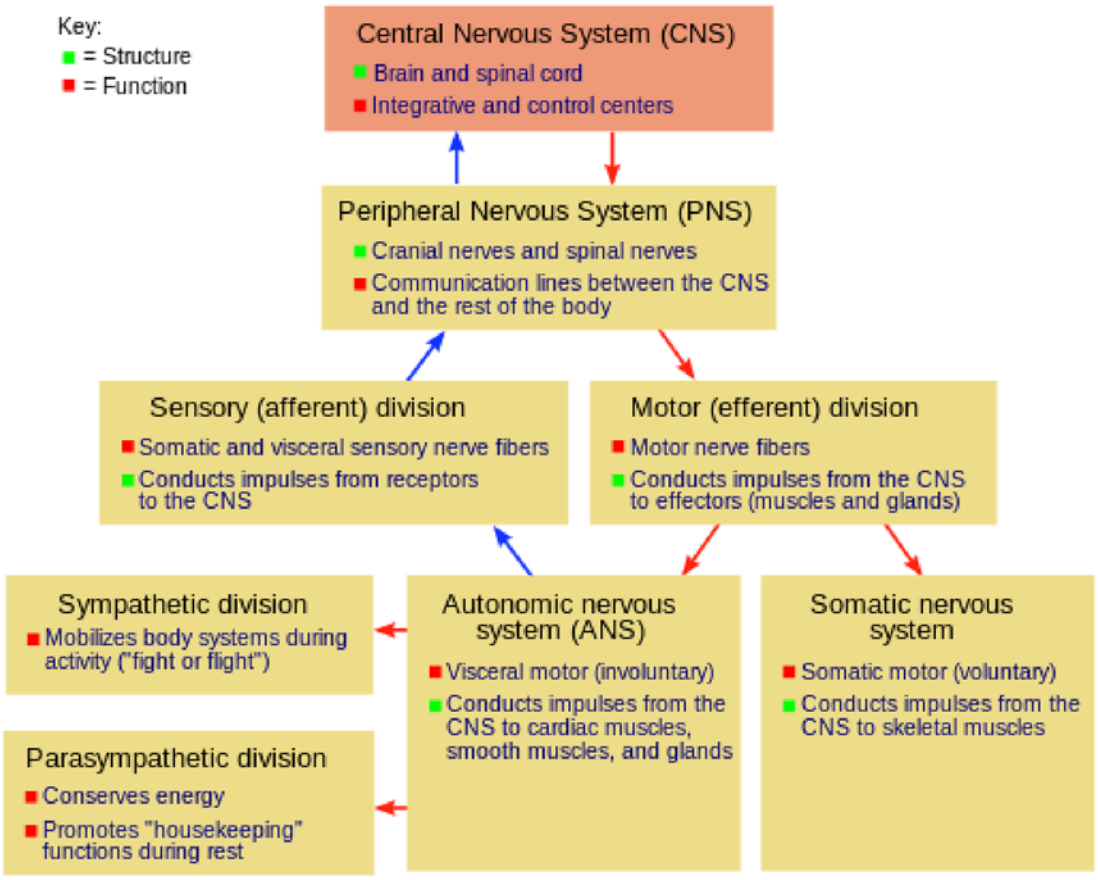


Figure 4.1. Major parts of the vertebrae nervous system [50].

broken down into two more parts identified as the somatic nervous system and the autonomic nervous system (ANS) which includes the cardiac muscles, the smooth muscles and the glands that mediates the involuntary movements. The focus of this study and the aforementioned experimental studies that attempts to regain muscle movement in paralyzed patients is rooted in the somatic nervous system as it controls the voluntary muscle movements.

The nerve cell or the neuron is the basic building block of the nervous system that is responsible for cellular communication between the brain and the rest of the body. The human brain alone comprises 86 billion neurons, each linked to thousands of other neurons [51]. Therefore, the brain has trillions of specialized connections between neurons known as synapses [52]. The brain signals which are transmitted from one neuron to another neuron through a synapse are conducted along the neuron

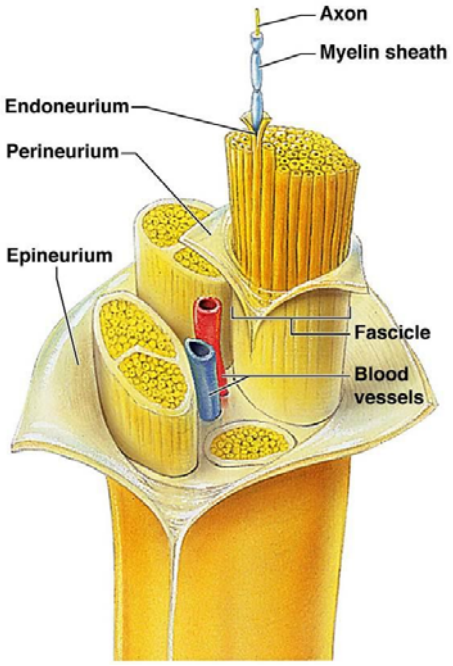


Figure 4.2. Structure of a nerve showing the fascicles and axons.

via a structure known as the axon. An axon also known as a nerve fiber is a long, thin projection of a nerve cell that conducts electrical impulses known as action potentials away from the neuron body. Typically, larger number of neurons are grouped together to create other structures. A multitude of axons are grouped together to create a structure called a fascicle. Multiple fascicles along with blood vessels are grouped together to create a nerve. Figure 4.2 shows an illustration of a nerve fiber and its internal structures.

There are many different types of neurons with different shapes, sizes and physiological differences. However, most of their general properties and behaviors can be summarized using a generic neuron as the illustration shown in figure 4.3.

Based on the presence of the myelin sheath around the axon, neurons are categorized in to two categories, the myelinated neurons and the non-myelinated neurons. Myelin is formed by a cell called a Schwann cell in the PNS and the CNS. Having a myelin sheath around the axon increases the speed of the nerve conduction. Myelination has also made possible the development of larger body sizes in vertebrates

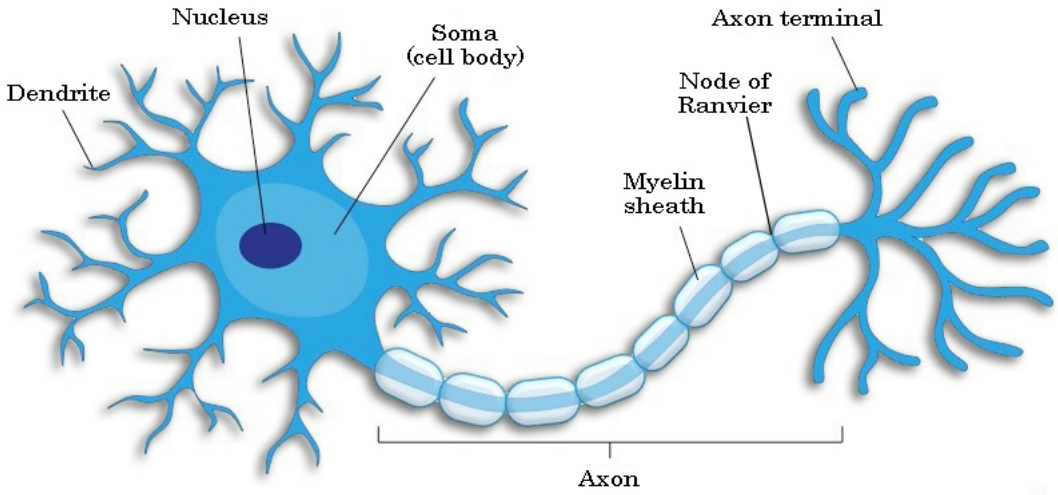


Figure 4.3. Structure of a neuron. The direction of pulse propagation is from left to right.

[53].An axon may have a large number of Schwann cells covering it with small gaps in between them. This gap in between Schwann cells is referred to as the node of Ranvier. It is at the nodes of Ranvier where series of ion exchanges (mainly sodium and potassium) occur to create nerve impulses that propagate information through the axon.

4.2 Membrane Potential

Like every animal cell, the neuron is enclosed in a plasma membrane. This membrane marks the border through which different substances can enter and leave the cell. The extra and intra-cellular fluid typically contains Na^+(sodium), K^+(potassium) and Cl^-(chloride) ions. Transmembrane proteins also known as ion transporters regularly push ions through the membrane and establish a concentration gradient across the membrane [54]. Typically, Na^+ and Cl^- are at high concentrations in the outside region of the nerve cells and at low concentrations in the inside regions whereas, K^+ is at a high concentration inside the nerve cell and at a low concentration outside the nerve cell (see figure 4.4) [55]. The differences in the concentration of ions on each

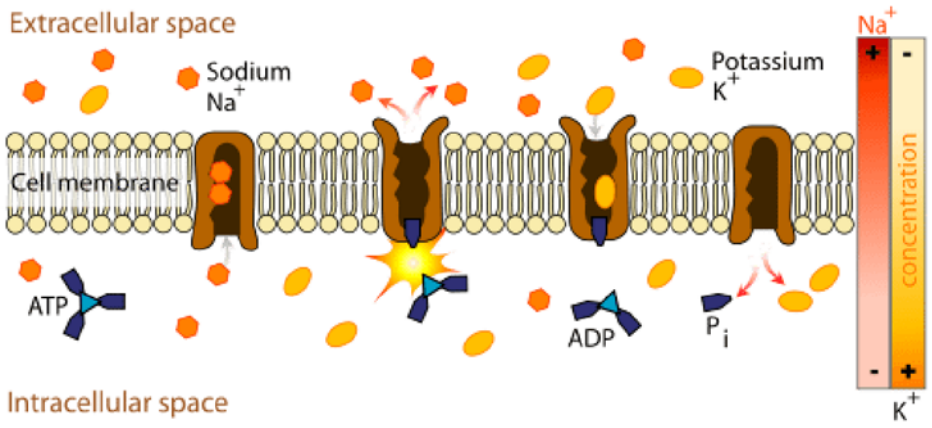


Figure 4.4. Illustration of the neuron cell membrane and the ion concentrations.

side of the membrane leads to a potential difference called the membrane potential and is shown by the equation (4.1). V_i is the potential inside the cell and V_o is the potential outside the cell.

$$V_m = V_i - V_o \tag{4.1}$$

The unbalanced sodium ions outside the cell and the potassium ions inside the cell line up on the either side of the membrane surface and attract each other. The point at which the forces of the electric fields completely counteract the force due to diffusion is called the equilibrium potential. At this point, the net flow of the specific ions is zero. Typical values of this resting or equilibrium potential varies between $-40mv$ and $-80mv$.

The membrane potential that depends on the movement of the ions through the membrane can be explained by the Goldman equation (4.2) [56, 57, 58, 59].

$$V_m = \frac{g_{Na}E_{Na} + g_K E_K + g_{Cl}E_{Cl}}{g_{Na} + g_K + g_{Cl}} \tag{4.2}$$

where, g_{Na}, g_K and g_{Cl} are the ion conductivities and E_{Na}, E_K and E_{Cl} are the

Nernst potentials of the corresponding ions (4.3) [60, 61].

$$E_j = \frac{RT}{z_j F} ln \frac{{c_j}^o}{{c_j}^i} \tag{4.3}$$

where, R is the gas constant, T is the temperature, z_j is the valence of the specific ion, F is the Faraday's constant, ${c_j}^o$ is the concentration of the specific ion outside the cell and ${c_j}^o$ is the concentration of the specific ion inside the cell. By substituting corresponding values to the (4.3), it can be found that sodium and potassium has the following resting potentials when measured from outside to inside (4.4).

$$E_{Na} = 55mV \tag{4.4a}$$

$$E_K = -75mV \tag{4.4b}$$

4.3 Nerve Conduction and Action Potentials

Neurons can receive signals from tens of thousands of other neurons [62]. The information received from one neuron is passed to other neurons via a time varying voltage pulse that propagates through the axon. This voltage pulse is created in a neuron by a rapid rise followed by a fall in the membrane potential. This spike of voltage is referred to as the action potential. The typical form of an action potential and the different phases it goes through are shown in the figure 4.5. The proteins that are embedded in the cell membrane are believed to be responsible for creating the action potentials. They can assume different formations and allow specific ions to pass through them. Therefore, they are considered as gates that depend on the voltage and called voltage-gated ion channels.

As it was mentioned before, the action potential propagation through the axon is mediated by the voltage-gated ion channels. As the membrane potential is near the resting potential ($-70mv$), the ion channels are closed. If the membrane receives a

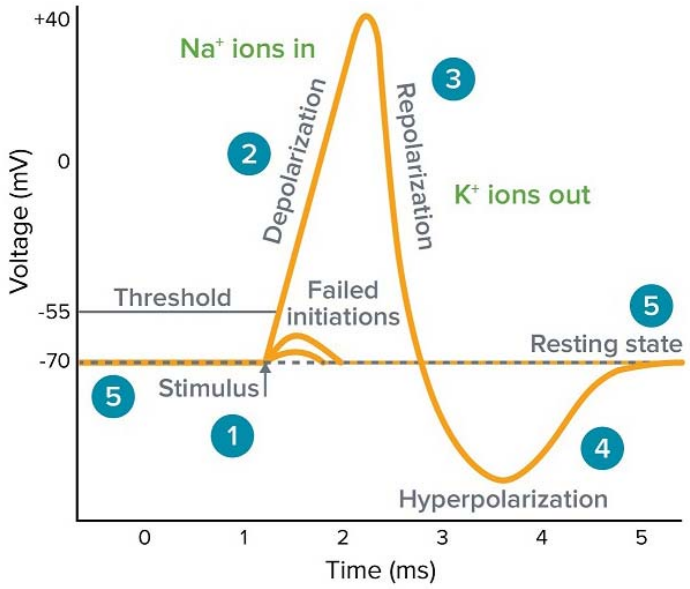


Figure 4.5. Typical form of an action potential

stimulus that is greater than the threshold, the ion channels rapidly begin to open thus depolarizing the membrane. The opened ion channels allow an inward flow of Na^+ ions which changes the electric field gradient, and in-turn further increases the membrane potential. This process proceeds until more and more channels are opened. The rapid influx of Na^+ ions causes the polarity of the membrane to reverse and the Na^+ gated channels to close. As the Na^+ gated channels start closing, the K^+ channels start to open and an outward flow of K^+ ions is started. This makes the electric field gradient to reverse and the membrane potential to go back to its resting value.

4.4 Modeling Neurons - The HH Model

For decades, researchers have been using different mathematical models to simulate the behavior of the axons and the neurons. Having accurate mathematical models that accurately demonstrates the behavior of a biological system aids scientific exploration immensely.

Researchers started making significant progress in the field of nerve excitation

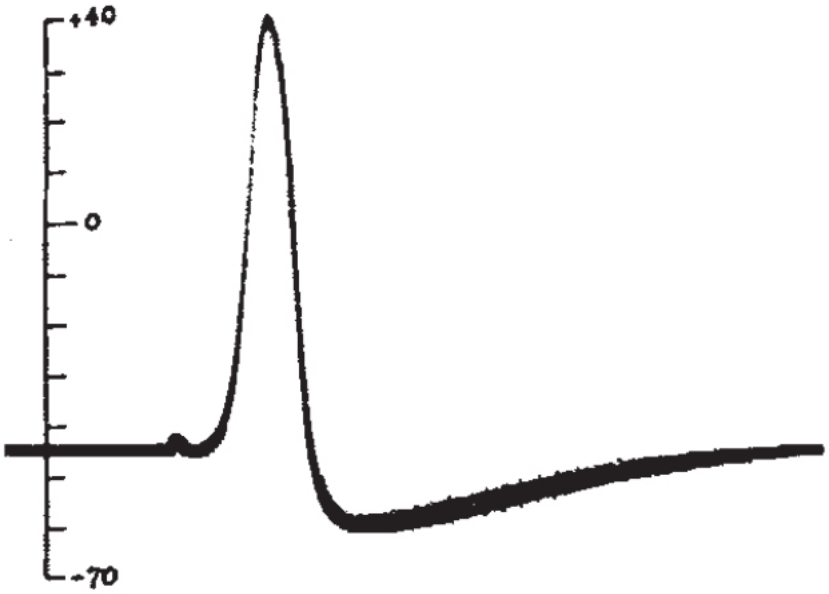


Figure 4.6. Photograph of an action potential recorded from a squid axon that was published for the first time 1939 [64].

since the beginning of 1930's. In an extensive work published in 1936, Hill proposed the formal description of nerve excitation using two variables, which were called threshold(U) and local potential(V) [63]. After the publication of the two factor theory by Hill, a large number of mathematicians entered the field of neuroscience proposing new approaches to explain the neural excitation. However, no advancements to the theory was made until Hodgkin and Huxley published their seminal experimental findings in 1939. For the first time in history they recorded the intracellular action potentials from a giant squid axon [64]. Their findings first published in 1939, was a true revolution in the field of neuroscience.

Right after their ground breaking findings in 1939, Hodgkin and Huxley had to abandon their research for few years due to the world war. However, in 1945 they started to continue the work they began in 1939 on action potentials in the squid giant axon [65]. These studies utilized a voltage clamped technique, which led to the conclusion that there exists independent pathways for sodium and potassium in the cell membrane [66].

Around the same time, Hodgkin was working on another study where he stimulated crustacean nerves using constant current stimuli. In this work he identified different classes or types of neuron excitations. Type 1 neurons fired with a wide range of frequencies that depended upon the strength of the stimulus current. Neurons of type 2 fired with a narrow range of frequencies that were relatively insensitive to the stimulus strength. Type 3 neurons that had a higher threshold either failed to repeat or fired once or twice only when the threshold was exceeded [67].

Following the studies on the crustacean nerves, in 1952, Hodgkin and Huxley published a series of papers that described a physiological model which simulates type 2 behavior of a giant squid axon. Hodgkin and Huxley were awarded the 1963 Nobel prize in Physiology and medicine for this outstanding work.

Hodgkin and Huxley publish a series of five papers in 1952 describing their findings. The first four papers summarized the experimental technique they've used to characterize the membrane potentials [68, 69, 70, 71]. The fifth paper put the experimental data into an extensive theoretical framework [72].

The model represented the nerve membrane as an electrical network. The membrane was considered to be an insulator and its capacitance was modeled using a capacitor with fixed capacitance. The proteins that control the flow of ions or the ion pathways were modeled using resistors. Figure 4.7 shows the electrical network presented in their work [72].

The three parallel resistors in the model represent the ion pathways as mentioned before. Out of the three resistors, the two R_{Na} and R_K are ion specific for Na^+ and K^+ respectively. The third one R_L represents the leakage phenomena that is generated by the unspecified ions. Modern extended versions of the Hodgkin-Huxley model specify extra resistors to represent the Ca^+ and Cl^- ions as well. Although this extra description allows an accurate representation of the membrane, the conceptual basis of today's modern models is perfectly aligned with the original Hodgkin-Huxley

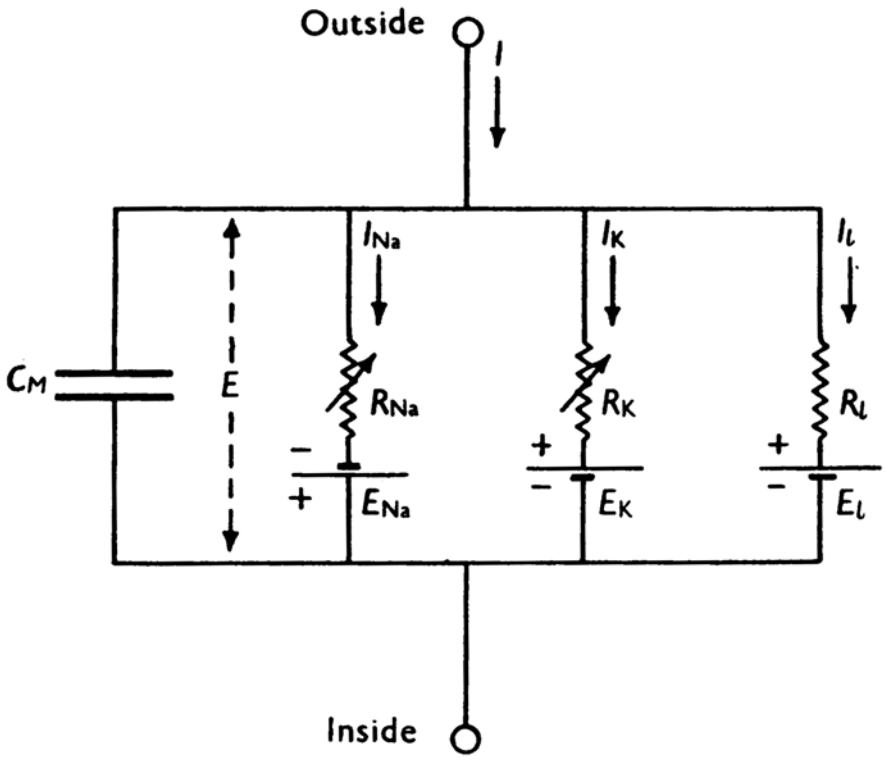


Figure 4.7. Electrical model of the neuron membrane [72].

version.

Referring to the figure 4.7, the total current passing through the membrane can be taken as the summation of the membrane current and the ionic current (4.5).

$$I = C_M \frac{dV}{dt} + I_i \tag{4.5}$$

where,

I is the total membrane current density (inward current positive)

I_i is the ionic current density (inward current positive)

V is the displacement of the membrane potential from its resting value (polarization positive)

C_M is the membrane capacitance

t is time

The ionic current I_i can be further subdivide into current components carried by

Na^+ ions (I_{Na}), K^+ ions (I_K) and other unspecified ions (I_L).

$$I_i = I_{Na} + I_K + I_L \tag{4.6}$$

The individual ionic currents can be represented in terms of the ionic conductances (g_{Na}, g_K and $\bar{g}_L$).

$$I_{Na} = g_{Na}(E - E_{Na}) \tag{4.7a}$$

$$I_K = g_K(E - E_K) \tag{4.7b}$$

$$I_L = \bar{g}_L(E - E_L) \tag{4.7c}$$

where, E_{Na} and E_K are the equilibrium potentials for the Na^+ and K^+ ions. E_L is the potential at which the leakage current due to unspecified ions is zero. The equilibrium potentials can be calculated using the Nernst equation (4.3). For application purposes it is more convenient to write the equation (4.7) as follows.

$$I_{Na} = g_{Na}(V - V_{Na}) \tag{4.8a}$$

$$I_K = g_K(V - V_K) \tag{4.8b}$$

$$I_L = \bar{g}_L(V - V_L) \tag{4.8c}$$

where,

$$V = E - E_r \tag{4.9a}$$

$$V_{Na} = E_{Na} - E_r \tag{4.9b}$$

$$V_K = E_K - E_r \tag{4.9c}$$

$$V_L = E_L - E_r \tag{4.9d}$$

where, E_r is the absolute value of the resting potential. V, V_{Na}, V_K and V_L are the displacements of the corresponding potentials from the resting potential.

From the data collected via their voltage clamp experiments, Hodgkin and Huxley proposed the following equations to model the behavior of g_{Na} and g_K.

$$g_{Na} = \bar{g}_{Na} m^3 h \tag{4.10a}$$

$$g_K = \bar{g}_K n^4 \tag{4.10b}$$

The two new equations (4.10) introduced three new functions m, n and h that are dependant on membrane potential and time. These three functions are called gating variables and may be given a physical meaning under some assumptions. Given a physical basis, m, n and h are identified as potassium channel activation, sodium channel activation and sodium channel inactivation respectively. Each of the gating variables obeys a linear differential equation and is written as follows.

$$\frac{dm}{dt} = \alpha_m(1-m) - \beta_m m \tag{4.11a}$$

$$\frac{dn}{dt} = \alpha_n(1-n) - \beta_n n \tag{4.11b}$$

$$\frac{dh}{dt} = \alpha_h(1-h) - \beta_h h \tag{4.11c}$$

The functions α_m, α_n and α_h and β_m, β_n and β_h are called the transfer rate constants and are dependant on the membrane potential. As mentioned before, they cannot be derived from theory and are fitted to the experimental data. Each of the rate constants requires a several parameters to get a perfect fit to the experimental data. Therefore, it can be said that the Hodgkin-Huxley model is a parametrization of the electrical features of the cell membrane.

Although the equations (4.10) and (4.11) may be given a physical meaning under some assumptions, the two particular equations were chosen among many possible

solutions that could fit the data from the experiments performed by Hodgkin and Huxley. This implies that the two equations lack a real physical basis.

The transfer rate constants introduced in the equation (4.11) are given by the equations (4.12) and (4.13) in the original paper [72].

$$\alpha_m = \frac{0.1(V+25)}{\exp\frac{V+25}{10} - 1} \tag{4.12a}$$

$$\alpha_n = \frac{0.01(V+10)}{\exp\frac{V+10}{10} - 1} \tag{4.12b}$$

$$\alpha_h = 0.07 \exp\frac{V}{20} \tag{4.12c}$$

$$\beta_m = 4 \exp\frac{V}{18} \tag{4.13a}$$

$$\beta_n = 0.125 \exp\frac{V}{80} \tag{4.13b}$$

$$\beta_h = \frac{1}{\exp\frac{V+30}{10} + 1} \tag{4.13c}$$

In the model equations, potentials are given in mV, current densities in $\mu A/cm^2$, conductances in $m \cdot moh/cm^2$, capacity in $\mu F/cm^2$ and time in $msec$.

4.4.1 The propagating action potential

The Hodgkin-Huxley model described in the section 4.4 is static description where a uniform membrane potential is predicted at each instant over the entire length of the axon. Therefore no current along the axon can be seen and the net membrane current remains zero. The form of the action potential for a stimulus at $t = 0$ can therefore be calculated by solving (4.14) with $I = 0$, the initial condition $V = V_0$ and

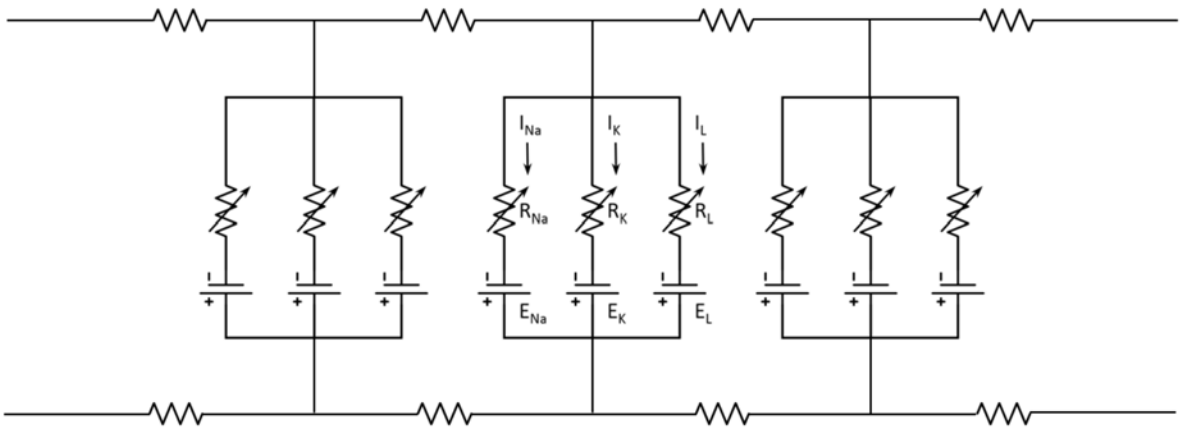


Figure 4.8. One dimensional electrical cable model of the squid giant axon. [74].

the steady state values of the m, n and h.

$$I = C_M \frac{dV}{dt} + \bar{g}_K n^4 (V - V_K) + \bar{g}_{Na} m^3 h (V - V_{Na}) + \bar{g}_L (V - V_L) \tag{4.14}$$

A dynamical extension to the static model can be obtained by combining the known Hodgkin-Huxley model with cable theory [73, 74]. One dimensional cable model of the squid giant axon is illustrated in the figure 4.8.

The cable model provides a relationship between the membrane current and the spacial variation of the membrane potential (4.15).

$$i = \frac{1}{r_1 + r_2} \frac{\partial^2 V}{\partial x^2} \tag{4.15}$$

where, i is the membrane current per unit length, r_1 and r_2 are the internal and external resistances per unit length and x is the distance along the axon. r_1 can be neglected in such situations where the axon is surrounded by a large volume of conducting fluid and $r_2 \gg r_1$. Therefore, the equation can be re-written as,

$$i = \frac{1}{r_2} \frac{\partial^2 V}{\partial x^2} \tag{4.16}$$

or

$$I = \frac{a}{2R}\frac{\partial^2 V}{\partial x^2} \tag{4.17}$$

where, I is the membrane current density, a is the radius of the axon and R is the specific resistance of the intracellular space. Substituting (4.17) in (4.14), the equation (4.18) can be obtained [74].

$$\frac{a}{2R}\frac{\partial^2 V}{\partial x^2} = C_M\frac{\partial V}{\partial t} + \bar{g}_K n^4(V - V_K) + \bar{g}_{Na} m^3 h(V - V_{Na}) + \bar{g}_L(V - V_L) \tag{4.18}$$

During steady state propagation, the membrane potential against time at any given point is similar in shape to that of the membrane potential against the distance at any one time. Therefore, it can be written that,

$$\frac{\partial^2 V}{\partial x^2} = \frac{1}{\theta^2}\frac{\partial^2 V}{\partial t^2} \tag{4.19}$$

where t is the velocity of conductance. Combining (4.18) and (4.19) one can change the above partial differential equation to an ordinary differential equation (4.20).

$$\frac{a}{2R\theta^2}\frac{d^2 V}{dt^2} = C_M\frac{dV}{dt} + \bar{g}_K n^4(V - V_K) + \bar{g}_{Na} m^3 h(V - V_{Na}) + \bar{g}_L(V - V_L) \tag{4.20}$$

The equation (4.18) or (4.20), along with the equation (4.11) are often referred to as the complete Hodgkin-Huxley equations for the propagating action potential.

4.4.2 Modeling in MATLAB

The implementation of a complex model in a simulation environment is not an easy task, specially when it involves solving multiple differential equations. Therefore, the

model was first implemented in the MATLAB environment. Since it is quite complicated to solve partial differential equations, it was decided to test the static model in MATLAB and implement the dynamic model with partial differential equations in COMSOL, once the MATLAB tests are completed.

4.4.2.1 Runge-Kutta Method

The central part of modeling the Hodgkin-Huxley model using a computational tool is to solve the differential equations. There are multitude of numerical methods available which can be used to solve differential equations. Each of them have their pros and cons and the user has to determine which method is best for the application. Sometimes the computational tool comes equipped with its own unique set of tools that can be used as well.

Although, a method such as the Euler method can be used to solve the differential equations, the order of the error in the solutions is relatively high and the solution obtained is often considered unacceptable. Therefore, it was decided to employ a Runge-Kutta method as it offers a good balance between the computational speed and the accuracy.

The Runge-Kutta methods are a family of numerical methods which include the Euler routine for the approximation of ordinary differential equations. The most widely used method in the family is the fourth order Runge-Kutta method which is generally referred to as the **RK4** method. The method utilizes the first five terms of the Taylor series to derive the relevant equations[75].

Assume that the problem in hand is a first order differential equation with a known initial condition as shown below,

$$\frac{dy}{dx} = f(x, y) \qquad \text{with} \qquad y(0) = y_0$$

Given the size of the time step is h, the first five terms of the Taylor series can be

written as:

$$y_{i+1} = y_i + f(x_i, y_i)h + \frac{1}{2!}f'(x_i, y_i)h^2 + \frac{1}{3!}f''(x_i, y_i)h^3 + \frac{1}{4!}f'''(x_i, y_i)h^4$$

The Runge-Kutta method suggests that, the above equation can be represented in a form,

$$y_{i+1} = y_i + (a_1k_1 + a_2k_2 + a_3k_3 + a_4k_4)h$$

where, the constants are found to be,

$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)h$$

and,

$$\begin{aligned} k_1 &= f(x_i, y_i) \\ k_2 &= f(x_i + \frac{1}{2}, y_i + \frac{1}{2}k_1h) \\ k_3 &= f(x_i + \frac{1}{2}, y_i + \frac{1}{2}k_2h) \\ k_4 &= f(x_i + h, y_i + k_3h) \end{aligned}$$

4.4.2.2 MATLAB Simulation Results

Using the RK4 method described in the section 4.4.2.1, the static Hodgkin-Huxley model was solved. Listed below are the equations and constants used to solve the model [76].

$$\frac{dV}{dt} = \frac{1}{C_m}\left[I - \bar{g}_K n^4(V - V_K) - \bar{g}_{Na}m^3h(V - V_{Na}) - \bar{g}_L(V - V_L)\right] \tag{4.21}$$

and,

$$E_{Na} = 55.17mV \qquad \bar{g}_{Na} = 1.2mS/cm^2 \tag{4.22a}$$

$$E_K = -72.14mV \qquad \bar{g}_K = 0.36mS/cm^2 \tag{4.22b}$$

$$E_L = -49.42mV \qquad \bar{g}_L = 0.003mS/cm^2 \tag{4.22c}$$

$$C_m = 0.01\mu F/cm^2 \tag{4.23}$$

$$\frac{dU}{dt} = \alpha_U(1-U) - \beta_U U \tag{4.24}$$

where, U is m, n or h. The α and β are given by,

$$\alpha_m = \frac{0.1(V+35)}{1-\exp\left(\frac{-(V+35)}{10}\right)} \tag{4.25a}$$

$$\alpha_n = \frac{0.01(V+50)}{1-\exp\left(\frac{-(V+50)}{10}\right)} \tag{4.25b}$$

$$\alpha_h = 0.07\exp\left(\frac{-(V+60)}{20}\right) \tag{4.25c}$$

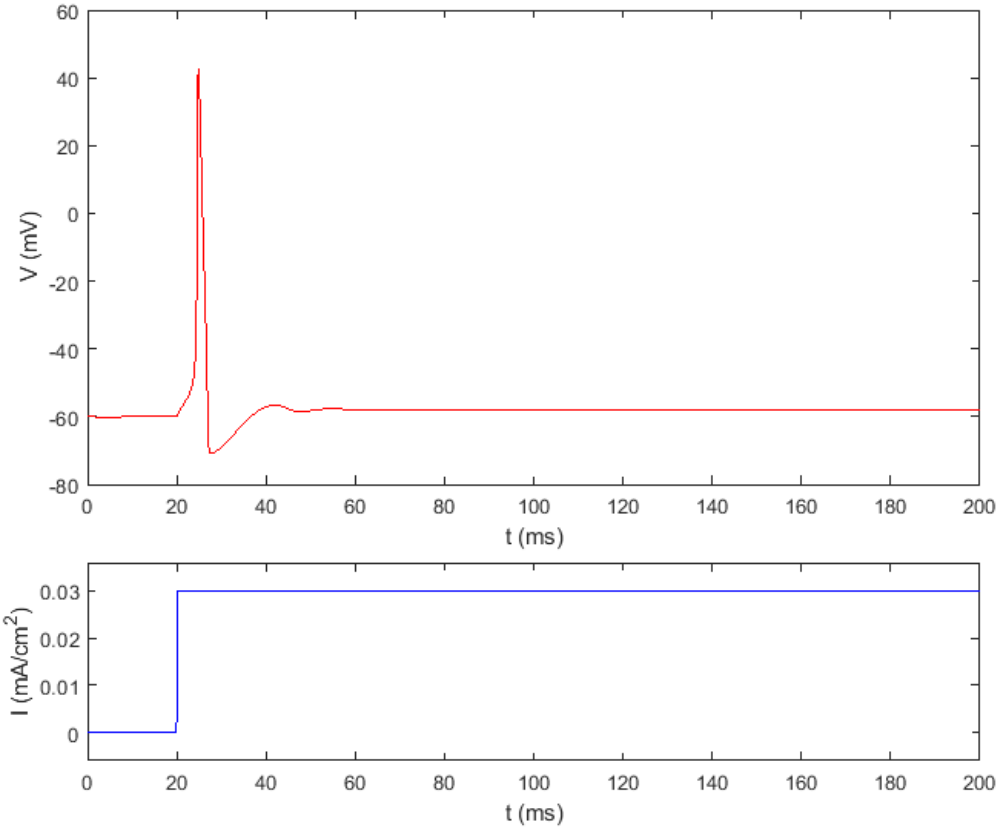
$$\beta_m = 4\exp\left(\frac{-(V+60)}{18}\right) \tag{4.26a}$$

$$\beta_n = 0.125\exp\left(\frac{-(V+60)}{80}\right) \tag{4.26b}$$

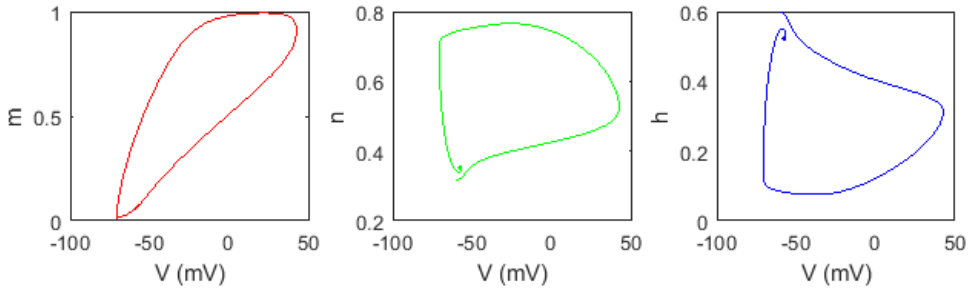
$$\beta_h = \frac{1}{1+\exp\left(\frac{-(V+30)}{10}\right)} \tag{4.26c}$$

The solution of the model provides the membrane potentials as well as the variations of the m, n and h values with respect to the membrane potential.

The figures 4.9 and 4.10 show the solutions obtained from the model. It can be seen from the results that the action potentials are generated in the axons for stimulus pulses above a certain threshold. Results also indicated that this threshold was about $0.03mA/cm^2$. Just above the threshold the model generates a single action potential spike after which the membrane potential returns to its resting value of $-60mV$ (fig. 4.9a). As the threshold of the stimulus increases a train of spikes are generated by

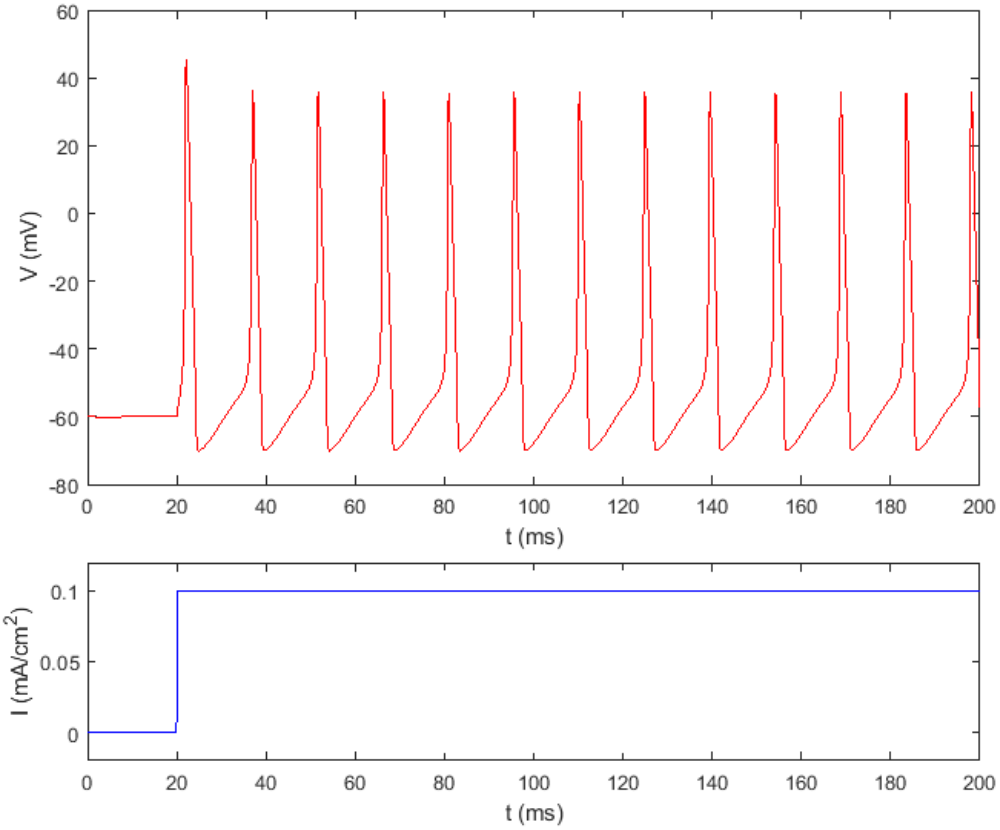


(a)

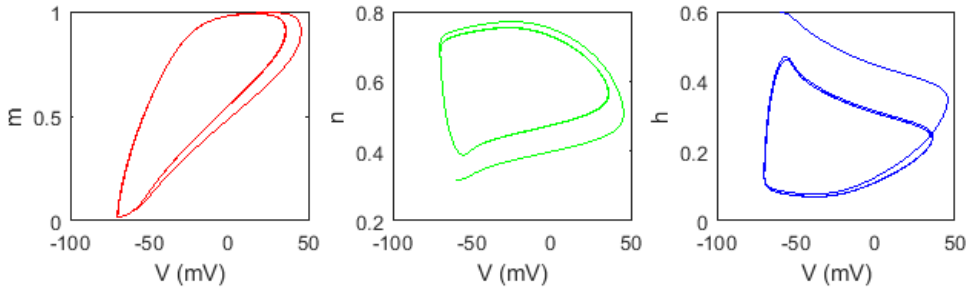


(b)

Figure 4.9. Single action potential spike obtained from the static Hodgkin-Huxley model. (a). Membrane potential and the external current that takes the shape of a step stimulus (b). Phase plots of the corresponding m, n and h values.



(a)



(b)

Figure 4.10. A train of action potential spikes obtained from the static Hodgkin-Huxley model. (a). Membrane potential and the external current that takes the shape of a step stimulus (b). Phase plots of the corresponding m, n and h values.

the model (fig. 4.10a). As the threshold of the stimulus was increased further, the frequency of the action potential spikes in the train of spikes was increased.

A nonlinear proportionality was observed between the stimulus threshold and the frequency of spikes in a selected range. The stimulus thresholds beyond the upper limit of the range produced a single peak that lacked the characteristics of the action potential spike. The proportionality was observed in the range $0.03mA/cm^2 < I < 2mA/cm^2$. In this range the frequency and the current was observed to have a relationship described by the equation (4.27) and is shown in the figure 4.11.

$$f = 141.76x^{0.29} \tag{4.27}$$

4.4.3 Modeling in COMSOL

4.4.3.1 Methodology

While it is important to understand the behavior of axons and neurons, scientists and researchers do not have direct access to isolate and stimulate neurons in living beings. Therefore, a variety of electrode arrays and high density micro-electrode arrays are being used to selectively activate and stimulate the nervous system. This requires optimizing the parameters of the stimulating device, which in-turn requires a thorough understanding of the effects of stimulation on biological systems. Two main stages can be identified in a typical system that model the external stimuli as well as the responses of the neurons. The first stage is the modeling of the electric field distributions in the muscles and tissues due to the external stimuli. The second is the computation of the responses of the target cells or the neurons. These two stages can be addressed by two main methodologies [77].

The computation of the electric field distribution in the extracellular space can be performed using a finite element analysis study. This allows taking into consideration the realistic geometry and the material properties such as permittivity and

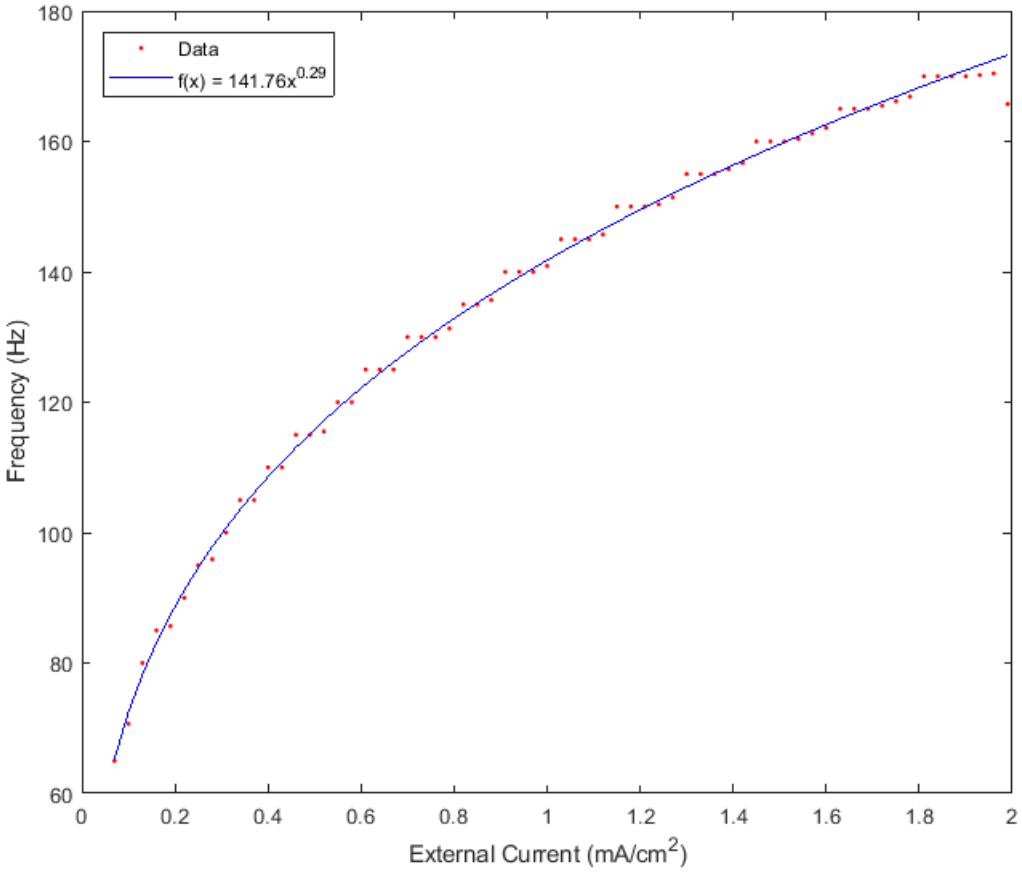


Figure 4.11. Frequency of the action potential spikes versus the external current stimulus threshold.

conductivity of the medium. The second stage of the process, the modeling of the neuron responses are typically performed in a framework that is separate from the finite element study environment [78, 79]. This approach is referred to as the the hybrid FEM-cable-equation approach. The NEURON software is a commonly used tool to simulate the cable equation based neuron models [80]. This method relies on two assumptions, the first, the membrane potential is only a function of the axial coordinates. Second, the effects on the extracellular potential due to the presence of the nerve fiber is negligible. However, these assumptions have been challenged in the literature [81]. An alternative method of calculating the neural responses in the same FEM environment as that used to calculate the extracellular potential has

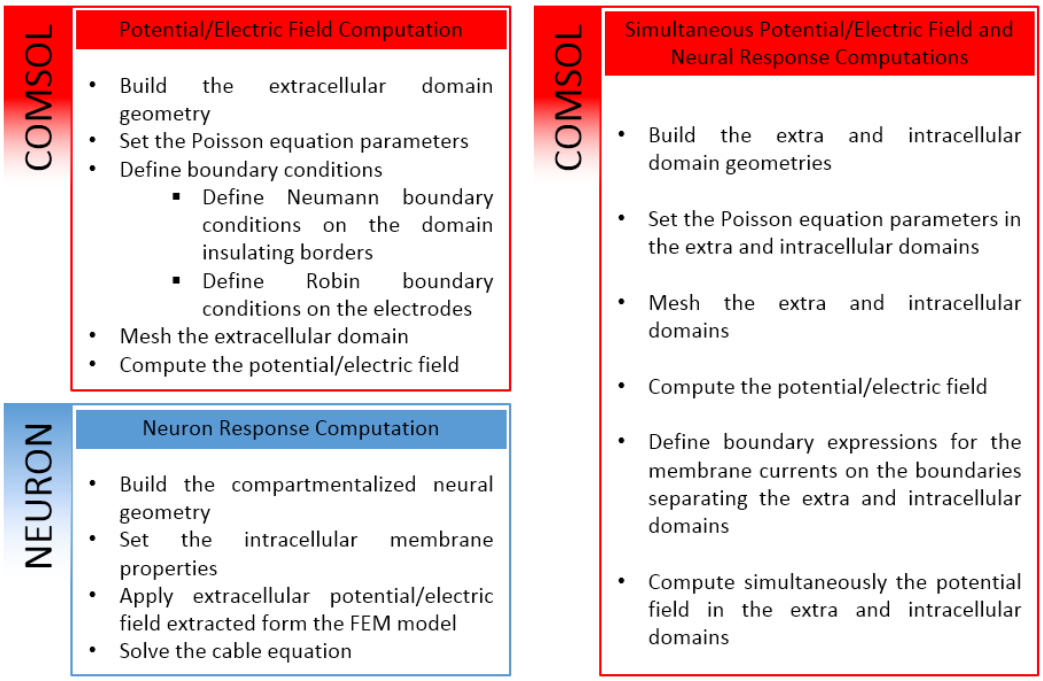


Figure 4.12. Two methodologies used to model effects on neurons due to external stimulation [77]. (a). Hybrid FEM-cable-equation approach and (b). Whole FEM approach.

proven valuable in recent studies [77, 82, 83]. This second method which does not depend on the above assumptions, uses only a finite element analysis environment and is identified as the whole FEM approach. COMSOL Multiphysics is a tool that is commonly used to perform the whole FEM approach. The two methodologies are summarized in the figure 4.12.

Due to the drawbacks of the hybrid FEM-cable-equation method and the steep nature of the learning curve of the NEURON software it was decided to use the whole FEM approach for this study.

As it is described in the outline of the whole FEM approach the process was started with building the geometry of the extra and intracellular domains. The 3D model of the spinal cord built in the chapter 3.2 could have been used as the extracellular domain of the geometry. However, due to the computational complexity of the procedure, a 2D geometry was created to represent a segment of the spinal

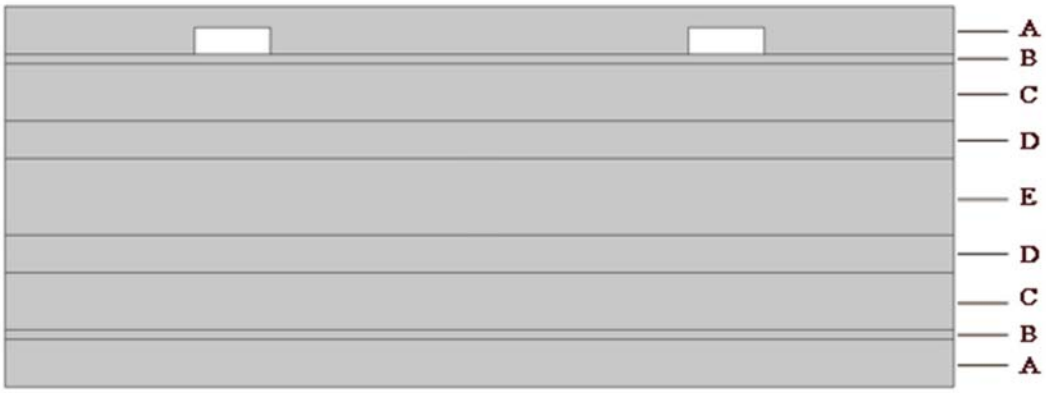


Figure 4.13. 2D model used for simulations representing the longitudinal section of the spinal cord with two embedded electrode; where A, B, C, D and E represent epidural fat, Dura matter, CSF, white matter and gray matter, respectively.

cord. To represent the intracellular domain or a single axon, a 20 mm long 1D structure was placed inside the model of the spinal cord segment. The 2D geometry of the extracellular domain was modeled referencing the 3D model described in the chapter 3.2. The model represented a longitudinal section of the spinal cord with two embedded electrodes (see figure 4.13).

Simulation of the behavior of the motor neurons under external stimulation is one of the main focuses of this study. These motor neurons known as alpha or gamma cells are located in the ventral horns of the spinal cord. Therefore the 1D structure that represents a motor neuron or a single motor axon was placed in the region that represents the ventral horns.

Material properties shown in the table 3.1 were used as the material properties for 2D spinal cord model. The physics of the simulation involves solving the simplified Maxwell's equations to compute the potential and electric field distributions in the 2D spinal cord model and solving the Hodgkin-Huxley cable equation in the 1D axon model simultaneously[83, 84]. The Dirichlet boundary conditions were applied to the two electrodes in the 2D model such that one of them was grounded while the other was used as the active electrode that provided the electric stimulus to the system. The simplified Maxwell's equation used to model the extracellular potential is shown

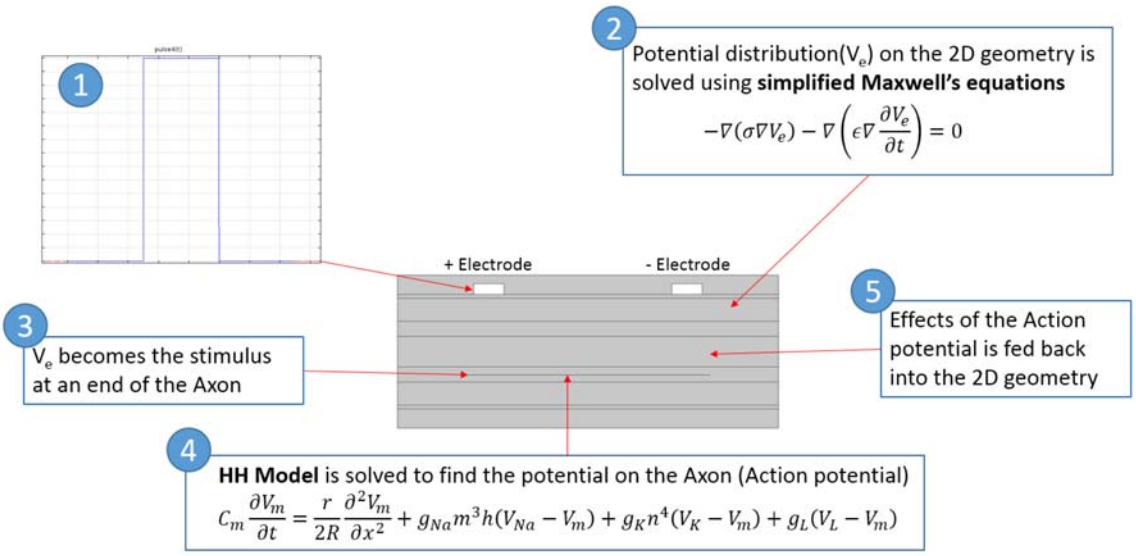


Figure 4.14. Summary of the steps of the whole FEM bi-domain simulation approach using COMSOL.

in the equation (4.28). The equations, (4.18), (4.22), (4.23), (4.24) and (4.25) were used to solve the membrane potential generated in the 1D motor axon.

$$-\nabla\left(\sigma\nabla V_e\right) - \nabla\left(\epsilon\nabla\frac{\partial V_e}{\partial t}\right) = 0 \tag{4.28}$$

The extracellular potential solved by the equation (4.28) is coupled to the intracellular potential by the equation (4.29), where V_e is the extracellular potential calculated from (4.28) and V_{rest} is the resting potential of the axon.

$$V = -V_e - V_{rest} \tag{4.29}$$

Once the physics were set, the model was meshed using a tetrahedral mesh with a finer mesh around the electrodes and a coarser mesh in the extracellular regions. The mesh consisted approximately 5500 domain elements and 800 boundary elements. The number of degrees of freedom solved for in the model was approximately 11500.

A summary of the approach used to solve the bi-domain model is shown in the figure 4.14. The whole FEM approach allows the five main steps shown in the fig-

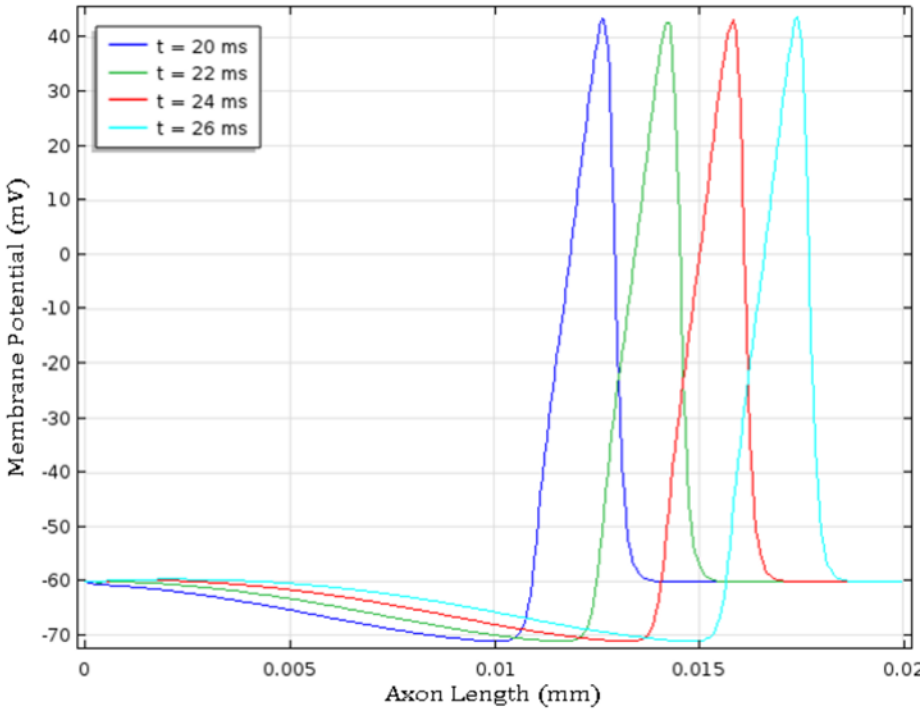
ure to be evaluated simultaneously. The main outcomes of the method include the ability to simulate the behavior of the axon for the external stimuli as well as the ability to simulate the effects on the extracellular domain due to the action potentials generated in the axons. This implies that the method is capable of simulating the electromyography (EMG) measurements.

Once the materials and physics were set, the model was simulated for a square pulse stimulus with a pulse width of $5ms$ and an amplitude of $15mV$ as shown in the step 1 of the figure 4.14. The radius and the resistivity of the axon was set to $5\mu m$ and $200\Omega \cdot cm$ respectively.

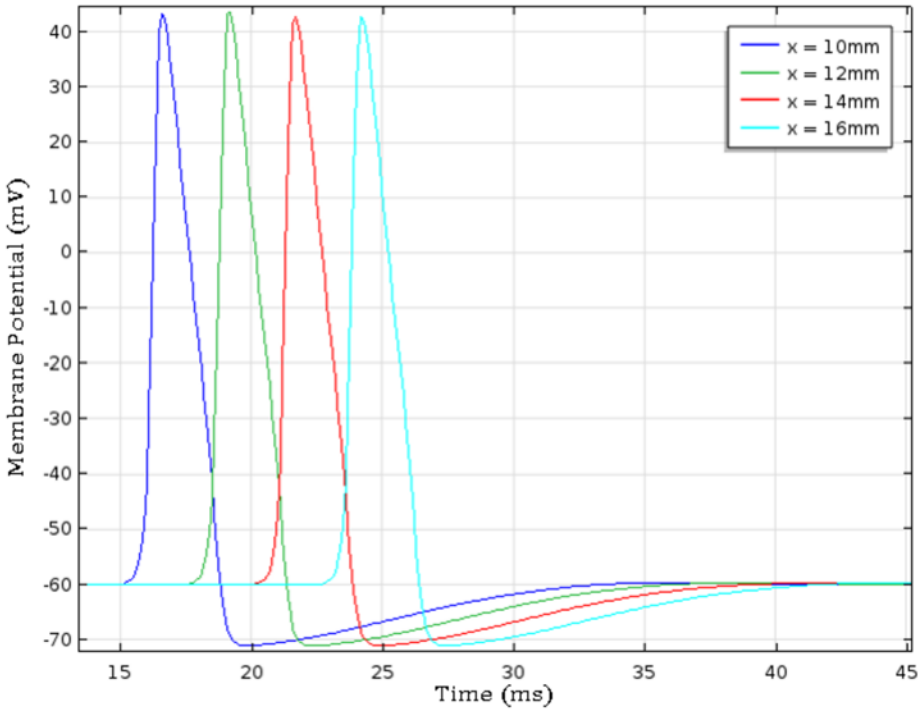
4.4.3.2 COMSOL Simulation Results

The results obtained from the simulations can be categorized in to three main categories. The effects on muscles or the extracellular region due to the external stimuli, the action potentials generated in the nerve axons or the intracellular region due to the external stimuli and the effects on muscles due to the action potentials and the external stimuli.

The results of the first category are similar to what was discussed in the chapters, 2.1.4 and 3.2. The action potential generated in the axon due to the external stimuli demonstrated that for stimuli above a threshold, nerve conduction and information transmission can be observed. The action potential information extracted from the simulation are shown in the figure 4.15. The role of the dendrites in the nerve axon is to catch the stimulation information received form an external stimuli or from axon terminals of a second axon. For this simulation, the left end of the 1D structure was set to receive the external input produced by the stimulus applied to one of the electrodes. Hence the left end of the 1D structure becomes the dendrite and the propagation direction is then set from left to right. It can be seen from the figure 4.15 that the propagation of information in the form of action potential indeed takes



(a)



(b)

Figure 4.15. Action potential obtained from the simulation. (a). Action potential at different points on the axon with respect to time. (b). Action potential along the axon at different time instances.

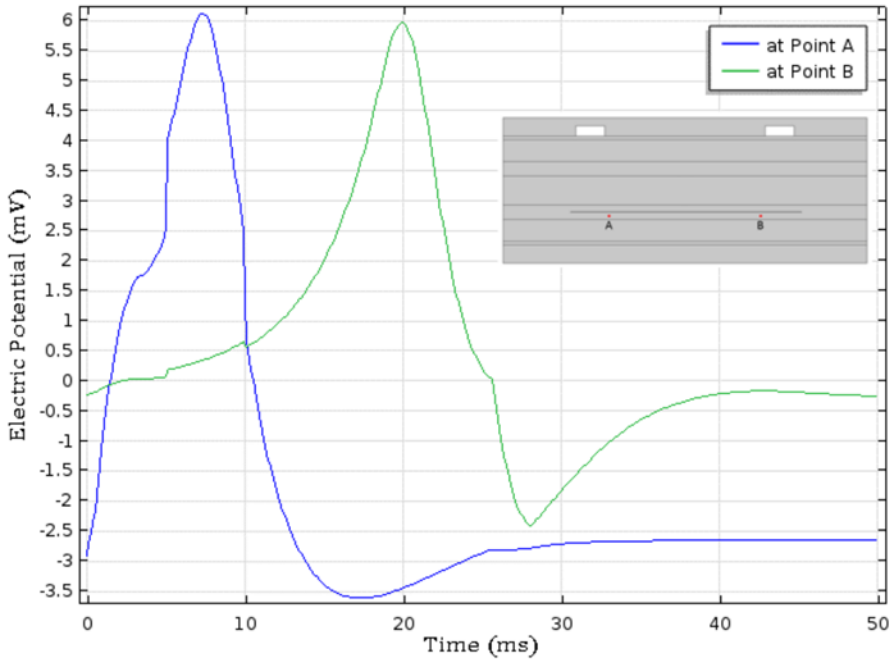


Figure 4.16. Effects on the extracellular region due to the action potential in the intracellular region

place from left to right as expected.

Figure 4.16 shows the effects on the extracellular region due to the action potential in the axon. Two data points were defined underneath the axon as shown in the figure and the electric potential due to the axon at these two points was extracted. The typical amplitudes of the EMG signals vary between $0 - 10mV$ (peak to peak) [85] which this simulation falls within. Further, EMG signals demonstrate specific features similar to action potentials[86]. Therefore, the extracellular potential results display similarities to the above features and show the typical EMG characteristics observed in the literature.

In Summary, the bi-domain model that utilizes a whole FEM approach provides the capability to solve the extracellular potential as well as the intracellular action potential due to external stimulation simultaneously. This method is preferred over the hybrid FEM-cable-equation approach since it does not make the assumptions that

were challenged in the literature[1].

4.5 Modeling Neurons - The Izhikevich Model

As it was discussed in the previous section, since it's inception, the Hodgkin-Huxley model has been used to study, understand and extend the knowledge on neural communication for over half a century. It's close relationship to the actual biology and chemistry of the neurons have been a driving factor for its wide use. Other researchers, mathematicians and scientists alike have put forward various other models to explain and model the behavior of the neurons. Among many of these models include, FitzHugh-Nagumo model [87], Hindmarsh Rose model [88] and the Wilson-Cowan model [89] just to name a few.

One of the major drawbacks of the classic Hodgkin-Huxley model over some of the other models listed above is it's inability to model bursting behavior of the neurons. Bursting can be defined as the repeated firing of groups of spikes in the neurons. The Hodgkin-Huxley model was developed to simulate the type 2 behavior of the neurons which generate action potential spikes in a narrow range of frequencies that are relatively insensitive to the stimulus amplitude [67]. However, there are neurons that demonstrate a bursting behavior which is distinct from the spiking patterns mentioned above. Neuron bursting has been observed and studied. The importance of the bursting behaviors have been discussed in the literature [90, 91, 92].

4.5.1 The Izhikevich Model

The Izhikevich model is a recently published simple mathematical model that is both computationally efficient and is capable of simulating variety of spiking as well as bursting patterns [93]. It has also been demonstrated that the Izhikevich model can be utilized to simulate a large number of neurons. Although the model has been

[1]Refer section 4.4.3.1

used to simulate the bursting patterns of the neurons, it has not been used in a FEM study to create a bi-domain model of the spinal cord. Hence, it was attempted to take the model described in the previous section one step further by extending its demonstrated capabilities in the literature. A bi-domain model with the Izhikevich model would be capable of simulating a variety of neurons including the ones that generate bursting patterns.

The model consists of two, two-dimensional ordinary differential equations as shown in the equation (4.30)

$$\frac{dv}{dt} = 0.04v^2 + 5v + 140 - u + I \tag{4.30a}$$

$$\frac{du}{dt} = a\,(bv - u) \tag{4.30b}$$

The model also contains an auxiliary condition (4.31) that resets the voltage spike.

$$\text{if} v \geq 30mV, \text{ then } \begin{cases} v \leftarrow c \\ u \leftarrow u + d \end{cases} \tag{4.31}$$

In (4.31), v and u are dimensionless variables where v represents the membrane potential of the nerve axon while u represents the membrane recovery variable. The parameters a, b, c and d are dimensionless parameters and are used to describe the time scale of the recovery variable u, the sensitivity of the recovery variable u, the after spike reset value of the membrane potential v and the after spike reset value of the recovery variable u respectively. The auxiliary condition is used to reset the variables v and u after the spike reaches its maximum ($+30mV$).

The membrane potential v has a unit of mV and the time t has has a unit of ms. The resting potential of the model depends on the value of the parameter b and can include a value between $-70mV$ and $-60mV$. The model does not have a fixed threshold potential. Therefore the threshold potential can reach a value as low as

$-55mV$ or as high as $-40mV$ depending on the history of the membrane potential prior to the spike.

4.5.2 Modeling in COMSOL

In order to perform simulations using the Izhikevich model, the same two dimensional model (see fig. 4.13) used in chapter 4.4.3 was used. The goal was to implement a model similar to the bi-domain model implemented previously using the Hodgkin-Huxley equations. Although it was possible to implement a bi-domain model using the Izhikevich model, it was complicated to implement the whole-FEM approach to solve the bi-domain model simultaneously for the extracellular potential and the intracellular potential because of the auxiliary condition in the Izhikevich model.

Implementing the auxiliary condition (4.31) involved extra steps of calculations, due to the nature of the Izhikevich model. This condition is used to reset the voltage spike after it reached a predefined maximum. This involved modifying the solutions of the differential equations (4.30) after the solution was computed. In order to perform this modification, the previous solutions of the differential equations must be accessed before the current step is computed. However, older versions of COMSOL did not allow the user to access these previous solutions. Throughout the simulation work mentioned in the previous chapters, various versions of COMSOL including $v4.4$, $v5.0$ and $v5.1$ have been used. In the latest version ($v5.1$) used in this work, COMOSL included a new feature called the previous solution operator which provided the ability to perform the very steps needed to implement the auxiliary condition. However, in order to utilize this feature the simulation of the model had to be broken into two steps to calculate the extracellular potential and intracellular potentials separately.

First, the extracellular potential in the 2D model was computed for a step potential stimulus and then the potential at the dendrite of the axon was extracted as a function of time. This voltage function was then applied to the Izhikevich model as the

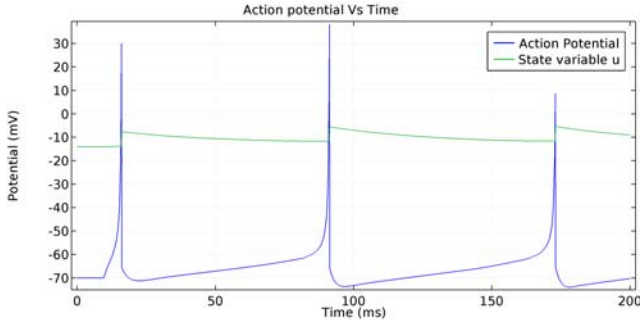
external stimulus to solve for the intracellular potential. Further, the Izhikevich model does not have the capability to calculate the propagation of the action potentials. Therefore having a geometry to represent a nerve axon became impractical. However, the simulation would provide the information on the activation status of selected axons.

4.5.3 COMSOL Simulation Results

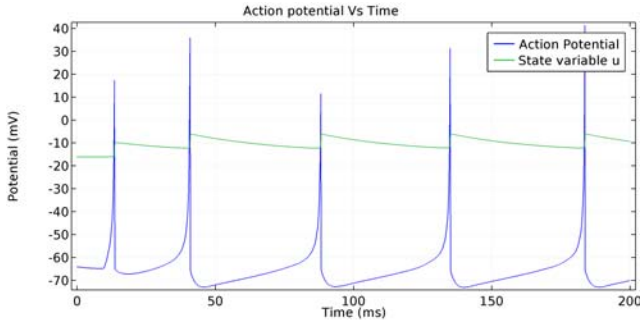
The bi-domain model was first solved in the 2D extracellular domain and the extracellular potential at the center point of the gray matter region was extracted as a function of time. This data was then applied to the Izhikevich model as the external stimulus. The Izhikevich model was solved multiple times for different model parameters to obtain different spiking and bursting patterns. The intracellular action potentials obtained in different simulation runs are shown in the figure 4.17. The different categories of spiking and bursting patterns are obtained by changing the values of 4 model parameters a, b, c and d between the runs. Different choices of the parameters give rise to different intrinsic firing patterns. These include some of the known types of neocortical and thalamic neurons [94, 95]. Parameter values used to generate the shown results are listed in the table 4.1.

Table 4.1. Values of the parameters a, b, c and d used to generate the firing patterns shown in fig. 4.17

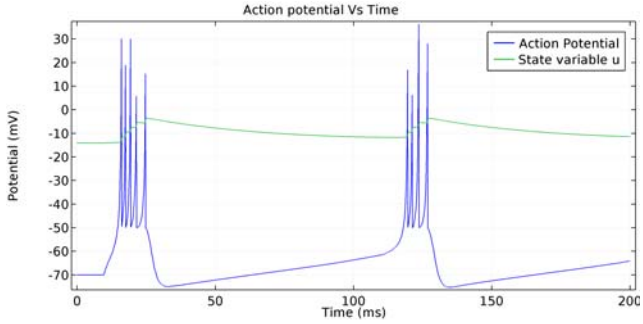
	Tonic Spiking	Phasic Spiking	Tonic Bursting	Phasic Bursting
a	0.02	0.02	0.02	0.02
b	0.20	0.25	0.20	0.25
c	−65.00	−65.00	−50.00	−55.00
d	6.00	6.00	2.00	0.05



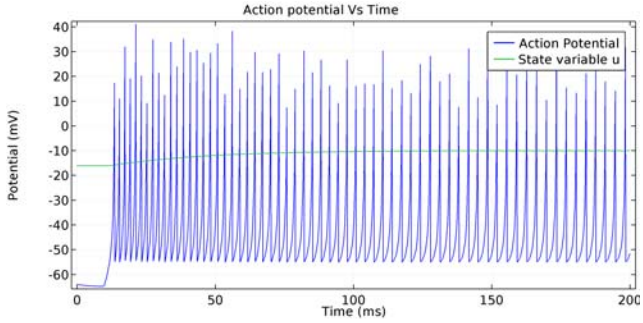
(a)



(b)



(c)



(d)

Figure 4.17. The spiking and bursting patterns obtained from the bi-domain simulations. (a). Tonic spiking (b). Phasic spiking (c). Tonic bursting (d). Phasic bursting

4.5.4 COMSOL Application

Multiple variables, different electrical stimuli, and the possibility of extracting time varying potentials at different point locations result in numerous possible variations in the simulations. The management of these options is addressed using a COMSOL application built with the COMSOL application builder utility.

The interface of the created application is shown in the figure 4.18. The application provides the ability to change all the variables, change the stimulation pulse, and change the location of the point at which the 2D potential is extracted.

The Izhikevich model has the ability to simulate a variety of spiking and bursting patterns. These patterns are a result of the values taken by the variables in Izhikevich model. Five of these variable value sets are also predefined within the application. The custom application also allows the user to easily navigate through the simulation process and visualize the resulting potential distributions and the action potentials. Figure 4.19 shows the simulation steps and the basic procedure of setting up a simulation using the created COMSOL application. The numbers 1, 2 and 3 shown in figure 4.19 shows the corresponding user input sections in the custom application shown in figure 4.18.

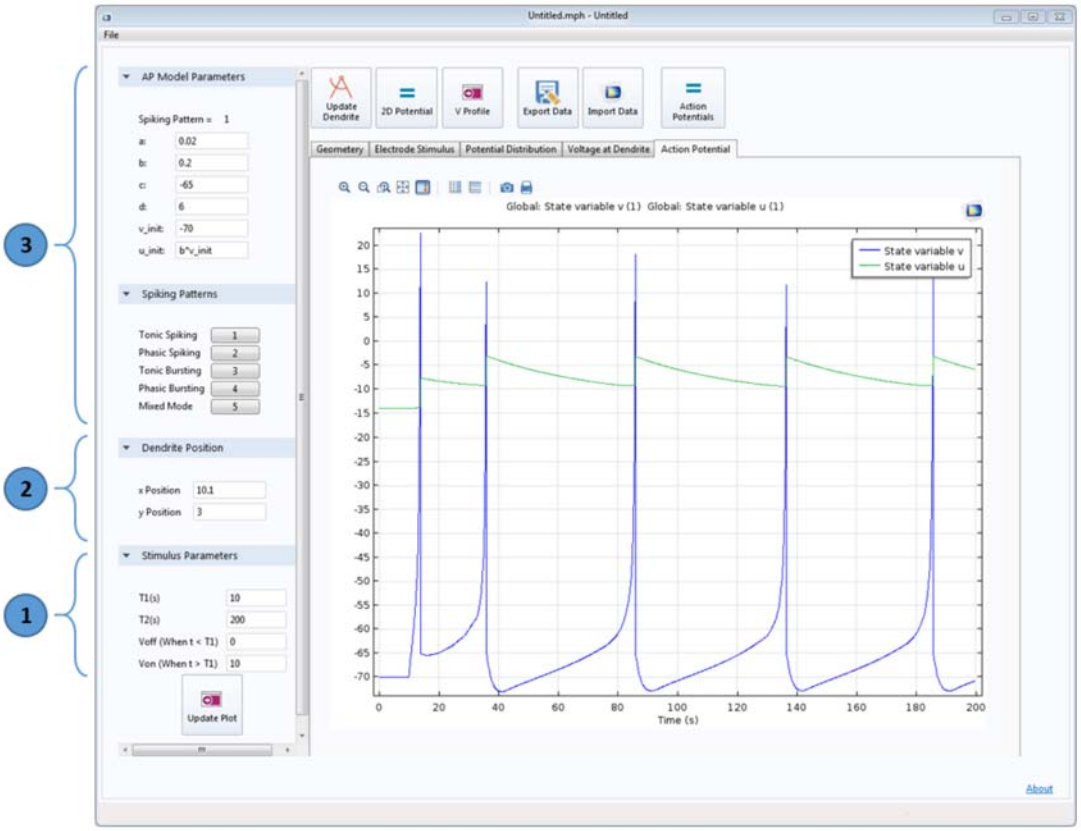


Figure 4.18. Interface of the COMSOL application.

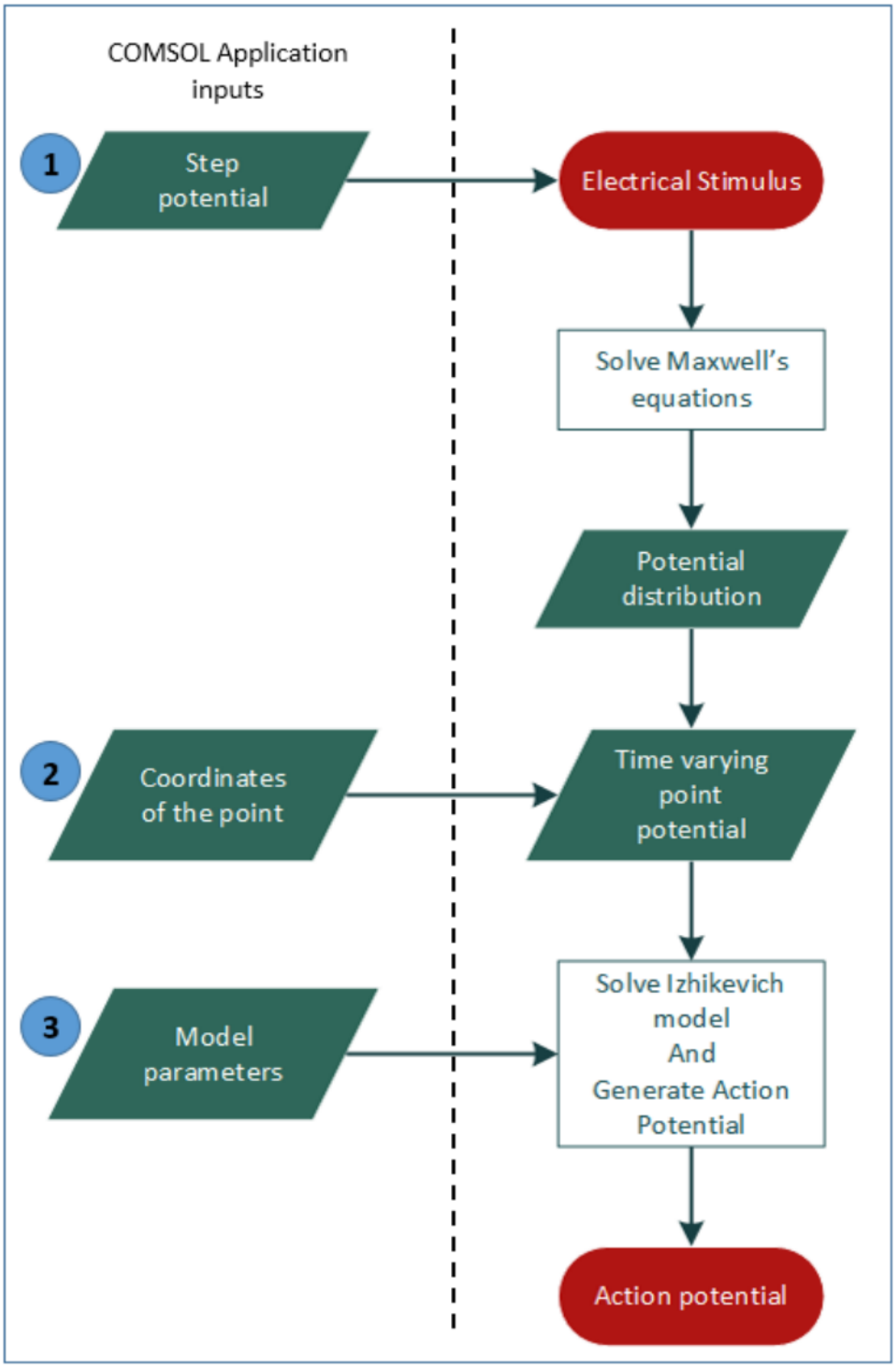


Figure 4.19. Flowchart of the simulation procedure using the COMSOL application.

CHAPTER 5

OPTIMIZING THE HODGKIN-HUXLEY MODEL

The Hodgkin-Huxley (HH) model is a widely used biophysically meaningful model that can simulate the action potentials in the nerve axons. It is mainly used to simulate the type 2 behavior of the axon firing[1]. However, other types of neuron spiking and bursting have been observed in the literature[2]. This chapter discusses a key novelty of this research, showing the optimization of the classic HH model parameters to simulate neuron bursting behavior. This has not been shown in previously in the literature. The results of the work discussed in this chapter demonstrates that it is possible to extend the HH model beyond its intended type 2 behavior and can be modified to simulate more complex neuron firing patterns including neuron bursting.

5.1 Optimizing the HH model

As it was discussed in the previous chapter, following the studies on crustacean nerves, Hodgkin and Huxley published a series of papers that describe a physiological model which simulates type 2 behavior of a giant squid axon. The complex model consists of a system of linear and non-linear differential equations along with a set of parametric equations that models the ion channels of the axon [68, 69, 70, 72]. For the last six decades, the HH model has been widely used to better understand the action potentials in the axons and the nervous system.

[1]Different types of neurons including type 2 neurons were discussed in section 4.4
[2]Neuron bursting was discussed in section 4.5

Although the original HH model is widely used, it lacks the ability to model the bursting behavior of the neurons as described in section 4.5. Even though there exists other models that simulate the bursting behavior as well as regular spiking, either they have simulation limitations or they are strictly mathematical and are not biophysically meaningful. It has been theorized that the HH model has the capability to simulate a wide variety of spiking and bursting patterns in addition to its intended type 2 behavior [96]. However, due to the computational complexity of the model, these capabilities have not been previously explored to improve and extend the model. In this work, it is demonstrated the possibility of extending the HH model by identifying defined constants in the model equations as variables that can be adjusted to get different spiking and bursting patterns, without adding any additional parameters to the original HH model.

The HH model consists of a system of differential equations (5.1) and (5.2) As described in section 4.4. The model also consists two sets of rate constants equations known as α (5.3) and β (5.4) equations.

$$\frac{dV_m}{dt} = \frac{1}{C_m}\left[I_{ext} - \bar{g}_K n^4 (V - V_K) - \bar{g}_{Na} m^3 h (V - V_{Na}) - \bar{g}_L (V - V_L)\right] \tag{5.1}$$

$$\frac{dU}{dt} = \alpha_U (1 - U) - \beta_U U \qquad \text{where, } U \text{ is } m, \, n \text{ or } h \tag{5.2}$$

$$\alpha_m = \frac{0.1(V + 35)}{1 - \exp\left(\dfrac{-(V + 35)}{10}\right)} \tag{5.3a}$$

$$\alpha_n = \frac{0.01(V + 50)}{1 - \exp\left(\dfrac{-(V + 50)}{10}\right)} \tag{5.3b}$$

$$\alpha_h = 0.07 \exp\left(\frac{-(V + 60)}{20}\right) \tag{5.3c}$$

$$\beta_m = 4 \exp\left(\frac{-(V+60)}{18}\right) \tag{5.4a}$$

$$\beta_n = 0.125 \exp\left(\frac{-(V+60)}{80}\right) \tag{5.4b}$$

$$\beta_h = \frac{1}{1+\exp\left(\frac{-(V+30)}{10}\right)} \tag{5.4c}$$

Simulation studies performed using the HH model in the sections 4.4.2 and 4.4.3 have shown that by changing the parameter values in (5.3) and (5.4), different membrane potentials can be produced. However, finding the parameters that result in a specific spiking or a bursting pattern can be computationally demanding and difficult. This chapter addresses this task of modeling the bursting behavior and finding the set of parameters to activate it.

The concept behind the optimization was to generalize equations (5.3) and (5.4) to increase the parameter space so that the solution space can be expanded. First, the constants in (5.3) and (5.4) that can be controlled and would affect the resulting membrane potential were identified. In this process, 23 constants, listed in table 5.1, were identified as tunable. Then, the identified 23 constants in the original HH model were transposed to variables that can be varied as shown in equations (5.5) and (5.6). Different sets of parameters could be assigned to these variables to obtain different spiking or bursting patterns. Equations (5.3) and (5.4) can be produced by substituting the original set of parameters from Table. 5.1 to the variables in (5.5) and (5.6).

$$\alpha_m = \frac{\alpha_{m1}(V_m + \alpha_{m2})}{\alpha_{m3} - e^{-\frac{V_m + \alpha_{m4}}{\alpha_{m5}}}} \tag{5.5a}$$

$$\alpha_n = \frac{\alpha_{n1}(V_m + \alpha_{n2})}{\alpha_{n3} - e^{-\frac{V_m + \alpha_{n4}}{\alpha_{n5}}}} \tag{5.5b}$$

$$\alpha_h = \alpha_{h1} e^{\alpha_{h2}(V_m + \alpha_{h3})} \tag{5.5c}$$

$$\beta_m = \beta_{m1} e^{\beta_{m2}(V_m + \beta_{m3})} \tag{5.6a}$$

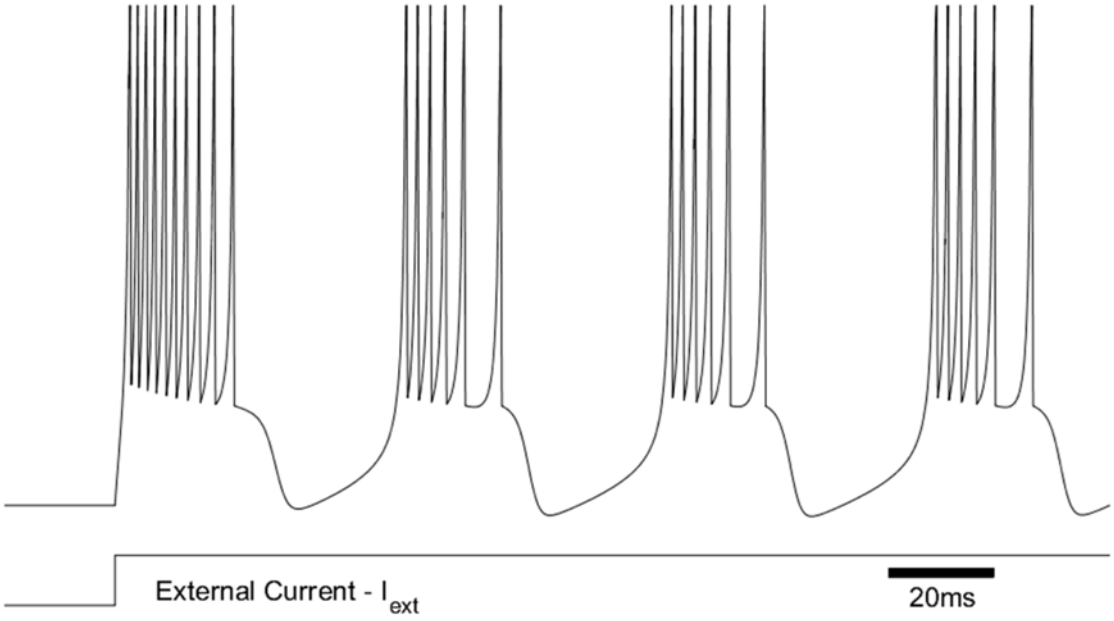
$$\beta_n = \beta_{n1} e^{-\frac{V_m + \beta_{n2}}{\beta_{n3}}} \tag{5.6b}$$

$$\beta_h = \frac{\beta_{h1}}{\beta_{h2} + e^{-\beta_{h3}(V_m + \beta_{h4})}} \tag{5.6c}$$

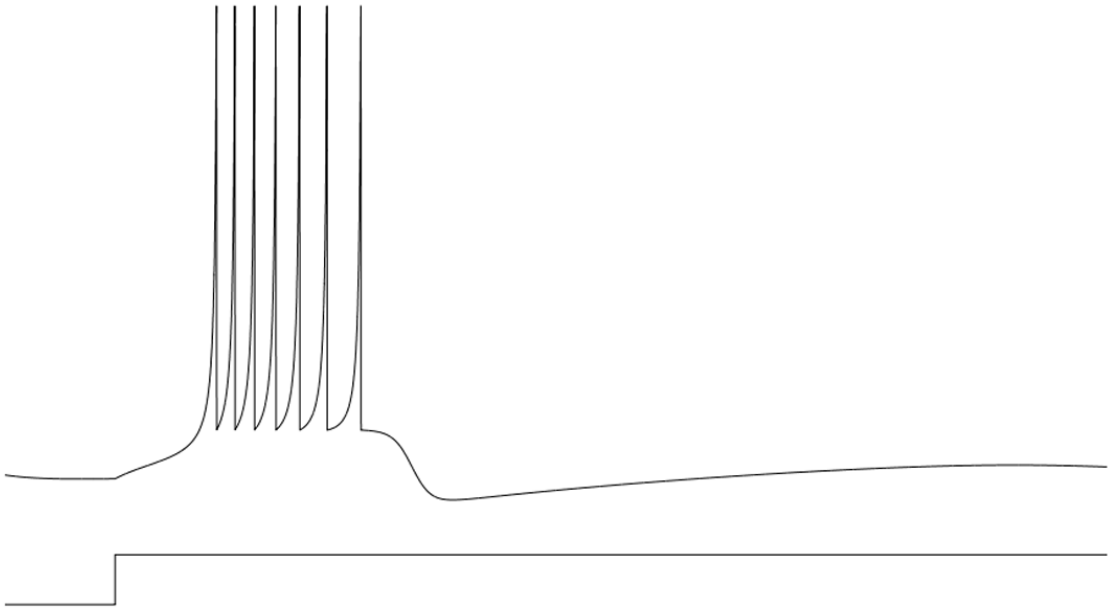
α_{mi}, α_{ni}, α_{hi}, β_{mi}, β_{ni} and β_{hi} represent the 23 parameters that were identified as tunable. External current I_{ext} that is applied to the system (*see* (5.1)) can also be treated as a tunable parameter that would affect the output. Therefore, the dimensionality of the optimization problem at hand was set to 24 to match target bursting patterns.

5.1.1 Target Bursting Patterns

In order to compute values for the identified parameters it was decided to use two target bursting patterns. Izhikevich identifies a large number of simulated spiking and bursting patterns in his 2004 paper [96]. The phasic bursting and the tonic bursting patterns were selected as the targets for this study. The simulated data for these two target bursting patterns shown in fig.5.1 were generated using the the mathematical model created by Izhikevich [93]. The goal of this work is to adapt these two target bursting patterns demonstrated by Izhikevich to the HH model because it has biophysical meaning and is not just a mathematical model.



(a)



(b)

Figure 5.1. Two target bursting patterns used in the study. (a). Tonic bursting. (b). Phasic bursting. Each plot shows the action potential response to an external step current input. Time resolution is $0.04ms$.[93].

Table 5.1. Parameter values that produce the original rate constants equations (5.3) and (5.4) form the generalized rate constants equations (5.5) and (5.6)

Parameter	Value
α_{m1}	0.100
α_{m2}	35.000
α_{m3}	1.000
α_{m4}	35.000
α_{m5}	10.000
α_{n1}	0.010
α_{n2}	50.000
α_{n3}	1.000
α_{n4}	50.000
α_{n5}	10.000
α_{h1}	0.070
α_{h2}	-0.050
α_{h3}	60.000
β_{m1}	4.000
β_{m2}	-0.056
β_{m3}	60.000
β_{n1}	0.125
β_{n2}	60.000
β_{n3}	80.000
β_{h1}	1.000
β_{h2}	1.000
β_{h3}	0.100
β_{h4}	30.000

5.2 Parameter Optimization and Fitness Function

The word *optimization* comes from the root *optimum* which means the best or most favorable point. Therefore when an optimization is performed on a mathematical problem, the result is the selection of the best elements from an ensemble of alternative elements. In the most simplest case, the optimization can be either maximizing or minimizing a function of the mathematical problem by choosing values for the function from a given domain. Mathematically, this can be written as,

$$
\begin{aligned}
\text{given: a function} \quad & f : A \to R \\
\text{find elements} \quad & \mathbf{x}_0 \in A \\
\text{where} \quad & f(\mathbf{x}_0) \leq f(\mathbf{x}) \text{ for all } \mathbf{x} \in A \\
& \text{or} \\
& f(\mathbf{x}_0) \geq f(\mathbf{x}) \text{ for all } \mathbf{x} \in A
\end{aligned}
$$

This mathematical formulation is identified as the definition of an optimization of a mathematical function. Here, the domain A is some subset of the real numbers R and known as the search space of the function f.

The function f is identified by different names such as, the objective function, the lost function, the cost function, the error function, the fitness function, etc. An optimum solution would be the set of values from A that minimizes (or maximizes) the function f.

The error function for this problem was chosen to indicate a measure of how close a resulting HH membrane potential pattern was to one of the targets we have chosen in section (fig. 5.1). In this case, the error function was the relative difference between the target pattern and the pattern that was obtained by modifying the HH parameters.

The calculation of the relative error was done in parts. The mean squared error between the reference data set and the obtained data set from the modified HH model was the major component of this calculation and is shown in (5.7).

$$
e = \frac{1}{q} \sum_{i=1}^{q} (V_{mi} - V_{refi})^2 \tag{5.7}
$$

where V_{mi} is the membrane potential obtained from the Hodgkin-Huxley model, V_{refi} is the reference potential shown in fig. 5.1 and q is the number of time samples in the V_m data array.

This problem falls into the category of constrained optimization because there are large sections of the solution space that carry infeasible solutions. Therefore, to avoid having these infeasible solutions in the population and in-turn to improve the effectiveness of the optimization process, additional penalties were attributed to the error function by comparing features of V_m with V_{ref}. These features include the number of local and global extrema and the values and the positions of the extrema.

Once the error function or the fitness function is defined, a selected optimization can be used to minimize of maximize the function. Since the error function indicates the difference between the reference and calculated bursting patterns, the error function increases as the two patterns differ from each other. Therefore, in this case the error function needed to be minimized to find the optimal solution. The next section discusses some of the optimization methods that can be used to optimize an error function.

5.3 Different Optimization Techniques

Mathematical optimization of functions is a whole branch of study which concentrates solely on different optimization techniques and methods to find the best values for the objective functions. Before deciding on a specific optimization technique to solve this problem, different options were considered. Some of the methods considered and employed are described below.

5.3.1 Brute Force Method

One of the simplest methods that can be employed to solve problems is the brute force[3] approach. Since this method lacks a mathematical foundation, it is often identified as the *naive* brute force approach. Regardless of it's lack of mathematical basis this method can be useful in certain situations to explore the problem while a better

[3]Brute Force is also known as the proof by exhaustion

method of optimization is implemented. Although brute force is useful, it is computationally extremely expensive and very inefficient since the method evaluates each and every possible solution in the parameter space to figure out the best out of them. For instance, if this method was to be employed for this particular problem, and if it was assumed each of the tunable parameters could take four possible values, the number of permutations the solver would have to go through is $4^{23} > 70\ Trillion$. Even if a computer could go through all of these 70 trillion permutations in a reasonable amount of time, it cannot be guaranteed that four discrete values would provide the optimum solution to the problem.

5.3.2 Random Search

Randomness is often used in mathematics to solve problems that are deterministic in nature. Monte Carlo methods are a whole class of algorithms that are based on randomness. The optimization methods that generate and use random variables are known as the *stochastic optimizations*. Random search is one of the simplest stochastic optimization algorithms used commonly in mathematics. The random search is considered to be a better option compared to the brute force approach. Therefore, a random search algorithm was employed in the initial stage of the optimization process. The random search algorithm used in this work can be summarized as follows.

Step 1. Define the parameter space by setting up upper and lower limits for the parameters.

Step 2. Generate a random set of parameters selecting values from the parameter space.

Step 3. Evaluate the error function for the selected random parameter set.

Step 4. Stop the optimization if the stopping criteria is met. Otherwise return to Step 2.

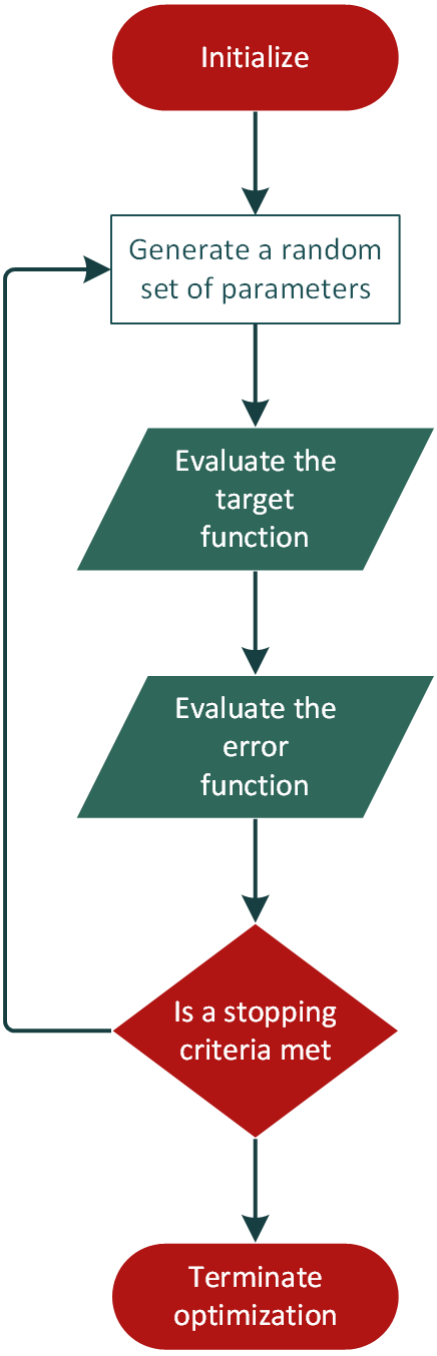


Figure 5.2. Flow chart of the random search algorithm.

A flowchart of the above algorithm is shown in the figure 5.2.

One of the major factors that determines if the random search can ever find an optimum result is the definition of the search space. To have a non-zero probability of finding the optimum result, the optimum parameters must be within the param-

eter search space. However, a mathematical way to guarantee the existence of the optimum result within the parameter space is non-existent. Therefore, to improve the probability of finding a desired outcome, several random search simulations must be run with varying search spaces between them.

A custom Matlab program was developed to simplify the process of running multiple simulations with varying parameter spaces. The GUI of the software (fig. 5.3) allows the user to update the parameter space easily and run a random search simulations. The software allows the user to change lower and upper limits of the search space using few different ways. The user can either type in the desired limits or set the limits based on a percentage difference from a reference values. The user can also choose to save the intermediate results for future reference. The saved intermediate results show the output for a given set of parameters as shown in the figure 5.4).

Multiple instances of the random search optimizations were run alongside other optimizations. Although this method did not yield any valuable results, the intermediate results provided some insight in to the limits of the parameters.

5.3.3 Gradient Descent

Gradient descent is another commonly used optimization algorithm which is a first-order iterative[4] algorithm that is used to find the extrema (usually a minimum) of a function. To find a local minima of a function, the gradient of the function is evaluated for a given set of parameters and then the parameters are shifted in the direction of the negative gradient. Repeating this procedure guarantees the discovery of the local optimum of the target function. However, it cannot be certain that the global optimum of the function can be found using this method.

In general, gradient descent algorithms are capable of finding optimum points of a function. These optimum points include minima, maxima or inflection points. If

[4] "An iterative method is a mathematical procedure that uses an initial guess to generate a sequence of improving approximate solutions." - *Wikipedia*

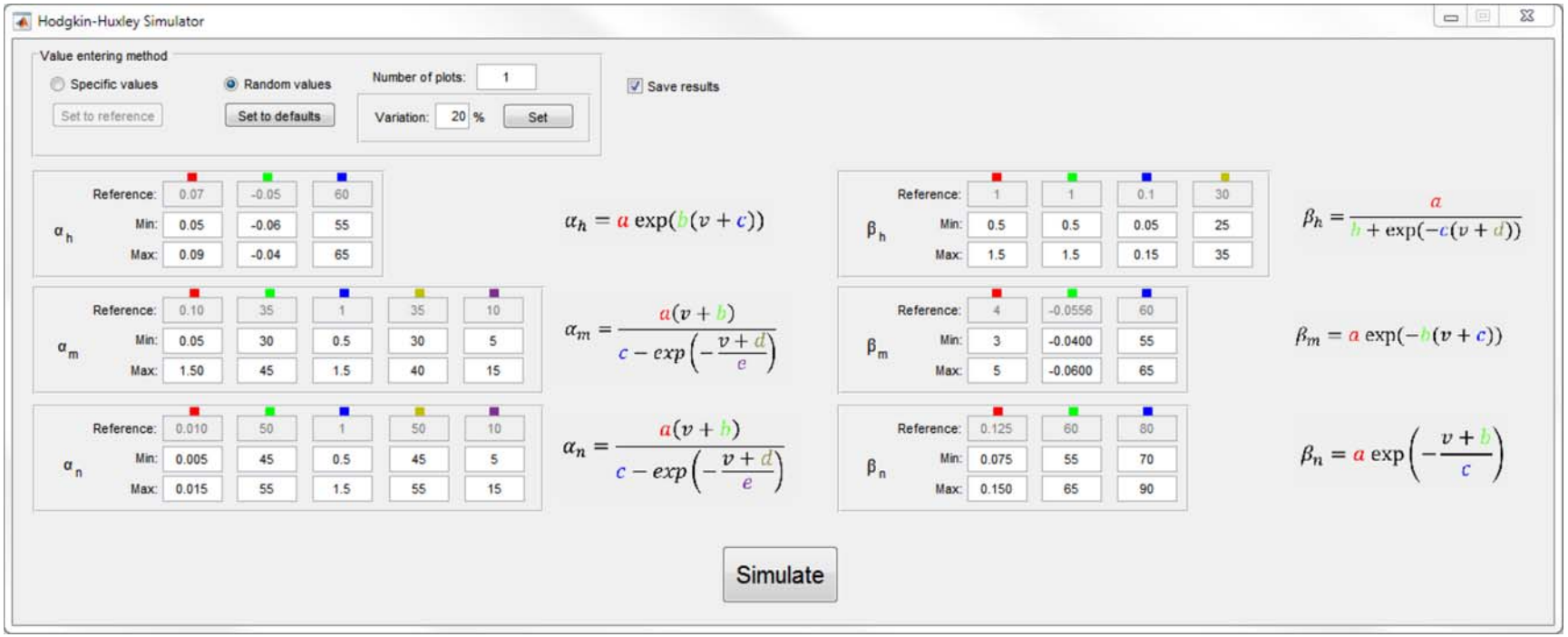
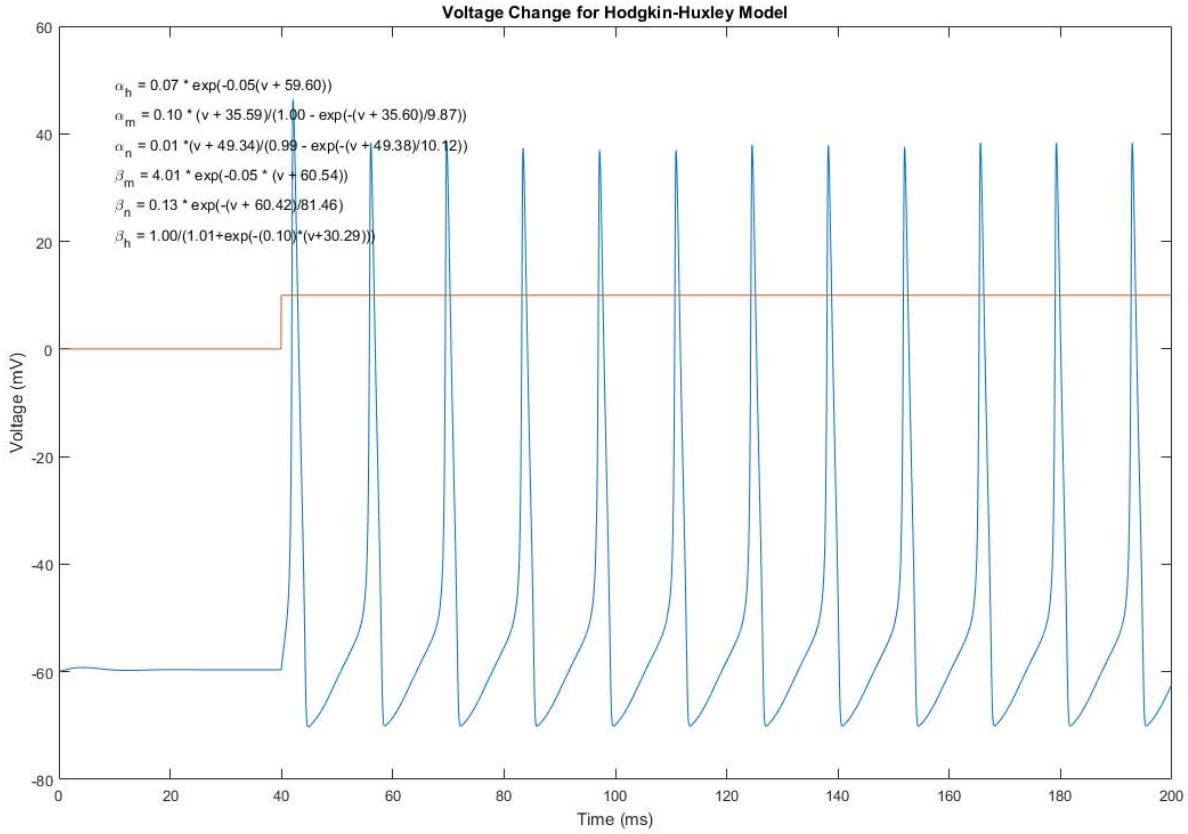


Figure 5.3. GUI of the random search simulation software developed using Matlab.



	1	2	3	4	5
a_h	0.0700	-0.0500	60	NaN	NaN
a_h_rand	0.0692	-0.0507	59.6004	NaN	NaN

	1	2	3	4	5
a_m	0.1000	35	1	35	10
a_m_rand	0.0983	35.5867	1.0026	35.5985	9.8725

	1	2	3	4	5
a_n	0.0100	50	1	50	10
a_n_rand	0.0099	49.3396	0.9871	49.3815	10.1199

	1	2	3	4	5
b_h	1	1	0.1000	30	NaN
b_h_rand	1.0033	1.0120	0.0983	30.2870	NaN

	1	2	3	4	5
b_m	4	-0.0556	60	NaN	NaN
b_m_rand	4.0061	-0.0550	60.5428	NaN	NaN

	1	2	3	4	5
b_n	0.1250	60	80	NaN	NaN
b_n_rand	0.1261	60.4183	81.4646	NaN	NaN

Figure 5.4. An intermediate result saved from the random search simulation software.

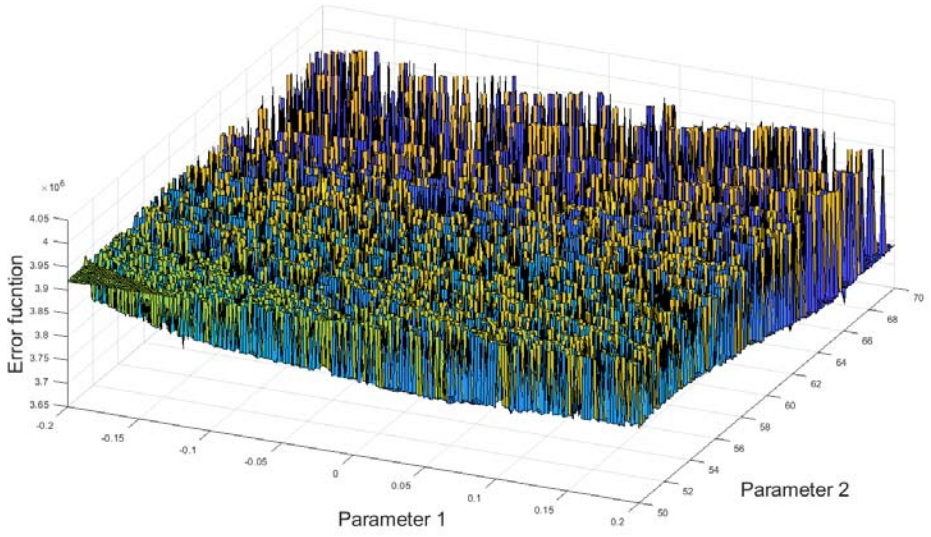


Figure 5.5. The error function with respect to two parameters of the HH model.

the target function has large number of these optimum points in the search space, the task of finding the global optimum using the gradient descent becomes inefficient. Depending on the target function some techniques such as use of multiple starting points can be used to increase the effectiveness of the algorithm. However, in the specific case, the error function contains large number of local extrema which makes the use of gradient descent ineffective. The figure 5.5 shows the shape of the error function for this specific case with respect to two parameters of the HH model.

5.3.4 Genetic Algorithm

Over the last few decades heuristic search methods have gained an extreme popularity. Having once been looked down on in the field of optimization, these algorithms have gained a huge amount of respect today. One of the top contenders in this category is the genetic algorithm (GA). The term genetic algorithm and its basic fundamentals were first proposed by Holland in his book *Adaptation in Natural and Artificial Systems* in 1975 [97]. The GA falls in to a whole class of algorithms known as evolutionary algorithms (EA) that are based on the concepts of the natural selection and biological evolution.

Like all the other EAs, GA is inspired by the process of natural selection where the survival of the fittest is guaranteed in a competing environment. The survivors of this environment would go on to produce offspring that carry the strong genetic markers to the future generations. Continuing this process of passing the strong genetic markers down generations is the key aspect of natural selection.

In a GA application, a solution to a problem is encoded by a chromosome. A population is created using these chromosomes. A selection is made from the population based on a fitness function to produce offspring who inherits the characteristics of their parents. The generated offspring become the new population and the process continues until a stopping criteria is met.

The entire process of a GA is divided in to five basic phases.

1. Initial population
2. Fitness function
3. Selection
4. Crossover
5. Mutation

The processes performed in phases 4 and 5 are called genetic operations.

5.3.4.1 Initial Population

The process begins with a set of solutions to the problem called individuals. This initial set of individuals is called the initial population. Each of these solutions or the individuals are encoded using some form of encoding method to be used in the algorithm. Once encoded, each individual is called a chromosome. There are various encoding methods being used in practice. The binary encoding method and the value encoding method are two of the techniques that are very popular.

Binary encoding is the process where all the information in the solution is encoded using a string of bits 0 and 1.

Table 5.2. Example of a binary encoding

Chromosome A	011100110110000101101100
Chromosome B	011101010010111001101100

Although binary encoding is a commonly used it does not fit as a natural method for some optimization problems.

Value encoding is the process where all the information is encoded using some numbers, letters or objects. For most problems this method fits as a natural method of encoding.

Table 5.3. Examples of value encoding

Chromosome A	2.345	7.768	11.8978	5.231
Chromosome B	AMC	TKK	NPS	DLG
Chromosome C	+	−	×	÷

Each segment of a chromosome is called a gene(a parameter or a variable of the solution) and a set of genes are joined to create a chromosome.

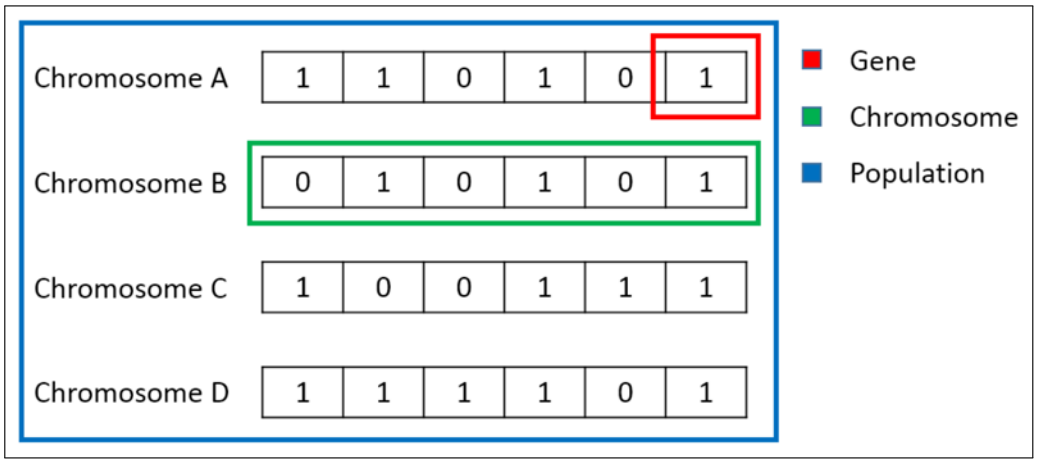


Figure 5.6. Examples of gene, chromosome and population terms.

5.3.4.2 Fitness Function

The fitness function determines how fit a chromosome is. Fitness function can be one of the most important components of a GA. The probability that a selected

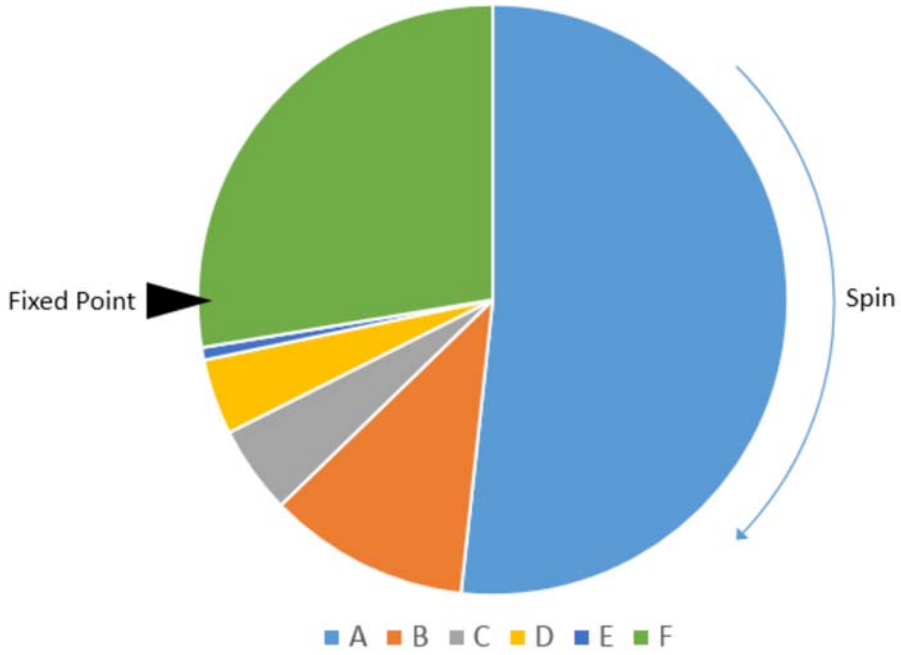


Figure 5.7. Illustration of a roulette wheel selection method.

chromosome be used to produce the next generation is based on its fitness score.

5.3.4.3 Selection

Selection is the process of selecting chromosomes based on their fitness score, to produce the next population. There exist various methods on how to select the best chromosomes. For example, roulette wheel selection, tournament selection and rank selection are some of the common methods described in the literature [97, 98].

In this work, the roulette wheel selection method[5] was employed. The roulette wheel selection can be visualized as follows. Consider a pie chart with n slices where n being the number of chromosomes in the population. Each chromosome gets a slice in the pie chart which is proportional to the fitness value it scored. Now the pie chart is rotated and the chromosome that lands on a fixed point is selected as the first parent. The same process is repeated to choose the second parent.

[5]The roulette wheel method is a member of one of the most popular selection methods known as the fitness proportionate selection

In the figure 5.7, A, B, C, D, E and F are chromosomes and the roulette wheel has selected chromosome F as a parent. It is evident from the figure that the higher the fitness score, the higher the probability of being selected as a parent. In the development process, roulette wheel selection can be performed using the following algorithm [99].

Step 1. Evaluate the fitness, f_i, of each chromosome in the population.

Step 2. Calculate the probability, p_i, of selecting each chromosome of the population:

$$p_i = \frac{f_i}{\sum_{j=1}^{n} f_i}$$

where n is the population size.

Step 3. Calculate the cumulative probability, q_i, for each individual:

$$q_i = \sum_{j=1}^{i} p_j$$

Step 4. Generate a random number, r, where $r \in (0, 1)$.

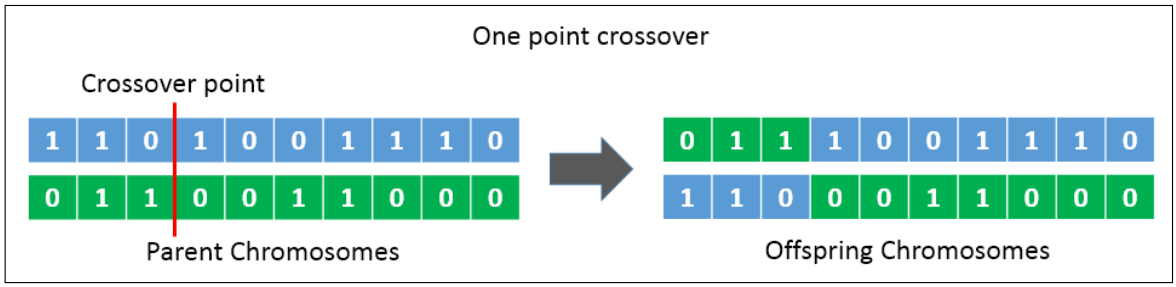
Step 5. if $r < q_1$ then select the first chromosome, x_1, else select the chromosome, x_i, where $q_i - 1 < r \leq q_i$.

Step 6. Repeat the steps 4 and 5 n times to select n candidates for the mating pool.

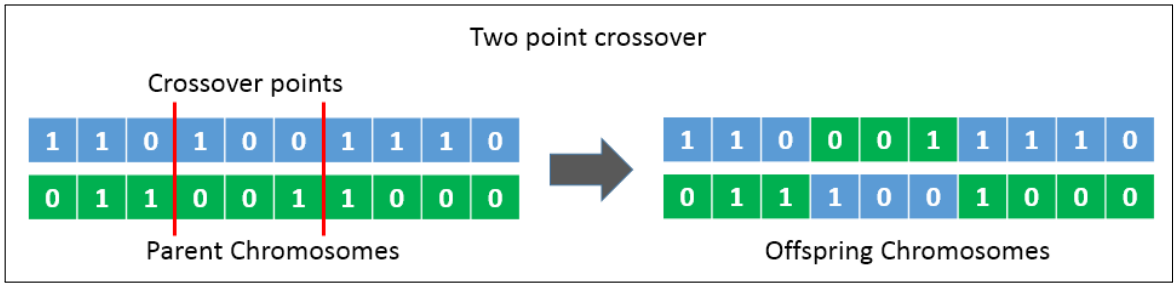
5.3.4.4 Crossover

After the mating pool has been selected, individuals from the pool are recombined or "crossed-over" to create new offspring. It is intended that the new offspring are better than their parents. In the literature, a variety of crossover methods have been described. Some of these methods are problem specific and cannot be scaled to all of the GA optimization problems. Some of the generic crossover methods are shown here.

One of the most commonly used crossover methods is the **n-point crossover**.



(a)



(b)

Figure 5.8. Illustration of an n-point crossover method. (a). One point crossover. (b). Two point crossover.

One-point and two-point crossover methods fall under this category. In one-point crossover, a crossover site is selected at random and a parent chromosome is sliced into two alleles[6]. The alleles in the same side of the crossover points of the two parents are interchanged to create two new offspring. Similarly, in two-point crossover, two crossover points are selected at random and the alleles between the two crossover points are interchanged to create new offspring as shown in figure 5.8.

5.3.4.5 Mutation

After the crossover operation, it is expected that the offspring are better than their parents. Even if the offspring were not superior to their parents, it is expected, they will be different from their parents. However, if the alleles that were interchanged between the parents in the crossover operation were identical, the offspring would not be different from their parents. The mutation is the process by which the above

[6]Allele - Any of the alternative forms of a gene

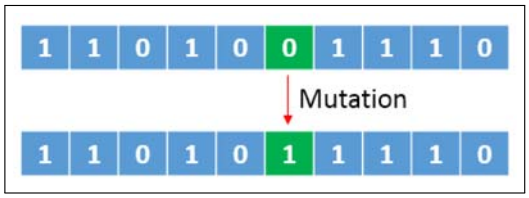


Figure 5.9. Illustration of a single bit mutation in a binary encoded chromosome.

described problem is eliminated. The mutation adds diversity to the population and ensures the possibility of exploring the entire population without a premature convergence. Typically, in a population of offspring, the mutation is applied to only a small subset of offspring.

The figure 5.9 shows an illustration of a single bit mutation in a binary encoded chromosome. Depending on the optimization problem, mutation can occur in one or more genes in the chromosome.

5.3.4.6 Stopping Criteria

After the genetic operations, the crossover and the mutation, a new population is created and is tested for its fitness for the target problem. This process is repeated until a stopping criteria is reached. There are only a few different stopping criteria being used in practice. Some of the common stopping criteria are:

1. A target fitness value is found or a solution that satisfies the problem is found.
2. Fixed number of generations have reached.
3. Percentage increase of best fitness value over a fixed number of generations is less than a predefined value.

One or more of above criteria are used to terminate a GA optimization.

The figure 5.10 shows a flowchart of a GA combining all of the above described components.

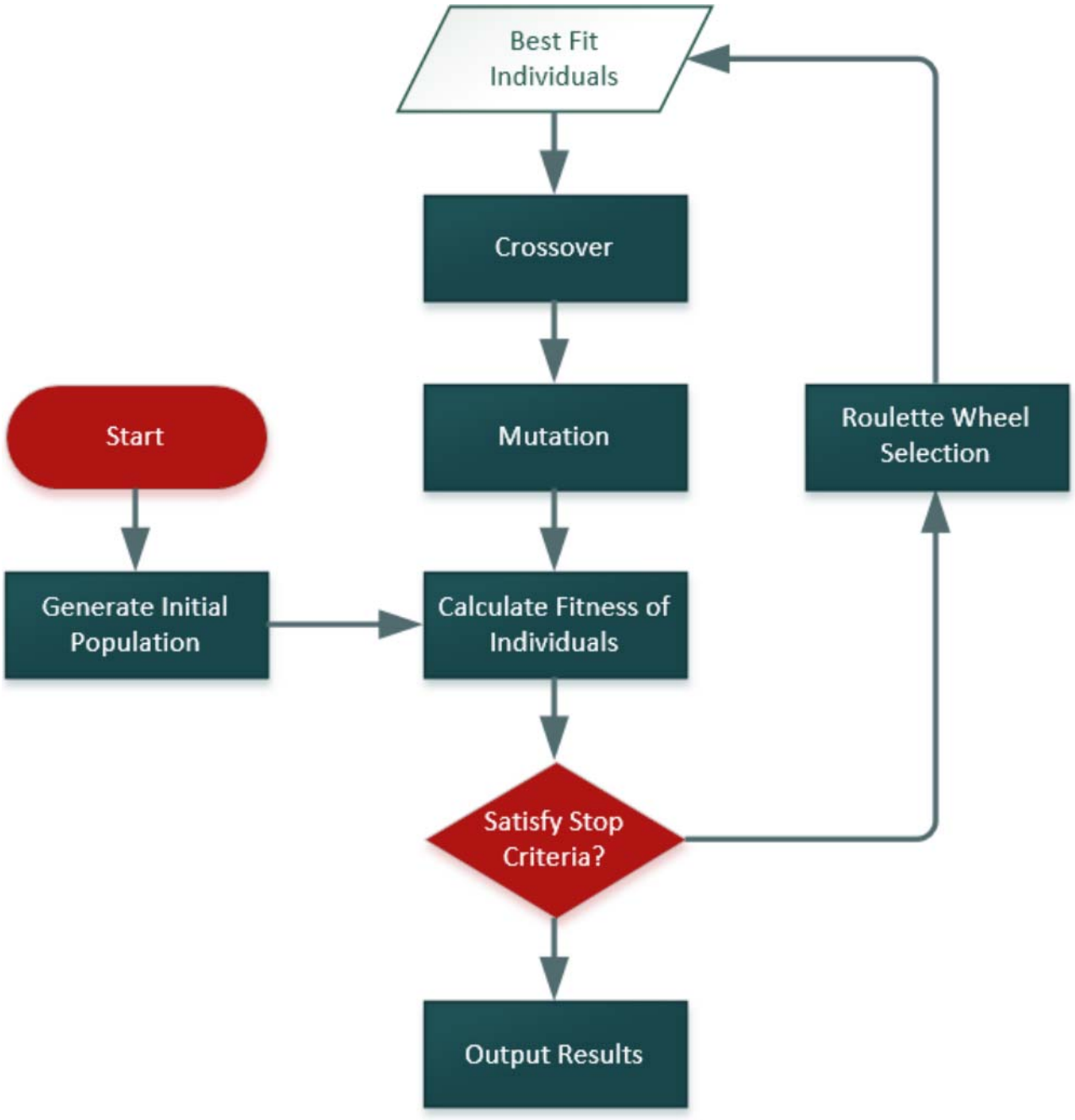


Figure 5.10. A flowchart of a genetic algorithm.

5.4 Optimizing using the Genetic Algorithm

The HH optimization began by identifying 24 parameters that would be optimized to produce two target bursting patterns as it was described in the previous sections. After exploring different optimization techniques, it was decided to use a GA due to the explained drawbacks of the other methods.

The first step of the process is to create an initial population of chromosomes that contain the information of the 24 target parameters. Since the target parameters could have positive or negative real numbers, the value encoding was used and a set of chromosomes that contained all 24 parameter values was created. A random parameter generator was used to create the initial population. The chromosomes that did not have a fitness above a critical value were rejected from the candidate pool to ensure the quality of the initial population.

Equation (5.7) was used to calculate the fitness of a given chromosome. Since (5.7) is an error equation, smaller error values provided a higher fitness score. In the actual optimization process, the GA would try to minimize the error function in (5.7).

A population of randomly generated 200 chromosomes was used to initialize the problem. The size of the initial population is a significant characteristic of GAs that determines the quality of the solution found as well as the computational resources required to find it [100, 101]. Although this can be subjective and dependent upon various factors, the initial population size is often selected to be ten times the number of dimensions in the problem [102].

The roulette wheel selection method was used throughout multiple optimization runs. Before the selection step, 5% of the best chromosomes were selected for the next generation. The rest of the population was created using the genetic operations. While an 80% of the next generation is created using crossover, the remaining 15% is created using both crossover and mutation. The above percentages used in the GA

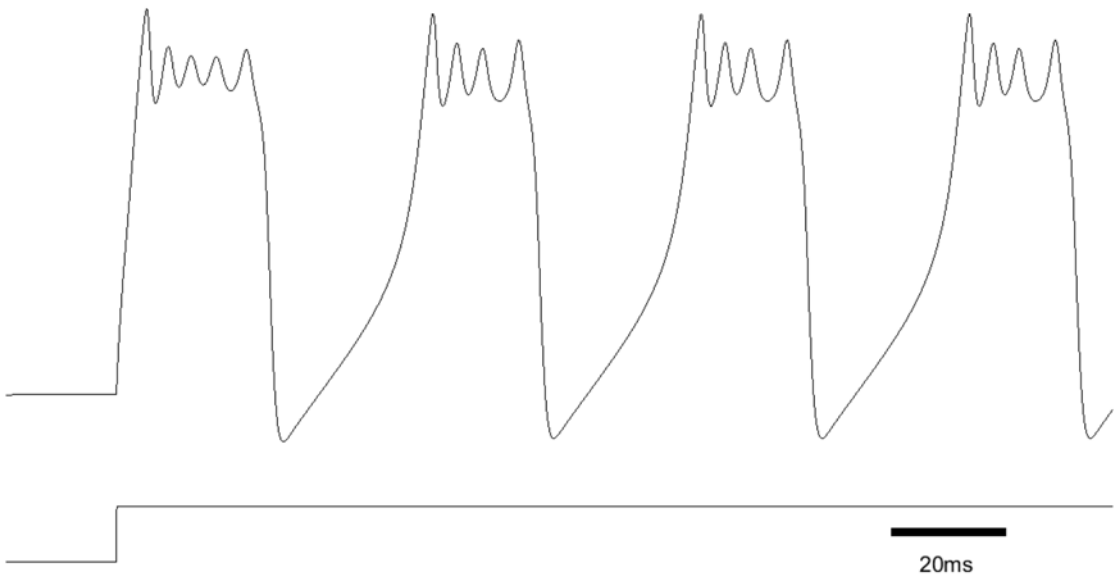
were chosen arbitrarily.

Two stopping criteria (criteria 1 and 3 listed in the section 5.3.4.6) were used to stop the optimization. Typically the optimizations are set to stop after a certain number of generations. The number of generations that a GA would go through to produce an optimum result may vary on various factors and the maximum value for this number of generations could depend on the size of the population, fitness of the initial population as well as the percentage of mutation [103, 104]. This can also be dependent upon the specific optimization problem. Due to all of the above reasons it was decided not to use the number of generations stopping criteria.

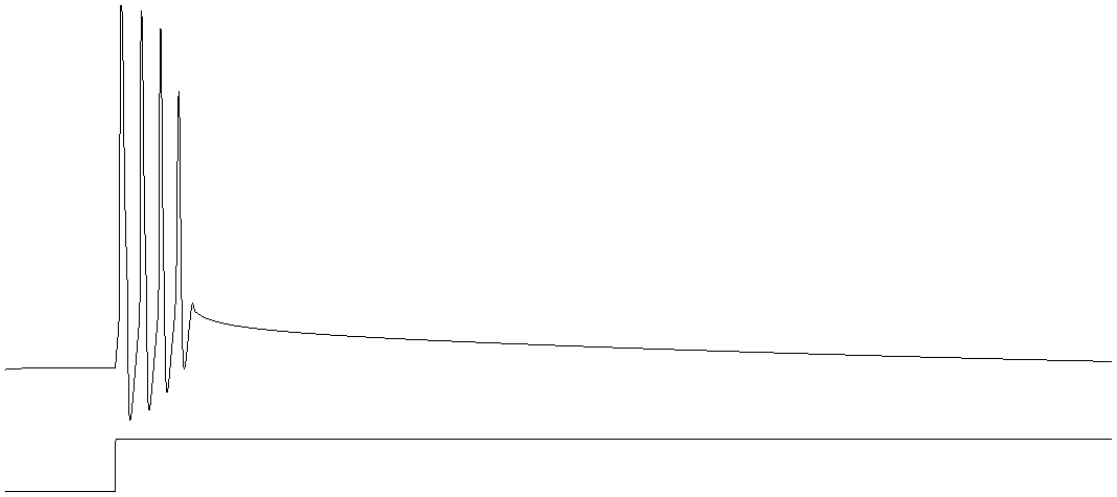
5.4.1 Results of the Genetic Algorithm optimization

A genetic algorithm based parameter search method was employed to explore the possibility of using HH model to simulate bursting behavior of the neurons. 23 constants in the HH model were converted to parameters that can be modified and the genetic algorithm was used to optimize these parameters to obtain two neuron bursting patterns. Results from the genetic algorithm shown in fig.5.11 displayed membrane potential patterns that have the characteristics of bursting behaviors similar to the references used for the optimization. All the simulations were performed with an external current that takes the shape of a step function. The optimized parameters found in the study are listed in the tonic and phasic columns in Table 5.4. One can produce fig.5.11 by substituting the tonic and phasic values from Table 5.4 in the equations (5.5) and (5.6). and solving the HH model. The absolute percentage error values of the two resulting bursting patterns with respect to the target bursting patterns are listed in Table 5.5. Equation (5.8) was used to calculate the absolute percentage error.

$$absolute\ percentage\ error = \left| \frac{V_m - V_{ref}}{V_{ref}} \right| \times 100\% \tag{5.8}$$



(a)



(b)

Figure 5.11. Two bursting patterns found from the genetic algorithm corresponding to the used references. (a). Tonic bursting. (b). Phasic bursting.

One of the important observations made while studying these results is the behavior of the gating variables in the phase space as shown in the fig. 5.12. When the HH model was originally developed in 1952, the authors had to come up with some postulates about the existence of three fictional particles that would describe

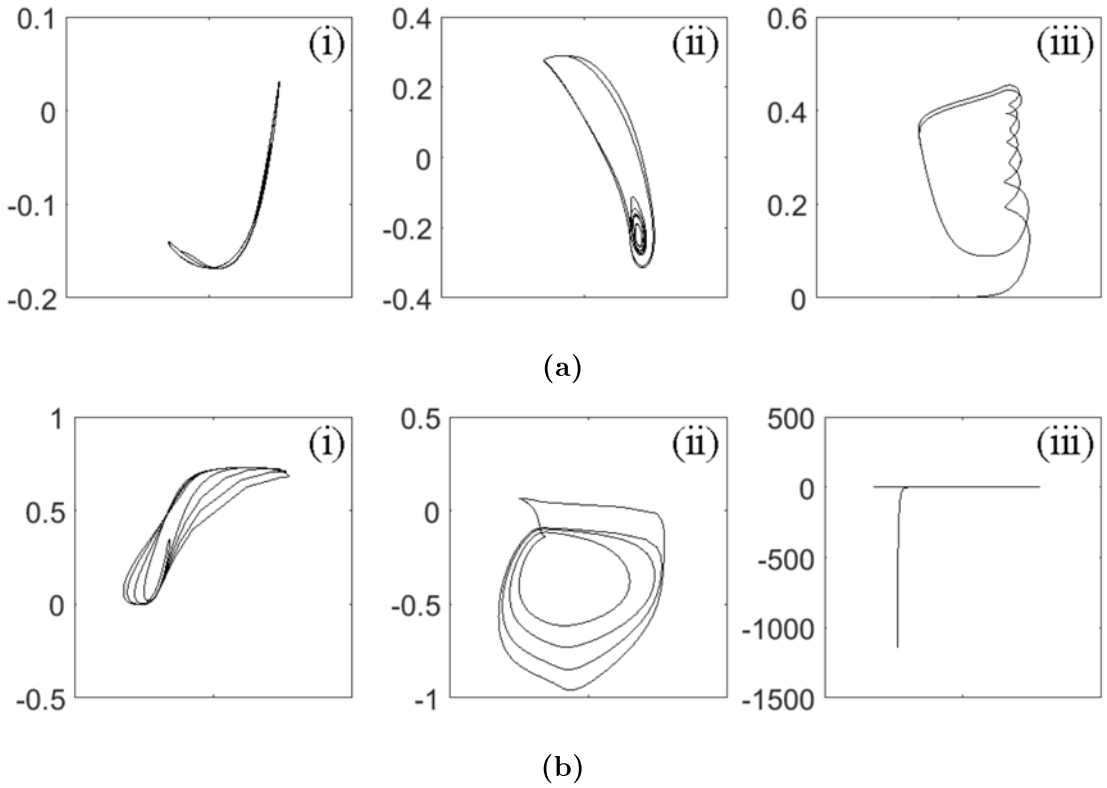


Figure 5.12. Phase space plots corresponding to the bursting patterns shown in figure 5.11. (a). Tonic bursting. (b). Phasic bursting. (i), (ii) and (iii) corresponds to the gating variables m, n and h respectively.

the states of the Sodium and Potassium channels in the axon. n,m and h are the dimensionless quantities that are associated with the activation of the Potassium channel, activation of the Sodium channel and the inactivation of the Sodium channel respectively. The values of n,m and h were such that $0 \leq n, m, h \leq 1$. In order to fit the data they obtained from the experiments, Hodgkin and Huxley chose (4.10) among many possible solutions to explain Potassium and Sodium channel activation. The behavior of the gating variables in phase space for the parameters in the original model is shown in the fig. 4.9b and fig. 4.10b. It can be noted that the values range between 0 and 1. However, when simulating the bursting behavior the values of the gating variables exceeded these limits.

The observation of having gating variables that exceed the original limits can be due to the limitations of the HH model. More recent studies have found the existence

Table 5.4. Optimized parameter values that produce fig. 5.11 compared with the parameters that produce the original HH model (5.3) and (5.4)

Parameter	Original	Tonic	Phasic
α_{m1}	0.100	−0.160	−0.113
α_{m2}	35.000	26.450	50.790
α_{m3}	1.000	0.200	−0.241
α_{m4}	35.000	33.000	36.560
α_{m5}	10.000	11.301	6.307
α_{n1}	0.010	0.086	0.013
α_{n2}	50.000	35.573	38.338
α_{n3}	1.000	−2.745	−1.200
α_{n4}	50.000	50.116	43.679
α_{n5}	10.000	12.608	14.799
α_{h1}	0.070	0.007	−0.077
α_{h2}	−0.050	0.220	−0.028
α_{h3}	60.000	38.500	64.935
β_{m1}	4.000	5.450	4.404
β_{m2}	−0.056	0.046	0.013
β_{m3}	60.000	52.420	58.466
β_{n1}	0.125	0.993	0.828
β_{n2}	60.000	55.346	50.960
β_{n3}	80.000	89.568	79.297
β_{h1}	1.000	0.140	−0.748
β_{h2}	1.000	2.044	−1.901
β_{h3}	0.100	−0.116	0.740
β_{h4}	30.000	39.537	29.871
I_{ext}	0.100	0.115	0.207

Table 5.5. Absolute percentage error values of the two resulting bursting patterns with respect to the target bursting patterns

	Tonic	Phasic
Absolute percentage error	5.88%	17.91%

of additional channels in the neuronal membranes beyond Sodium and Potassium. The presence of these additional channels has implications on the membrane potentials [105, 106]. Further, as stated before the equations associated with the gating variables lack a physical basis. Hence the known limits of the gating variables may only suit the specific pattern or patterns of membrane potentials for which the original model was intended for. However, this study explores the possibilities of the

model beyond its intended capabilities.

Although the results did not perfectly match the target patterns that were used, they demonstrated the ability to modify the HH model to simulate bursting behavior with an error less than 6% for tonic bursting and an error less than 18% for phasic bursting. While the proposed HH model seems more complex compared to a simple mathematical model like Izhikevich model that offers more flexibility, it has been demonstrated that HH model is comparable to Izhikevich model in computational cost [107]. Therefore, it is worth exploring the possibility of expanding the HH model beyond its original model parameters to simulate the bursting behavior because the equations of the HH model have a biophysical meaning unlike a pure mathematical model like the Izhikevich model.

CHAPTER 6

SUMMARY AND FUTURE WORK

Over the last few decades, neural prostheses have been used to improve human health and to restore neural functions lost due to injury. The spinal cord implants(SCI) used to perform spinal cord stimulation to relieve chronic pain is a great example for a neural prosthesis. The use of neural prostheses such as SCI and the successes they have had in the neuroscience field has led the scientific community to perform extensive research in the field. Recent studies in this field found the capability to use SCS to enable voluntary motor functions in patients with spinal cord injury [9, 10]. These recent developments have pushed researchers to explore and better understand the underlying functionality of the SCS that activates motor functions in paralyzed patients. However, it is not practical and very difficult to perform extensive experimentation on the human spinal cord. Therefore, modeling and simulation can be utilized in neuroscience to portray visual depictions of the SCS and to get a better picture of the above mentioned observations.

The work in this dissertation has embodied different modeling and simulation approaches related to SCS. The work began with simulations to visualize the electric field distributions in the spinal cord during stimulation. The researchers who conducted the groundbreaking discovery that the SCS can be used to activate motor function, believed that the controlled activation of motor neurons utilizing different electric filed patterns is the key to push their work to the next level and allow pa-

tients with spinal cord injuries the ability to walk again. Studies performed in this regard using FEM has shown the ability to easily configure electrode positioning and stimulus parameters to quickly perform simulations to help visualize the electric field distributions in the spinal cord.

The electrical stimulation on biological tissue raises significant safety concerns that cannot be disregarded. Therefore, safety measures must be taken and the importance of these safety measures must be studied. One of the most discussed topics related to safety in SCS is the charge balancing in electrical stimulation. Simulations were performed to visualize the charge balancing and to explore the electro-chemistry related to SCS. The results of the electro-chemistry solutions showed the importance of using a charge balanced, biphasic pulse as the stimulus to prevent the charge accumulation in biological tissue. Further, a novel concept known as pulse position optimization (PPO) was developed to minimize the pulse collisions in SCS and custom software was developed to perform PPO. PPO software provide the ability to optimize up to 5 independent stimuli waveforms to minimize pulse collisions.

Neuron level simulations were performed to explore the potential simulation work in the micro or cellular level to aid SCS studies. The modeling of spiking and bursting behavior of neurons were explored using Hodgkin-Huxley and the Izhikevich models. The Bi-domain models were built to combine electric filed simulations with neuron level simulations to provide information on neuron activation for given voltage pulse stimuli. This work utilized finite element modeling using software modules as well as equation based modeling techniques in COMSOL multiphysics software platform.

One of the major contrasts between the two models used in the cellular level simulations; the HH model and the Izhikevich model is that the HH model has a biological significance while the Izhikevich model is purely mathematical. A major drawback in the classic HH model is that it does not have the capability to simulate bursting behavior of neurons. However, HH model consists of enough parameters

that can be modified to replicate bursting behavior. Therefore, the last part of this work was focused on modifying the classic HH model parameters to produce bursting behaviors. Two bursting patterns were selected as target patterns to limit the scope of the work. Different optimization techniques were studied and considered as potential methods of optimization before selecting a machine learning approach to solve the problem. A genetic algorithm based method was used to find the parameters for the HH model that would produce two of the known bursting patterns described by Izhikevich [93]. The optimized HH model was able to generate bursting patterns corresponding to the two target bursting patterns used in this study with error values $< 18\%$ for Phasic bursting and $< 6\%$ for Tonic Bursting.

A comparison of the best mean absolute percent error values obtained from the methods considered to optimize HH parameters are listed in Table 6.1. Gradient descent method was not employed for any simulations, therefore, no error values were calculated.

Table 6.1. Comparison of the best mean absolute percent error values obtained from the different methods considered to optimize the HH parameters

Optimization Method	Tonic	Phasic
Genetic Algorithm	5.88%	17.91%
Random Search	$> 70.00\%$	$> 70.00\%$
Brute Force	$> 90.00\%$	$> 90.00\%$
Gradient Descent	NA	NA

The work done in this study provided a lot of insights to potential future work that can be performed to improve the understanding as well as the effectiveness of SCS. One of most important areas of investigation is the integration of multiple neurons in bi-domain models. The simulations performed in this work, focused on embedding a single neuron in spinal cord to couple electric field effects to neuron activation. This can be extended to model multiple neurons to study the action potential propagation through a chain of neurons. In order to transmit signals from one neuron to another, the physics and chemistry of synapses must be explored.

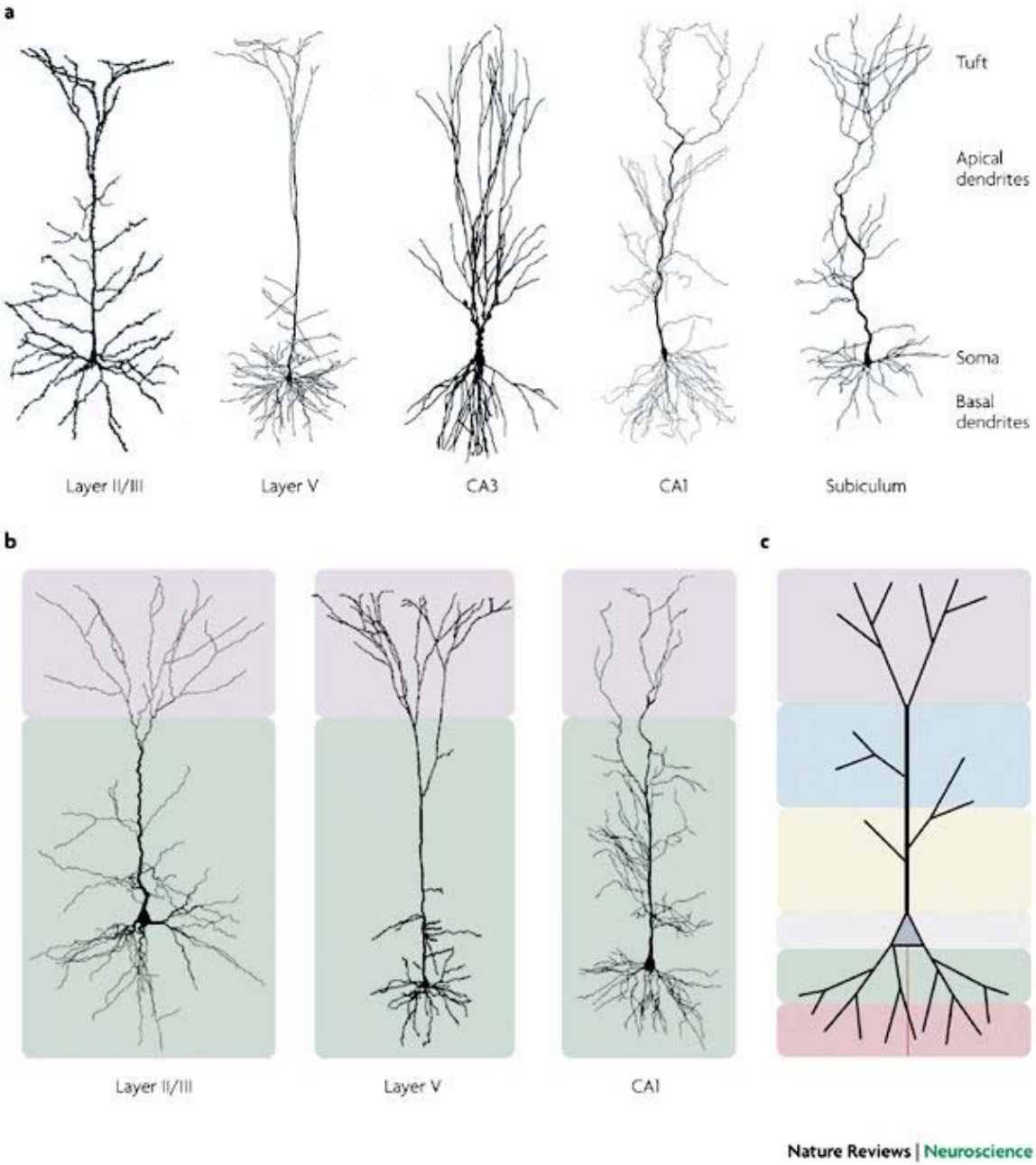


Figure 6.1. (a), (b). The structures of pyramidal neurons from different cortical areas. (c). A schematic drawing of a pyramidal neuron.[109].

Further, detailed descriptions of actual neurons (fig. 6.1) are documented and can be found in the literature [108, 109]. Information on these actual neurons can be used to build accurate models of actual neurons so that these models can be used to implement bi-domain models.

One other potential extension of the bi-domain model is the incorporation of the actual spinal cord damage data into the modeling process. Even though the electric field distribution patterns in the spinal cord can be evaluated for a given stimuli, the result may not be accurate for a selected individual with a certain form of damage to their spinal cord. Hence, to better customize the simulations for each individual and to optimize the application of the external electrical stimuli, the data related to the damage of the spinal cord must be embedded into the model. Finite element analysis studies conducted by Arle *et al.* had shown that it is possible to model effects of scar on patterns of dorsal column stimulation [48]. Similar techniques can be utilized to model the effects of spinal cord damage on the electric field patterns. Including such secondary effects on the electric field evaluations can produce the most accurate results.

It is vital to use the actual stimulation parameters in the simulation studies to make the relevant correlation between the stimuli and the desired simulation outputs. However, this study did not have the access to most of the stimulation parameter data used in human experiments. Therefore, the simulation studies were limited to proof of concept stage. The next phase of the study could use the real stimulation data and tune the simulations to assist human experiment studies.

The use of machine learning and data analytic techniques not only in scientific tasks but also in day-to-day tasks has become a new phenomenon. As shown in the last part of this dissertation, machine learning and various optimization techniques can improve SCS work significantly. The optimizations done in this work show the ability to improve existing models to give more information about the internal mechanics of SCS. Similar optimization techniques can be used to map the different stimuli input to one or more known outputs of SCS. For instance, the provided stimuli in SCS can be recorded alongside various muscle movement data such as EMG. The recorded data can then be used with machine learning algorithms such as the

genetic algorithm used in this study to find the optimum stimuli to obtain certain muscle movements. This process can even be automated to reduce hundreds of man hours of manual fine tuning of stimulus parameters to find the optimum one. If a machine learning system is developed with a feedback mechanism to determine whether or not a certain SCS input is effective, the system in theory could learn the optimum parameters on its own. This method could be the most preferable since one set of optimum parameters used for one patient may not be the optimum for a second patient.

REFERENCES

[1] G. Handa. Neural prosthesis - past, present and future. *Indian Journal of Physical Medicine and Rehabilitation*, 17(1), 2006.

[2] E. Fernandez, B. Greger, P. A. House, I. Aranda, C. Botella, J. Albisua, C. Soto-Sanchez, A. Alfaro, and R. A. Normann. Acute human brain responses to intracortical microelectrode arrays: challenges and future prospects. *Front Neuroeng*, 7:24, 2014.

[3] A. Prochazka, V. K. Mushahwar, and D. B. McCreery. Neural prostheses. *J Physiol*, 533(Pt 1):99–109, 2001.

[4] H. H. Lim, M. Lenarz, and T. Lenarz. Auditory midbrain implant: a review. *Trends Amplif*, 13(3):149–80, 2009.

[5] N. J. Kenefick, R. J. Nicholls, R. G. Cohen, and M. A. Kamm. Permanent sacral nerve stimulation for treatment of idiopathic constipation. *Br J Surg*, 89(7):882–8, 2002.

[6] M. B. Chancellor and E. J. Chartier-Kastler. Principles of sacral nerve stimulation (sns) for the treatment of bladder and urethral sphincter dysfunctions. *Neuromodulation*, 3(1):16–26, 2000.

[7] K. Kumar, G. M. Wyant, and R. Nath. Deep brain stimulation for control of intractable pain in humans, present and future: a ten-year follow-up. *Neurosurgery*, 26(5):774–81; discussion 781–2, 1990.

[8] Christopher Foundation and Dana Reeve. One degree of separation: Paralysis and spinal cord injury in the united states, 2009.

[9] S. Harkema, Y. Gerasimenko, J. Hodes, J. Burdick, C. Angeli, Y. Chen, C. Ferreira, A. Willhite, E. Rejc, R. G. Grossman, and V. R. Edgerton. Effect of epidural stimulation of the lumbosacral spinal cord on voluntary movement, standing, and assisted stepping after motor complete paraplegia: a case study. *Lancet*, 377(9781):1938–47, 2011.

[10] C. A. Angeli, V. R. Edgerton, Y. P. Gerasimenko, and S. J. Harkema. Altering spinal cord excitability enables voluntary movements after chronic complete paralysis in humans. *Brain*, 137(Pt 5):1394–409, 2014.

[11] Emily Waltz. Spinal stimulation gets paralyzed patients moving. *IEEE Spectrum*, November, 24 Oct 2013.

[12] Emily Waltz. Electrical spine stimulation helps paralyzed patients regain some movement. *IEEE Spectrum*, May, 9 Apr 2014.

[13] A. L. Benabid, P. Pollak, D. Hoffmann, C. Gervason, M. Hommel, J. E. Perret, J. de Rougemont, and D. M. Gao. Long-term suppression of tremor by chronic stimulation of the ventral intermediate thalamic nucleus. *The Lancet*, 337(8738):403–406, 1991.

[14] J. Christopher and Winfree. Spinal cord stimulation for the relief of chronic pain. *Current Surgery*, 62(5):476–481, 2005.

[15] Krishna Kumar, Cory Toth, RahulK Nath, and Patricia Laing. Epidural spinal cord stimulation for treatment of chronic pain–some predictors of success. a 15-year experience. *Surgical Neurology*, 50(2):110–121, 1998.

[16] D. R. McNeal. Analysis of a model for excitation of myelinated nerve. *IEEE Transactions on Biomedical Engineering*, BME-23(4):329–337, 1976.

[17] F. Rattay. Analysis of models for external stimulation of axons. *IEEE Transactions on Biomedical Engineering*, BME-33(10):974–977, 1986.

[18] Medtronic. Top 40 medical device companies, 2014. `https://www.mddionline.com/top-40-medical-device-companies`.

[19] Medtronic. *Medtronic SPECIFY 5-6-5, Lead Kit.* 2007.

[20] C. Gabriel, S. Gabriel, and E. Corthout. The dielectric properties of biological tissues: I. literature survey. *Phys Med Biol*, 41(11):2231–49, 1996.

[21] D. R. Merrill, M. Bikson, and J. G. Jefferys. Electrical stimulation of excitable tissue: design of efficacious and safe protocols. *J Neurosci Methods*, 141(2):171–98, 2005.

[22] N. Laotaveerungrueng. *A high-voltage, high-current multi-channel arbitrary waveform generator ASIC for neural interface and MEMS applications.* Thesis, 2011.

[23] J. L. Vargas Luna, M. Krenn, J. A. Cortes, and W. Mayr. Comparison of current and voltage control techniques for neuromuscular electrical stimulation in the anterior thigh. *Biomed Tech (Berl)*, 2013.

[24] S. F. Lempka, M. D. Johnson, S. Miocinovic, J. L. Vitek, and C. C. McIntyre. Current-controlled deep brain stimulation reduces in vivo voltage fluctuations observed during voltage-controlled stimulation. *Clin Neurophysiol*, 121(12):2128–33, 2010.

[25] J. Simpson and M. Ghovanloo. An experimental study of voltage, current, and charge controlled stimulation front-end circuitry. In *2007 IEEE International Symposium on Circuits and Systems*, pages 325–328.

[26] M. Ghovanloo. Switched-capacitor based implantable low-power wireless microstimulating systems. In *2006 IEEE International Symposium on Circuits and Systems*, page 4 pp.

[27] S. F. Lempka, S. Miocinovic, M. D. Johnson, J. L. Vitek, and C. C. McIntyre. In vivo impedance spectroscopy of deep brain stimulation electrodes. *J Neural Eng*, 6(4):046001, 2009.

[28] J. C. Williams, J. A. Hippensteel, J. Dilgen, W. Shain, and D. R. Kipke. Complex impedance spectroscopy for monitoring tissue responses to inserted neural implants. *J Neural Eng*, 4(4):410–23, 2007.

[29] S. J. Dorgan and R. B. Reilly. A model for human skin impedance during surface functional neuromuscular stimulation. *IEEE Trans Rehabil Eng*, 7(3):341–8, 1999.

[30] S. F. Cogan. Neural stimulation and recording electrodes. *Annu Rev Biomed Eng*, 10:275–309, 2008.

[31] M. Bazant, K. Chu, and B. Bayly. Current-voltage relations for electrochemical thin films. *SIAM Journal on Applied Mathematics*, 65(5):1463–1484, 2005.

[32] Hainan Wang, Alexander Thiele, and Laurent Pilon. Simulations of cyclic voltammetry for electric double layers in asymmetric electrolytes: A generalized modified poissonnernstplanck model. *The Journal of Physical Chemistry C*, 117(36):18286–18297, 2013.

[33] D. S. Bolintineanu, A. Sayyed-Ahmad, H. T. Davis, and Y. N. Kaznessis. Poisson-nernst-planck models of nonequilibrium ion electrodiffusion through a protegrin transmembrane pore. *PLoS Comput Biol*, 5(1):e1000277, 2009.

[34] J. Cartailler, Z. Schuss, and D. Holcman. Analysis of the poissonnernstplanck equation in a ball for modeling the voltagecurrent relation in neurobiological microdomains. *Physica D: Nonlinear Phenomena*, 339:39–48, 2017.

[35] A. Poisson and A. Papaud. Diffusion coefficients of major ions in seawater. *Marine Chemistry*, 13(4):265–280, 1983.

[36] A. Butterwick, A. Vankov, P. Huie, Y. Freyvert, and D. Palanker. Tissue damage by pulsed electrical stimulation. *IEEE Transactions on Biomedical Engineering*, 54(12):2261–2267, 2007.

[37] Stuart F. Cogan, Kip A. Ludwig, Cristin G. Welle, and Pavel Takmakov. Tissue damage thresholds during therapeutic electrical stimulation. *Journal of neural engineering*, 13(2):021001–021001, 2016.

[38] J. J Sit. *An Asynchronous, Low-Power Architecture for Interleaved Neural Stimulation, using Envelope and Phase Information.* Thesis, 2000.

[39] M. Ortmanns, A. Rocke, M. Gehrke, and H. J. Tiedtke. A 232-channel epiretinal stimulator asic. *IEEE Journal of Solid-State Circuits*, 42(12):2946–2959, 2007.

[40] K. Sooksood, T. Stieglitz, and M. Ortmanns. An experimental study on passive charge balancing. *Adv. Radio Sci.*, 7:197–200, 2009.

[41] K. Sooksood, T. Stieglitz, and M. Ortmanns. An active approach for charge balancing in functional electrical stimulation. *IEEE Transactions on Biomedical Circuits and Systems*, 4(3):162–170, 2010.

[42] H. Y. Ko, J. H. Park, Y. B. Shin, and S. Y. Baek. Gross quantitative measurements of spinal cord segments in human. *Spinal Cord*, 42(1):35–40, 2004.

[43] C. Gabriel, S. Gabriel, and E. Corthout. The dielectric properties of biological tissues: I. literature survey. *Phys Med Biol*, 41(11):2231–49, 1996.

[44] S. Gabriel, R. W. Lau, and C. Gabriel. The dielectric properties of biological tissues: Ii. measurements in the frequency range 10 hz to 20 ghz. *Phys Med Biol*, 41(11):2251–69, 1996.

[45] S. Gabriel, R. W. Lau, and C. Gabriel. The dielectric properties of biological tissues: Iii. parametric models for the dielectric spectrum of tissues. *Phys Med Biol*, 41(11):2271–93, 1996.

[46] V. Anderson and J. Rowley. Tissue dielectric properties calculator, 1998.

[47] X. Min, A. R. Kent, S. P. Rosenberg, and T. A. Fayram. Modeling dermatome selectivity of single-and multiple-current source spinal cord stimulation systems. *Conf Proc IEEE Eng Med Biol Soc*, 2014:6246–9, 2014.

[48] J. E. Arle, K. W. Carlson, L. Mei, and J. L. Shils. Modeling effects of scar on patterns of dorsal column stimulation. *Neuromodulation*, 17(4):320–33; discussion 333, 2014.

[49] R. Zengin, N. G. Gener, and F. Kkdeveci. Numerical analysis of spinal cord stimulation with a 2-electrode percutaneous lead. In *2014 18th National Biomedical Engineering Meeting*, pages 1–4.

[50] Nervous system. `https://en.wikipedia.org/wiki/Nervous_system`.

[51] E. Strickland. From macro to micro: A visual guide to the brain, 2017. `https://spectrum.ieee.org/biomedical/imaging/from-macro-to-micro-a-visual-guide-to-the-brain`.

[52] Stephen M. Stahl. *Stahl's Essential Psychopharmacology: Neuroscientific Basis and Practical Applications.* Cambridge University Press, 2013.

[53] J. L. Salzer and B. Zalc. Myelination. *Current Biology*, 26(20):R971–R975, 2016.

[54] A. L. Hodgkin and R. D. Keynes. Active transport of cations in giant axons from sepia and loligo. *The Journal of Physiology*, 128(1):28–60, 1955.

[55] R Appali. *Modeling the coupling of action potential and electrodes*. Thesis, 2014.

[56] David E. Goldman. Potential, impedance, and rectification in membranes. *The Journal of General Physiology*, 27(1):37–60, 1943.

[57] L. Barr. Membrane potential profiles and the goldman equation. *Journal of Theoretical Biology*, 9(3):351–356, 1965.

[58] John P. Sandblom and George Eisenman. Membrane potentials at zero current: The significance of a constant ionic permeability ratio. *Biophysical Journal*, 7(3):217–242, 1967.

[59] Stephen H. Wright. Generation of resting membrane potential. *Advances in Physiology Education*, 28(4):139–142, 2004.

[60] V. A. Shaposhnik. Walter nernst and analytical chemistry. *Journal of Analytical Chemistry*, 63(2):199–201, 2008.

[61] Gerhard Ertl. Walther nernst and the development of physical chemistry. *Angewandte Chemie International Edition*, 54(20):5828–5835, 2015.

[62] Thomas P. Trappenberg. *Fundamentals of Computational Neuroscience*. Oxford University Press Inc., 1st edition, 2002.

[63] A. V. Hill. Excitation and accommodation in nerve. *Proceedings of the Royal Society of London. Series B - Biological Sciences*, 119(814):305, 1936.

[64] A. L. Hodgkin and A. F. Huxley. Action potentials recorded from inside a nerve fibre. *Nature*, 144:710, 1939.

[65] A. L. Hodgkin and A. F. Huxley. Resting and action potentials in single nerve fibres. *The Journal of Physiology*, 104(2):176–195, 1945.

[66] A. L. Hodgkin and B. Katz. The effect of sodium ions on the electrical activity of giant axon of the squid. *J Physiol*, 108(1):37–77, 1949.

[67] A. L. Hodgkin. The local electric changes associated with repetitive action in a non-medullated axon. *J Physiol*, 107(2):165–81, 1948.

[68] A. L. Hodgkin, A. F. Huxley, and B. Katz. Measurement of current-voltage relations in the membrane of the giant axon of loligo. *The Journal of Physiology*, 116(4):424–448, 1952.

[69] A. L. Hodgkin and A. F. Huxley. Currents carried by sodium and potassium ions through the membrane of the giant axon of loligo. *J Physiol*, 116(4):449–72, 1952.

[70] A. L. Hodgkin and A. F. Huxley. The components of membrane conductance in the giant axon of loligo. *J Physiol*, 116(4):473–96, 1952.

[71] A. L. Hodgkin and A. F. Huxley. The dual effect of membrane potential on sodium conductance in the giant axon of loligo. *J Physiol*, 116(4):497–506, 1952.

[72] A. L. Hodgkin and A. F. Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *J Physiol*, 117(4):500–44, 1952.

[73] A. L. Hodgkin and W. A. H. Rushton. The electrical constants of a crustacean nerve fibre. *Proceedings of the Royal Society of London. Series B - Biological Sciences*, 133(873):444, 1946.

[74] C. Koch. *Biophsyics of computation: Information processing in single neurons.* Oxford University press, 1999.

[75] B. Carnahan, H. A. Luther, and J. O. Wikes. *Applied numerical methods.* Wiley, 1990.

[76] J. R. Clay, D. Paydarfar, and D. B. Forger. A simple modification of the hodgkin and huxley equations explains type 3 excitability in squid giant axons. *J R Soc Interface*, 5(29):1421–8, 2008.

[77] Sébastien Joucla, Alain Glière, and Blaise Yvert. Current approaches to model extracellular electrical neural microstimulation. *Frontiers in Computational Neuroscience*, 8:13, 2014.

[78] R. J. Greenberg, T. J. Velte, M. S. Humayun, G. N. Scarlatis, and E. De Juan. A computational model of electrical stimulation of the retinal ganglion cell. *IEEE Transactions on Biomedical Engineering*, 46(5):505–514, 1999.

[79] S. C. Bellinger, J. M. Rho, and P. N. Steinmetz. Modeling action potential generation during single and dual electrode stimulation of ca3 axons in hippocampal slice. *Computers in Biology and Medicine*, 40(5):487–497, 2010.

[80] M. L. Hines and N. T. Carnevale. The neuron simulation environment. *Neural Comput.*, 9(6):1179–1209, 1997.

[81] V. Schnabel and J. J. Struijk. Evaluation of the cable model for electrical stimulation of unmyelinated nerve fibers. *IEEE Transactions on Biomedical Engineering*, 48(9):1027–1033, 2001.

[82] C. Moulin, A. Glière, D. Barbier, S. Joucla, B. Yvert, P. Mailley, and R. Guillemaud. A new 3-d finite-element model based on thin-film approximation for microelectrode array recording of extracellular action potential. *IEEE Transactions on Biomedical Engineering*, 55(2):683–692, 2008.

[83] Johannes Martinek, Yvonne Stickler, Martin Reichel, Winfried Mayr, and Frank Rattay. A novel approach to simulate hodgkin-huxley-like excitation with comsol multiphysics. *Artificial Organs*, 32(8):614–619, 2008.

[84] J. Martinek, Y. Stickler, M. Reichel, and F. Rattay. Simulating hodgkin-huxley-like excitation using comsol multiphysics, 2008.

[85] Laxmi Shaw and Sangeeta Bhaga. Online emg signal analysis for diagnosis of neuromuscular diseases by using pca and pnn. *International Journal of Engineering Science and Technology*, 4:25–36, 2012.

[86] Jamileh Yousefi and Andrew Hamilton-Wright. Characterizing emg data using machine-learning tools. *Computers in Biology and Medicine*, 51:1–13, 2014.

[87] Richard FitzHugh. Impulses and physiological states in theoretical models of nerve membrane. *Biophysical Journal*, 1(6):445–466, 1961.

[88] J. L. Hindmarsh and R. M. Rose. A model of neuronal bursting using three coupled first order differential equations. *Proc R Soc Lond B Biol Sci*, 221(1222):87–102, 1984.

[89] H. R. Wilson and J. D. Cowan. A mathematical theory of the functional dynamics of cortical and thalamic nervous tissue. *Kybernetik*, 13(2):55–80, 1973.

[90] J. P. Weick, Y. Liu, and S. C. Zhang. Human embryonic stem cell-derived neurons adopt and regulate the activity of an established neural network. *Proc Natl Acad Sci U S A*, 108(50):20189–94, 2011.

[91] C. M. Gray and D. A. McCormick. Chattering cells: superficial pyramidal neurons contributing to the generation of synchronous oscillations in the visual cortex. *Science*, 274(5284):109–13, 1996.

[92] E. M. Izhikevich, N. S. Desai, E. C. Walcott, and F. C. Hoppensteadt. Bursts as a unit of neural information: selective communication via resonance. *Trends Neurosci*, 26(3):161–7, 2003.

[93] E. M. Izhikevich. Simple model of spiking neurons. *IEEE Trans Neural Netw*, 14(6):1569–72, 2003.

[94] Barry W. Connors and Michael J. Gutnick. Intrinsic firing patterns of diverse neocortical neurons. *Trends in Neurosciences*, 13(3):99–104, 1990.

[95] Jay R. Gibson, Michael Beierlein, and Barry W. Connors. Two networks of electrically coupled inhibitory neurons in neocortex. *Nature*, 402:75, 1999.

[96] E. M. Izhikevich. Which model to use for cortical spiking neurons? *IEEE Trans Neural Netw*, 15(5):1063–70, 2004.

[97] John H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, MI, 1975.

[98] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Longman Publishing Co., Inc., 1989.

[99] Sastry K., Goldberg D., and Kendall G. *Genetic Algorithms.* Springer, Boston, MA, 2005.

[100] Pedro A. Diaz-Gomez and Dean F. Hougen. Initial population for genetic algorithms: A metric approach, 2007.

[101] Tianshi Chen, Ke Tang, Guoliang Chen, and Xin Yao. A large population size can be unhelpful in evolutionary algorithms. *Theoretical Computer Science*, 436(Supplement C):54–70, 2012.

[102] R. Storn. On the usage of differential evolution for function optimization. In *Proceedings of North American Fuzzy Information Processing*, pages 519–523.

[103] M. W. Gutowski. Amazing geometry of genetic space or are genetic algorithms convergent? *CoRR*, abs/cs/0512019, 2005.

[104] S. Shannon. *Leading Edge Computer Science Research.* Nova Science, 2006.

[105] P. R. Adams, A. Constanti, D. A. Brown, and R. B. Clark. Intracellular ca2+ activates a fast voltage-sensitive k+ current in vertebrate sympathetic neurones. *Nature*, 296(5859):746–9, 1982.

[106] P. Sah and E. S. Faber. Channels underlying neuronal calcium-activated potassium currents. *Prog Neurobiol*, 66(5):345–53, 2002.

[107] M. J. Skocik and L. N. Long. On the capabilities and computational costs of neuron models. *IEEE Trans Neural Netw Learn Syst*, 25(8):1474–83, 2014.

[108] Galina Schevzov, Nicole S. Bryce, Rowena Almonte-Baldonado, Josephine Joya, Jim J. C. Lin, Edna Hardeman, Ron Weinberger, and Peter Gunning. Specific features of neuronal size and shape are regulated by tropomyosin isoforms. *Molecular Biology of the Cell*, 16(7):3425–3437, 2005.

[109] N. Spruston. Pyramidal neurons: dendritic structure and synaptic integration. *Nature Reviews Neuroscience*, 9:206, 2008.

Appendix A: Acronyms and Symbols

SCI	Spinal Cord Implant
DBI	Deep Brain Implant
SCS	Spinal Cord Stimulation
FES	Functional Electrical Stimulation
FEM	Finite Element Method
CCS	Constant Current Stimulation
VCS	Voltage Controlled Stimulation
PPO	Pulse Position Optimization
CSF	Cerebrospinal Fluid
GUI	Graphical User Interface
CNS	Central Nervous System
PNS	Peripheral Nervous System
ANS	Autonomic Nervous System
EMG	Electromyography
GA	Genetic Algorithm
EA	Evolutionary Algorithms

ϵ	Permittivity
σ	Conductivity (S/m)
C_{dl}	Double layer capacitance (F)
R_f	Faradaic resistance (Ω)
R_s	Tissue resistance (Ω)
Δ_ϕ	Equlibrium potential (V)
N_i	Flux $(mol \cdot m^{-2}s^{-1})$
D_i	Diffusion Coefficient (m^2s^{-1})
$u_{m,i}$	mobility $(s \cdot mol \cdot Kg^{-1})$
z_i	Charge number
F	Faraday's constant $(C \cdot mol^{-1})$
c_i	Concentration $(mol \cdot m^{-3})$
ρ	Space charge density $(C \cdot m^{-3})$
V_m	Membrane potential (mV)
C_m	Membrane capacitance $(\mu F/cm^2)$
$g_{Na},\ g_K,\ g_{Cl}$	Ionic conductivity of Sodium, Potassium and Chloride ions
$E_{Na},\ E_K$	Equilibrium potential of Sodium and Potassium ions
E_L	Equilibrium potential at zero leakage current

Appendix B: Pulse Position Optimization Program

The following shows the PPO algorithm and the supporting programs used to achieve and simulate PPO.

1. **minimize_collisions.m**

```
function str = minimize_collisions(freq, src_pw, src_amp, ...
    int_delay,...
     recharge_factor, shorting_factor, time, loop_increment, val)
tic;
%% Enter Waveform Data
sink_pw = src_pw*recharge_factor; % sink - pulse width (micro ...
    seconds)
sink_amp = src_amp/recharge_factor; % sink amplitude
shorting = src_pw*shorting_factor; % shorting (micro seconds)
iter = val;

%% Generate Waveforms
p_cell = cell(1,iter);
state_cell = cell(1,iter);

for i=1:iter
    [p_cell{i}, state_cell{i}, ¬] = wave_gen( src_pw(i), ...
        src_amp(i),...
         int_delay(i), sink_pw(i), sink_amp(i), shorting(i), ...
            freq(i), time);
end

%% Modify waveform states to separate resting and non-resting ...
    portions
for i=1:iter
    state_cell{i} = state_cell{i}>0;
end

%% offset waveforms and waveform states
% collision data stores the offset values and number of collisions.
% collision_data(n) = number of collisions
% collision_data(1:n-1) = offset values of n-1 waveforms

collision_data = zeros(1,5);
collision_data(1)=0;
if  (val == 3 || val == 4 || val ==5)
```

```
32      collision_data(2)=0;
33  elseif (val == 4 || val ==5)
34      collision_data(3)=0;
35  elseif (val ==5)
36      collision_data(4)=0;
37  end
38  collision_data(val)=1e5;
39
40  fall = strfind(state_cell{val},[1 0]);
41  rise = strfind(state_cell{val},[0 1]);
42  shift_gap = abs(fall(1)-rise(1))+abs(fall(2)-rise(1));
43  permutations = ceil(shift_gap/loop_increment)^(val-1);
44
45  index =0;
46  canceled = 0;
47  wb = waitbar(0,'Finding the optimum position...',...
48      'CreateCancelBtn','setappdata(gcbf,''Cancel'',1)');
49  set(wb,'Name','Waveform Generator: Progress')
50
51  if val == 2
52      for offset_i=0:loop_increment:shift_gap
53          if canceled == 1
54              break;
55          end
56              index=index+1;
57              % offset n-1 waveforms so they won't start at once
58              x_offset = offset_i;
59
60              state_cell_copy = state_cell;
61
62              z_cell = cell(1,iter-1);
63          for i=1:1
64              z_cell{i} = p_cell{i}*0;
65              z_cell{i}(x_offset+1:end)=state_cell_copy{i}...
66              (1:end-x_offset);
67              state_cell_copy{i}=z_cell{i};
68          end
69
70
71              %% Calculate Collisions and Grace periods
72              no_collision = 0;
73              no_of_collisions = ...
                    count_collisions(state_cell_copy,val);
74
75              if(collision_data(2)>no_of_collisions)
76                  collision_data(1)=offset_i;
77                  collision_data(2)=no_of_collisions;
78              end
79
80              %update waitbar
81              if (mod(i,1)==0) % to slow the update rate
82                  waitbar(index/permutations);
83              end
84              if getappdata(wb,'Cancel')
```

```
                canceled = 1;
                break;
            end

    end
end

if val == 3
    for offset_i=0:loop_increment:shift_gap
        if canceled == 1
            break;
        end
        for offset_j=0:loop_increment:shift_gap

            index=index+1;
            % offset n-1 waveforms so they won't start at once
            x_offset = [offset_i,offset_j];

            state_cell_copy = state_cell;

            z_cell = cell(1,iter-1);
            for i=1:2
                z_cell{i} = p_cell{i}*0;
                z_cell{i}(x_offset(i)+1:end)=...
                    state_cell_copy{i}(1:end-x_offset(i));
                state_cell_copy{i}=z_cell{i};
            end

            %% Calculate Collisions and Grace periods
            no_of_collisions = ...
               count_collisions(state_cell_copy,val);

            if(collision_data(3)>no_of_collisions)
                collision_data(1)=offset_i;
                collision_data(2)=offset_j;
                collision_data(3)=no_of_collisions;
            end

            %update waitbar
            if (mod(i,1)==0) % to slow the update rate
                waitbar(index/permutations);
            end
            if getappdata(wb,'Cancel')
                canceled = 1;
                break;
            end
        end
    end
end

if val == 4
    for offset_i=0:loop_increment:shift_gap
        if canceled == 1
            break;
```

```
138         end
139         for offset_j=0:loop_increment:shift_gap
140             if canceled == 1
141                 break;
142             end
143             for offset_k=0:loop_increment:shift_gap
144                 index=index+1;
145                 % offset n-1 waveforms so they won't start at once
146                 x_offset = [offset_i,offset_j,offset_k];
147 
148                 state_cell_copy = state_cell;
149 
150                 z_cell = cell(1,iter-1);
151                 for i=1:3
152                     z_cell{i} = p_cell{i}*0;
153                     z_cell{i}(x_offset(i)+1:end)=...
154                         state_cell_copy{i}(1:end-x_offset(i));
155                     state_cell_copy{i}=z_cell{i};
156                 end
157 
158                 %% Calculate Collisions and Grace periods
159                 no_of_collisions = ...
                        count_collisions(state_cell_copy,val);
160 
161                 if(collision_data(4)>no_of_collisions)
162                     collision_data(1)=offset_i;
163                     collision_data(2)=offset_j;
164                     collision_data(3)=offset_k;
165                     collision_data(4)=no_of_collisions;
166                 end
167 
168                 %update waitbar
169                 if (mod(i,1)==0) % to slow the update rate
170                     waitbar(index/permutations);
171                 end
172                 if getappdata(wb,'Cancel')
173                     canceled = 1;
174                     break;
175                 end
176             end
177         end
178     end
179 end
180 
181 if val == 5
182     for offset_i=0:loop_increment:shift_gap
183         if canceled == 1
184             break;
185         end
186         for offset_j=0:loop_increment:shift_gap
187             if canceled == 1
188                 break;
189             end
190             for offset_k=0:loop_increment:shift_gap
```

```
191                 if canceled == 1
192                     break;
193                 end
194                 for offset_l=0:loop_increment:shift_gap
195                     index=index+1;
196                     % offset n-1 waveforms so they won't start ...
                           at once
197                     x_offset = ...
                           [offset_i,offset_j,offset_k,offset_l];
198
199                     state_cell_copy = state_cell;
200
201                     z_cell = cell(1,iter-1);
202                     for i=1:4
203                         z_cell{i} = p_cell{i}*0;
204                         z_cell{i}(x_offset(i)+1:end)=...
205                             state_cell_copy{i}(1:end-x_offset(i));
206                         state_cell_copy{i}=z_cell{i};
207                     end
208
209                     %% Calculate Collisions and Grace periods
210                     no_of_collisions=count_collisions...
211                     (state_cell_copy,val);
212                     if(collision_data(5)>no_of_collisions)
213                         collision_data(1)=offset_i;
214                         collision_data(2)=offset_j;
215                         collision_data(3)=offset_k;
216                         collision_data(4)=offset_l;
217                         collision_data(5)=no_of_collisions;
218                     end
219
220                     %update waitbar
221                     if (mod(i,1)==0) % to slow the update rate
222                         waitbar(index/permutations);
223                     end
224                     if getappdata(wb,'Cancel')
225                         canceled = 1;
226                         break;
227                     end
228                 end
229             end
230         end
231     end
232 end
233
234 delete(wb);
235
236 cal_time = toc;
237 if val == 2
238     if canceled == 1
239         str = sprintf(['Optimization program aborted\n'...
240             'Current Best shifting value: %d\n'...
241             '(Elapsed time is %.3f ...
                   seconds)'],collision_data(1),cal_time);
```

```
242     else
243         str = sprintf(['Best shifting value for the entered ...
                data:'...
244             '%d\n (Elapsed time is %.3f seconds)'],...
245             collision_data(1),cal_time);
246     end
247
248 elseif val == 3
249     if canceled == 1
250         str = sprintf(['Optimization program aborted\n'...
251             'Current Best shifting values:'...
252             ' %d, %d\n (Elapsed time is %.3f seconds)'],...
253             collision_data(1),collision_data(2),cal_time);
254     else
255         str = sprintf(['Best shifting values for the entered ...
                data:'...
256             ' %d, %d\n (Elapsed time is %.3f seconds)'],...
257             collision_data(1),collision_data(2),cal_time);
258     end
259
260 elseif val == 4
261     if canceled == 1
262         str = sprintf(['Optimization program aborted\n'...
263             'Current Best shifting values: %d, %d, %d\n'...
264             ' (Elapsed time is %.3f seconds)'],collision_data(1),...
265             collision_data(2),collision_data(3),cal_time);
266     else
267         str = sprintf(['Best shifting values for the entered ...
                data:'...
268             ' %d, %d, %d\n (Elapsed time is %.3f seconds)'],...
269             collision_data(1),collision_data(2),collision_data(3),...
270             cal_time);
271     end
272
273 elseif val == 5
274     if canceled == 1
275         str = sprintf(['Optimization program aborted\n'...
276             'Current Best shifting values: %d, %d, %d, %d\n'...
277             ' (Elapsed time is %.3f seconds)'],...
278             collision_data(1),collision_data(2),...
279             collision_data(3),collision_data(4),cal_time);
280     else
281         str = sprintf(['Best shifting values for the entered ...
                data:'...
282             ' %d, %d, %d, %d\n (Elapsed time is %.3f seconds)'],...
283             collision_data(1),collision_data(2),collision_data(3),...
284             collision_data(4),cal_time);
285     end
286 end
287
288 end
```

2. waveforms.m

```
function waveforms(freq, src_pw, src_amp, int_delay, y_offset,...
    recharge_factor, shorting_factor, time, x_offset, ...
        disp_status, val)
clc;
%% Enter Waveform Data
sink_pw = src_pw*recharge_factor; % sink - pulse width (micro ...
    seconds)
sink_amp = src_amp/recharge_factor; % sink amplitude
shorting = src_pw*shorting_factor; % shorting (micro seconds)
iter = val;

%% Generate Waveforms
p_cell = cell(1,iter);
state_cell = cell(1,iter);

for i=1:iter
    [p_cell{i}, state_cell{i}, t] = wave_gen( src_pw(i), ...
        src_amp(i),...
         int_delay(i), sink_pw(i), sink_amp(i), shorting(i), ...
            freq(i), time);
end

%% offset waveforms and waveform states
z_cell = cell(1,iter-1);
for i=1:iter-1
    z_cell{i} = p_cell{i}*0;
    z_cell{i}(x_offset(i)+1:end)=p_cell{i}(1:end-x_offset(i));
    p_cell{i}=z_cell{i};

    z_cell{i} = p_cell{i}*0-111;
    z_cell{i}(x_offset(i)+1:end)=state_cell{i}(1:end-x_offset(i));
    state_cell{i}=z_cell{i};
end

%% Modify waveform states to separate resting and non-resting ...
    portions
for i=1:iter
    state_cell{i} = state_cell{i}>0;
end

%% Calculate Collisions and no-wave periods
if val == 2
    collision_combinations = 1;%2C2
elseif val==3
    collision_combinations = 4;%3C2+3C3
elseif val ==4
    collision_combinations = 11;%4C2+4C3+4C4
elseif val ==5
    collision_combinations = 26;%5C2+5C3+5C4+5C5
end

```

```
rest_cell = cell(1,collision_combinations);
collision_cell = cell(1,collision_combinations);
min_freq = zeros(1,collision_combinations);
str_array=cell(1,collision_combinations);
%%%%%%%%%% Collisions between waveform-A and waveform-B

index = 0;
if (val ==2 || val ==3 || val==4 || val ==5)
    for i=1:iter-1
        for j=i+1:iter
            index = index + 1;
            rest_cell{index}=state_cell{i}+state_cell{j};
            rest_cell{index}=rest_cell{index}==0;

            collision_cell{index}=state_cell{i}+state_cell{j};
            collision_cell{index}=collision_cell{index}==2;

            f_list = [freq(i),freq(j)];
            min_freq(index) = min(f_list);
            str_array{index}=sprintf('%dHz, %dHz',freq(i),freq(j));
        end
    end
end

if val==3
    index = index+1;
    f_list = [freq(i),freq(j)];
    min_freq(index) = min(f_list);
    str_array{index}=sprintf('All Three');
    collision_cell{index}=(state_cell{1}+state_cell{2}...
    +state_cell{3})==3;
    rest_cell{index}=(state_cell{1}+state_cell{2}+state_cell{3})==0;
end

%%%%%%%%%% Collisions between three waveforms (A-B-C, A-B-D, etc)
if (val==4 || val ==5)
for i=1:iter-2
    for j=i+1:iter-1
        for k = j+1:iter
            index = index + 1;
            rest_cell{index}=state_cell{i}+state_cell{j}+...
            state_cell{k};
            rest_cell{index}=rest_cell{index}==0;

            collision_cell{index}=...
                state_cell{i}+state_cell{j}+state_cell{k};
            collision_cell{index}=collision_cell{index}==3;

            f_list = [freq(i),freq(j),freq(k)];
            min_freq(index) = min(f_list);
            str_array{index}=...
                sprintf('%dHz, %dHz, %dHz',freq(i),freq(j),freq(k));
        end
    end
```

```
end
end

if val==4
    index = index+1;
    f_list = [freq(i),freq(j),freq(k)];
    min_freq(index) = min(f_list);
    str_array{index}=sprintf('All Four');
    collision_cell{index}=...
        (state_cell{1}+state_cell{2}+state_cell{3}+state_cell{4})==4;
    rest_cell{index}=...
        (state_cell{1}+state_cell{2}+state_cell{3}+state_cell{4})==0;
end

%%%%%%%%%% Collisions between four waveforms (A-B-C-D, A-B-D-E, ...
    etc)
if val==5
for i=1:iter-3
    for j=i+1:iter-2
        for k = j+1:iter-1
            for l = k+1:iter
                index = index + 1;
                rest_cell{index}=state_cell{i}+...
                    state_cell{j}+state_cell{k}+state_cell{l};
                rest_cell{index}=rest_cell{index}==0;

                collision_cell{index}=state_cell{i}+...
                    state_cell{j}+state_cell{k}+state_cell{l};
                collision_cell{index}=collision_cell{index}==4;

                f_list = [freq(i),freq(j),freq(k),freq(l)];
                min_freq(index) = min(f_list);
                str_array{index}=sprintf('%dHz, %dHz, %dHz, ...
                    %dHz',...
                    freq(i),freq(j),freq(k),freq(l));
            end
        end
    end
end
%%%%%%%%%% Collisions between all five waveforms
index=index+1;
f_list = [freq(i),freq(j),freq(k),freq(l)];
min_freq(index) = min(f_list);
str_array{index}=sprintf('All Five');
collision_cell{index}=(state_cell{1}+state_cell{2}+...
    state_cell{3}+state_cell{4}+state_cell{5})==5;
rest_cell{index}=(state_cell{1}+state_cell{2}+...
    state_cell{3}+state_cell{4}+state_cell{5})==0;
clear index;
end

%% Plot Waveforms
if disp_status(1)==1
    figure
```

```
153     hold on
154     for i=1:iter
155         plot(t, p_cell{i}+y_offset(i));
156         str = sprintf('f = %d Hz, PW = %d us',freq(i), src_pw(i));
157         text(1e4, y_offset(i)+src_amp(i)+3, str, 'Color', 'k');
158     end
159     hold off
160 
161     xlabel('time (\mus)');
162     ylabel('Amplitude');
163     title(sprintf('Re-charging - x%.1f, Shorting - x%.1f',...
164         recharge_factor, shorting_factor));
165 end
166 
167 %% Plot Collisions
168 if disp_status(3)==1
169     figure
170     hold on
171     for i = 1:collision_combinations
172         plot(t,collision_cell{i}+i*2,'r');
173         str = sprintf('%s colliding',str_array{i});
174         text(1e4, 2*i+1.5, str, 'Color', 'k');
175     end
176     hold off
177 
178     switch val
179         case 2
180             ylim([1.5 3.5]);
181         case 3
182             ylim([0 11]);
183         case 4
184             ylim([0 25]);
185         case 5
186             ylim([0 56]);
187     end
188 end
189 
190 %% Create a data table
191 if disp_status(2)==1
192     percentage_waiting = zeros(1,collision_combinations);
193     no_of_collisions = zeros(1,collision_combinations);
194     percentage_collisions = zeros(1,collision_combinations);
195     for i=1:collision_combinations
196         percentage_waiting(i)=sum(rest_cell{i})/...
197         length(rest_cell{i})*100;
198             if length(strfind(collision_cell{i},[1 0])) ≠...
199                     length(strfind(collision_cell{i},[0 1]))
200                 no_of_collisions(i) = length...
201                     (strfind(collision_cell{i},[0 1]))+1;
202             else
203                 no_of_collisions(i) = length...
204                     (strfind(collision_cell{i},[0 1]));
205             end
206         percentage_collisions(i) = no_of_collisions(i)/...
```

```
207             (min_freq(i)*time*1e-6)*100;
208     end
209
210     percentage_waiting = percentage_waiting';
211     no_of_collisions = no_of_collisions';
212     percentage_collisions = percentage_collisions';
213     total_collisions = sum(no_of_collisions);
214
215     data=[percentage_waiting,no_of_collisions,percentage_collisions];
216     cnames={'%No Pulse','# Collisions','% Collisions'};
217
218     switch val
219         case 2
220             xPos = 440;
221             yPos = 440;
222             tableH = 40;
223             tableW = 337;
224
225         case 3
226             xPos = 440;
227             yPos = 440;
228             tableH = 90;
229             tableW = 337;
230
231         case 4
232             xPos = 440;
233             yPos = 300;
234             tableH = 220;
235             tableW = 405;
236
237         case 5
238             xPos = 440;
239             yPos = 150;
240             tableH = 490;
241             tableW = 470;
242     end
243
244     f=figure('Position',[xPos yPos tableW tableH]);
245     t = uitable(f,'Position',[0.1 0.1 1.9 0.9],'Data',...
246         data,'ColumnName',cnames,'RowName',str_array);
247     tableextent = get(t,'Extent');
248     oldposition = get(t,'Position');
249     newposition = [oldposition(1) oldposition(2)...
250         tableextent(3) tableextent(4)];
251     set(t, 'Position', newposition);
252
253     fprintf('Total Collsions = %d\n',total_collisions);
254     fprintf(['NOTE: Percentage collision is a worst case number,'...
255         ' where\npercentage_collision ='...
256         ' no_of_collisions/no_of_pulses_from_loweset_'...
257         'frequency*100\n']);
258
259 end
260 end
```

3. wave_gen.m

```
function [ wave, wave_state, time ] = wave_gen( src_pw, src_amp,...
    int_delay, sink_pw, sink_amp, short, freq, T)

% use all the input variables in micro seconds
% make smallest time unit to 5us
% all the other parameters will be multiples of 5

dt = 10;
t_pulse = 1e6/freq;
t = 0:dt:t_pulse;
pulse = zeros(1,length(t));

% pulse_state keeps the record of the state of the pulse
% 1000 - charge
% 1100 - inter pulse delay
% 1200 - recharge
% 1300 - shorting
% -111 - wait
pulse_state = zeros(1,length(t));

t1 = src_pw;
t2 = t1 + int_delay;
t3 = t2 + sink_pw;
t4 = t3 + short;

    for i=1:length(t)
       if(i*dt<t1)
           pulse(i) = src_amp;
           pulse_state(i) = 1000;
       elseif(i*dt<t2)
           pulse(i) = 0;
           pulse_state(i) = 1100;
       elseif(i*dt<t3)
           pulse(i) = -sink_amp;
           pulse_state(i) = 1200;
       elseif(i*dt<t4)
           pulse(i) = 0;
           pulse_state(i) = 1300;
       else
           pulse(i) = 0;
           pulse_state(i) = -111;
       end
    end

time = 0:dt:T;

wave = zeros(1,length(time));
wave_state = zeros(1,length(time));

    for i=1:length(time)

```

```
52          index = mod (i,length(pulse));
53          if index == 0
54              index = length(pulse);
55          end
56          wave(i) = pulse(index);
57          wave_state(i) = pulse_state(index);
58      end
59  end
```

Appendix C: Matlab Programs Used for Optimization

1. **random_search.m**
The program that performed random search optimization.

```
function HH_simulate(alpha_mins, alpha_maxs, beta_mins, ...
    beta_maxs,...
                 n_plots, data_option_val, save_checkBox)

%Constants set for all Methods
Cm=0.01; % Membrane Capcitance uF/cm^2
dt=0.04; % Time Step ms
t=0:dt:200; %Time Array ms

% Define stimulus
I=zeros(1,length(t));
I(round(length(I)*0.1):end)=1e-1;

ENa=55.17; % mv Na reversal potential
EK=-72.14; % mv K reversal potential
El=-49.42; % mv Leakage reversal potential
gbarNa=1.2; % mS/cm^2 Na conductance
gbarK=0.36; % mS/cm^2 K conductance
gbarl=0.003; % mS/cm^2 Leakage conductance
% V(1)=-60; % Initial Membrane voltage

%% Known working Parameters
    working_vals_ah = [0.07,-0.05,60];
    working_vals_am = [0.1,35,1,35,10];
    working_vals_an = [0.01,50,1,50,10];

    working_vals_bm = [4,-0.0556,60];
    working_vals_bn = [0.125,60,80];
    working_vals_bh = [1,1,0.1,30];

%% File save options
if save_checkBox
    button = questdlg('Save the output plots in default folder?');
    if strcmp(button, 'Yes')
        disp('Plots will be saved in the default directory');
        disp('../Figures/');
        ff = pwd;
    %     if ¬exist('Figures','dir')
```

```
38          if ¬exist(sprintf('%s\\%s',ff,'Figures'),'dir')
39              mkdir('Figures');
40          end
41          folder = 'Figures';
42      elseif strcmp(button , 'No')
43          folder = uigetdir;
44          disp('Plots will be saved in the specified directory');
45          disp(folder);
46      else
47      %     uiwait(msgbox('No plots will be saved','','error'));
48          continue_choice = questdlg('No plots will be saved. Do ...
                you wish to continue?',...
49                                     'Warning',...
50                                     'Yes','No','No');
51          if strcmp(continue_choice,'No')
52              return
53          end
54      end;
55  else
56      button = 'Cancel';
57  end
58
59  if data_option_val == 0
60      %% Alphas - Min and Max values for variable
61      alpha_h_a_min = alpha_mins(1);
62      alpha_h_a_max = alpha_maxs(1);
63
64      alpha_h_b_min = alpha_mins(2);
65      alpha_h_b_max = alpha_maxs(2);
66
67      alpha_h_c_min = alpha_mins(3);
68      alpha_h_c_max = alpha_maxs(3);
69
70      alpha_m_a_min = alpha_mins(4);
71      alpha_m_a_max = alpha_maxs(4);
72
73      alpha_m_b_min = alpha_mins(5);
74      alpha_m_b_max = alpha_maxs(5);
75
76      alpha_m_c_min = alpha_mins(6);
77      alpha_m_c_max = alpha_maxs(6);
78
79      alpha_m_d_min = alpha_mins(7);
80      alpha_m_d_max = alpha_maxs(7);
81
82      alpha_m_e_min = alpha_mins(8);
83      alpha_m_e_max = alpha_maxs(8);
84
85      alpha_n_a_min = alpha_mins(9);
86      alpha_n_a_max = alpha_maxs(9);
87
88      alpha_n_b_min = alpha_mins(10);
89      alpha_n_b_max = alpha_maxs(10);
90
```

```
91      alpha_n_c_min = alpha_mins(11);
92      alpha_n_c_max = alpha_maxs(11);
93
94      alpha_n_d_min = alpha_mins(12);
95      alpha_n_d_max = alpha_maxs(12);
96
97      alpha_n_e_min = alpha_mins(13);
98      alpha_n_e_max = alpha_maxs(13);
99
100     %% Betas - Min and Max values for variable
101     beta_h_a_min = beta_mins(1);
102     beta_h_a_max = beta_maxs(1);
103
104     beta_h_b_min = beta_mins(2);
105     beta_h_b_max = beta_maxs(2);
106
107     beta_h_c_min = beta_mins(3);
108     beta_h_c_max = beta_maxs(3);
109
110     beta_h_d_min = beta_mins(4);
111     beta_h_d_max = beta_maxs(4);
112
113     beta_m_a_min = beta_mins(5);
114     beta_m_a_max = beta_maxs(5);
115
116     beta_m_b_min = beta_mins(6);
117     beta_m_b_max = beta_maxs(6);
118
119     beta_m_c_min = beta_mins(7);
120     beta_m_c_max = beta_maxs(7);
121
122     beta_n_a_min = beta_mins(8);
123     beta_n_a_max = beta_maxs(8);
124
125     beta_n_b_min = beta_mins(9);
126     beta_n_b_max = beta_maxs(9);
127
128     beta_n_c_min = beta_mins(10);
129     beta_n_c_max = beta_maxs(10);
130
131 elseif data_option_val == 1
132     %% Alphas - Min and Max values for variable
133     alpha_h_a_min = alpha_mins(1);
134     alpha_h_a_max = alpha_mins(1);
135
136     alpha_h_b_min = alpha_mins(2);
137     alpha_h_b_max = alpha_mins(2);
138
139     alpha_h_c_min = alpha_mins(3);
140     alpha_h_c_max = alpha_mins(3);
141
142     alpha_m_a_min = alpha_mins(4);
143     alpha_m_a_max = alpha_mins(4);
144
```

```
145     alpha_m_b_min = alpha_mins(5);
146     alpha_m_b_max = alpha_mins(5);
147
148     alpha_m_c_min = alpha_mins(6);
149     alpha_m_c_max = alpha_mins(6);
150
151     alpha_m_d_min = alpha_mins(7);
152     alpha_m_d_max = alpha_mins(7);
153
154     alpha_m_e_min = alpha_mins(8);
155     alpha_m_e_max = alpha_mins(8);
156
157     alpha_n_a_min = alpha_mins(9);
158     alpha_n_a_max = alpha_mins(9);
159
160     alpha_n_b_min = alpha_mins(10);
161     alpha_n_b_max = alpha_mins(10);
162
163     alpha_n_c_min = alpha_mins(11);
164     alpha_n_c_max = alpha_mins(11);
165
166     alpha_n_d_min = alpha_mins(12);
167     alpha_n_d_max = alpha_mins(12);
168
169     alpha_n_e_min = alpha_mins(13);
170     alpha_n_e_max = alpha_mins(13);
171
172     %% Betas - Min and Max values for variable
173     beta_h_a_min = beta_mins(1);
174     beta_h_a_max = beta_mins(1);
175
176     beta_h_b_min = beta_mins(2);
177     beta_h_b_max = beta_mins(2);
178
179     beta_h_c_min = beta_mins(3);
180     beta_h_c_max = beta_mins(3);
181
182     beta_h_d_min = beta_mins(4);
183     beta_h_d_max = beta_mins(4);
184
185     beta_m_a_min = beta_mins(5);
186     beta_m_a_max = beta_mins(5);
187
188     beta_m_b_min = beta_mins(6);
189     beta_m_b_max = beta_mins(6);
190
191     beta_m_c_min = beta_mins(7);
192     beta_m_c_max = beta_mins(7);
193
194     beta_n_a_min = beta_mins(8);
195     beta_n_a_max = beta_mins(8);
196
197     beta_n_b_min = beta_mins(9);
198     beta_n_b_max = beta_mins(9);
```

```
199
200     beta_n_c_min = beta_mins(10);
201     beta_n_c_max = beta_mins(10);
202 end
203
204     %% setting up alpha_h variables
205     alpha_h_a = (alpha_h_a_max - alpha_h_a_min).*rand(n_plots,1) ...
            + alpha_h_a_min;
206     alpha_h_b = (alpha_h_b_max - alpha_h_b_min).*rand(n_plots,1) ...
            + alpha_h_b_min;
207     alpha_h_c = (alpha_h_c_max - alpha_h_c_min).*rand(n_plots,1) ...
            + alpha_h_c_min;
208     a_h_vals = [alpha_h_a,alpha_h_b,alpha_h_c];
209
210     %% setting up alpha_m variables (TO EDIT)
211     alpha_m_a = (alpha_m_a_max - alpha_m_a_min).*rand(n_plots,1) ...
            + alpha_m_a_min;
212     alpha_m_b = (alpha_m_b_max - alpha_m_b_min).*rand(n_plots,1) ...
            + alpha_m_b_min;
213     alpha_m_c = (alpha_m_c_max - alpha_m_c_min).*rand(n_plots,1) ...
            + alpha_m_c_min;
214     alpha_m_d = (alpha_m_d_max - alpha_m_d_min).*rand(n_plots,1) ...
            + alpha_m_d_min;
215     alpha_m_e = (alpha_m_e_max - alpha_m_e_min).*rand(n_plots,1) ...
            + alpha_m_e_min;
216     a_m_vals = [alpha_m_a,alpha_m_b,alpha_m_c,alpha_m_d,alpha_m_e];
217
218     %% setting up alpha_n variables (TO EDIT)
219     alpha_n_a = (alpha_n_a_max - alpha_n_a_min).*rand(n_plots,1) ...
            + alpha_n_a_min;
220     alpha_n_b = (alpha_n_b_max - alpha_n_b_min).*rand(n_plots,1) ...
            + alpha_n_b_min;
221     alpha_n_c = (alpha_n_c_max - alpha_n_c_min).*rand(n_plots,1) ...
            + alpha_n_c_min;
222     alpha_n_d = (alpha_n_d_max - alpha_n_d_min).*rand(n_plots,1) ...
            + alpha_n_d_min;
223     alpha_n_e = (alpha_n_e_max - alpha_n_e_min).*rand(n_plots,1) ...
            + alpha_n_e_min;
224     a_n_vals = [alpha_n_a,alpha_n_b,alpha_n_c,alpha_n_d,alpha_n_e];
225
226     %% setting up beta_m variables
227     beta_m_a = (beta_m_a_max - beta_m_a_min).*rand(n_plots,1) + ...
            beta_m_a_min;
228     beta_m_b = (beta_m_b_max - beta_m_b_min).*rand(n_plots,1) + ...
            beta_m_b_min;
229     beta_m_c = (beta_m_c_max - beta_m_c_min).*rand(n_plots,1) + ...
            beta_m_c_min;
230     b_m_vals = [beta_m_a,beta_m_b,beta_m_c];
231
232     %% setting up beta_n variables
233     beta_n_a = (beta_n_a_max - beta_n_a_min).*rand(n_plots,1) + ...
            beta_n_a_min;
234     beta_n_b = (beta_n_b_max - beta_n_b_min).*rand(n_plots,1) + ...
            beta_n_b_min;
```

```
235     beta_n_c = (beta_n_c_max - beta_n_c_min).*rand(n_plots,1) + ...
            beta_n_c_min;
236     b_n_vals = [beta_n_a,beta_n_b,beta_n_c];
237
238     %% setting up beta_h variables
239     beta_h_a = (beta_h_a_max - beta_h_a_min).*rand(n_plots,1) + ...
            beta_h_a_min;
240     beta_h_b = (beta_h_b_max - beta_h_b_min).*rand(n_plots,1) + ...
            beta_h_b_min;
241     beta_h_c = (beta_h_c_max - beta_h_c_min).*rand(n_plots,1) + ...
            beta_h_c_min;
242     beta_h_d = (beta_h_d_max - beta_h_d_min).*rand(n_plots,1) + ...
            beta_h_d_min;
243     b_h_vals = [beta_h_a,beta_h_b,beta_h_c,beta_h_d];
244
245 %%
246 % create a waitbar if number of plots is larger than 5
247 if n_plots > 5
248     wb = waitbar(0,'Simulating and ...
            plotting...','CreateCancelBtn',...
249     'setappdata(gcbf,''Cancel'',1)');
250     set(wb,'Name','HH simulator: Progress','WindowStyle', 'modal');
251     % setappdata(wb,'Cancel',0)
252     frames = java.awt.Frame.getFrames();
253     frames(end).setAlwaysOnTop(1);
254 end
255 %
256 fig_index = 0;
257 for ki = 1:n_plots % 1st value of the "ah" equation
258
259     % update waitbar
260     if exist('wb','var')
261         waitbar(ki/n_plots);
262         if getappdata(wb,'Cancel')
263             break;
264         end
265     end
266
267     V(1)=-60;
268
269     temp_vals_ah = [a_h_vals(ki,1),a_h_vals(ki,2),a_h_vals(ki,3)];
270     temp_vals_am = [a_m_vals(ki,1),a_m_vals(ki,2),a_m_vals(ki,3),...
271     a_m_vals(ki,4),a_m_vals(ki,5)];
272     temp_vals_an = [a_n_vals(ki,1),a_n_vals(ki,2),a_n_vals(ki,3),...
273     a_n_vals(ki,4),a_n_vals(ki,5)];
274
275     temp_vals_bm = [b_m_vals(ki,1),b_m_vals(ki,2),b_m_vals(ki,3)];
276     temp_vals_bn = [b_n_vals(ki,1),b_n_vals(ki,2),b_n_vals(ki,3)];
277     temp_vals_bh = ...
            [b_h_vals(ki,1),b_h_vals(ki,2),b_h_vals(ki,3),b_h_vals(ki,4)];
278
279     m(1)=am(temp_vals_am,V(1))/(am(temp_vals_am,V(1))+...
280     bm(temp_vals_bm,V(1))); % Initial m-value
281     n(1)=an(temp_vals_an,V(1))/(an(temp_vals_an,V(1))+...
```

```
282     bn(temp_vals_bn,V(1))); % Initial n-value
283     h(1)=ah(temp_vals_ah,V(1))/(ah(temp_vals_ah,V(1))+...
284     bh(temp_vals_bh,V(1))); % Initial h-value
285
286 %% Runge-Kutta Method
287 V(1)=-61; % Initial Membrane voltage
288 m(1)=am(temp_vals_am,V(1))/(am(temp_vals_am,V(1))+...
289 bm(temp_vals_bm,V(1))); % Initial m-value
290 n(1)=an(temp_vals_an,V(1))/(an(temp_vals_an,V(1))+...
291 bn(temp_vals_bn,V(1))); % Initial n-value
292 h(1)=ah(temp_vals_ah,V(1))/(ah(temp_vals_ah,V(1))+...
293 bh(temp_vals_bh,V(1))); % Initial h-value
294
295 len = length(t);
296 current = 1e-1;
297
298 for i=1:len-1 % Loop through each step until time is finished
299 %     disp(i);
300     %4 step method of Runge-Kutta
301     K1=dt*HH(i,i,len,current,[V(i); n(i); m(i); h(i)],...
302     temp_vals_ah,temp_vals_am,temp_vals_an,temp_vals_bh,...
303     temp_vals_bm,temp_vals_bn);
304     % obtain 4 k variables (V,m,n,h) from HH function
305     k1=K1(1,1);n1=K1(2,1);m1=K1(3,1);h1=K1(4,1);
306     K2=dt*HH(i,i+(0.5*dt),len,current,[V(i)+(0.5*k1);n(i)+...
307     (0.5*n1);m(i)+(0.5*m1);h(i)+(0.5*h1)],temp_vals_ah,...
308     temp_vals_am,temp_vals_an,temp_vals_bh,temp_vals_bm,...
309     temp_vals_bn);
310     k2=K2(1,1);n2=K2(2,1);m2=K2(3,1);h2=K2(4,1);
311     K3=dt*HH(i,i+(0.5*dt),len,current,[V(i)+(0.5*k2);n(i)+...
312     (0.5*n2);m(i)+(0.5*m2);h(i)+(0.5*h2)],temp_vals_ah,...
313     temp_vals_am,temp_vals_an,temp_vals_bh,temp_vals_bm,...
314     temp_vals_bn);
315     k3=K3(1,1);n3=K3(2,1);m3=K3(3,1);h3=K3(4,1);
316     K4=dt*HH(i,i+dt,len,current,[V(i)+k3;n(i)+n3;m(i)+m3;h(i)+h3],...
317     temp_vals_ah,temp_vals_am,temp_vals_an,temp_vals_bh,...
318     temp_vals_bm,temp_vals_bn);
319     k4=K4(1,1);n4=K4(2,1);m4=K4(3,1);h4=K4(4,1);
320     %create next step for each variable
321     V(i+1)=V(i)+1/6*(k1+2*k2+2*k3+k4);
322     n(i+1)=n(i)+1/6*(n1+2*n2+2*n3+n4);
323     m(i+1)=m(i)+1/6*(m1+2*m2+2*m3+m4);
324     h(i+1)=h(i)+1/6*(h1+2*h2+2*h3+h4);
325 end
326 %set variables for graphing later
327 RK=V;
328 RKm=m;
329 RKn=n;
330 RKh=h;
331 clear V m n h;
332
333     %% Plots
334     %Plot the functions
335     if fig_index == 0
```

```
336         figure;
337     end;
338     fig_index = fig_index + 1;
339     subplot(1,15,1:11)
340     plot(t,RK);
341     hold on;
342     plot(t,I*100);
343     hold off;
344
345     xlabel('Time (ms)');
346     ylabel('Voltage (mV)');
347     title('Voltage Change for Hodgkin-Huxley Model');
348
349     y_lim = get(gca,'ylim');
350     y_text = y_lim(1)+((y_lim(2)-y_lim(1))*0.92);
351     text(10,y_text,sprintf('\\alpha_h = %.2f * exp(%.2f(v + ...
            %.2f))',...
352     temp_vals_ah(1),temp_vals_ah(2),temp_vals_ah(3)));
353
354     y_text = y_lim(1)+((y_lim(2)-y_lim(1))*0.88);
355     text(10,y_text,sprintf('\\alpha_m = %.2f * (v + %.2f)/(%.2f ...
            - exp(-(v + %.2f)/%.2f))',...
356     temp_vals_am(1),temp_vals_am(2),temp_vals_am(3),...
357     temp_vals_am(4),temp_vals_am(5)));
358
359     y_text = y_lim(1)+((y_lim(2)-y_lim(1))*0.84);
360     text(10,y_text,sprintf('\\alpha_n = %.2f *(v + %.2f)/(%.2f - ...
            exp(-(v + %.2f)/%.2f))',...
361     temp_vals_an(1),temp_vals_an(2),temp_vals_an(3),...
362     temp_vals_an(4),temp_vals_an(5)));
363
364     y_text = y_lim(1)+((y_lim(2)-y_lim(1))*0.80);
365     text(10,y_text,sprintf('\\beta_m = %.2f * exp(%.2f * (v + ...
            %.2f))',...
366     temp_vals_bm(1),temp_vals_bm(2),temp_vals_bm(3)));
367
368     y_text = y_lim(1)+((y_lim(2)-y_lim(1))*0.76);
369     text(10,y_text,sprintf('\\beta_n = %.2f * exp(-(v + ...
            %.2f)/%.2f)',...
370     temp_vals_bn(1),temp_vals_bn(2),temp_vals_bn(3)));
371
372     y_text = y_lim(1)+((y_lim(2)-y_lim(1))*0.72);
373     text(10,y_text,sprintf('\\beta_h = ...
            %.2f/(%.2f+exp(-(%.2f)*(v+%.2f)))',...
374     temp_vals_bh(1),temp_vals_bh(2),temp_vals_bh(3),...
375     temp_vals_bh(4)));
376
377     set(gcf,'Units','normalized','Position',[0.05 0.1 0.90 0.75]);
378
379     %% Plot the Data table
380     fig = gcf;
381     set(fig,'PaperUnits','inches')
382
383     data_vals = padcat(working_vals_ah, temp_vals_ah, ...
```

```
        working_vals_am,...
384     temp_vals_am, working_vals_an, temp_vals_an, working_vals_bh,...
385     temp_vals_bh, working_vals_bm, temp_vals_bm, ...
        working_vals_bn, temp_vals_bn);
386     create_table(fig,data_vals)
387
388     %% Save Plot
389     if ¬strcmp(button, 'Cancel')
390         print(fig,sprintf('%s\\%s_Fig_%d',folder,...
391         datestr(now,'dd-mmm-yyyy_HH-MM-SSAM'),fig_index),...
392         '-dpng','-r0')
393     end
394 end
395 if exist('wb','var')
396     delete(wb);
397 end
398     return;
399
400
401 %% Functions
402 function a=ah(vals,v) %Alpha value for variable h
403 a=vals(1)*exp(vals(2)*(v+vals(3)));
404
405 function a=am(vals,v) %Alpha for Variable m
406 a=vals(1)*(v+vals(2))/(vals(3)-exp(-(v+vals(4))/vals(5)));
407
408 function a=an(vals,v)%Alpha for variable n
409 a=vals(1)*(v+vals(2))/(vals(3)-exp(-(v+vals(4))/vals(5)));
410
411 function b =bh(vals,v) %beta value for variable h
412 b=vals(1)/(vals(2)+exp(-(vals(3))*(v+vals(4))));
413
414 function b=bm(vals,v) %Beta for variable m
415 b=vals(1)*exp(vals(2)*(v+vals(3)));
416
417 function b=bn(vals,v) %Beta for variable n
418 b=vals(1)*exp(-(v+vals(2))/vals(3));
419
420 function dydt = HH(index,i,len,current,y,temp_vals_ah,...
421 temp_vals_am,temp_vals_an,temp_vals_bh,temp_vals_bm,temp_vals_bn)
422 % Constants
423 ENa=55.17; % mv Na reversal potential
424 EK=-72.14; % mv K reversal potential
425 El=-49.42; % mv Leakage reversal potential
426 gbarNa=1.2; % mS/cm^2 Na conductance
427 gbarK=0.36; % mS/cm^2 K conductance
428 gbarl=0.003; % mS/cm^2 Leakage conductance
429
430 % I = 0.1; %Applied Current
431 % disp(i);
432
433 I=zeros(1,len);
434 I(round(length(I)*0.1):end)=current;
435
```

```
Cm = 0.01; %Membrane Capacitance
% Values set to equal input values
V = y(1);
n = y(2);
m = y(3);
h = y(4);
gNa=gbarNa*m^3*h;
gK=gbarK*n^4;
gl=gbarl;
INa=gNa*(V-ENa);
IK=gK*(V-EK);
Il=gl*(V-El);

%Hodgkin-Huxley Model Equation
dydt = [((1/Cm)*(I(index)-(INa+IK+Il))); ...
an(temp_vals_an,V)*(1-n)-bn(temp_vals_bn,V)*n; ...
am(temp_vals_am,V)*(1-m)-bm(temp_vals_bm,V)*m; ...
ah(temp_vals_ah,V)*(1-h)-bh(temp_vals_bh,V)*h];

function create_table(f, data)

t_ah = uitable('Parent', f, 'Position', [1200 530 465 58]);
t_am = uitable('Parent', f, 'Position', [1100 460 465 58]);
t_an = uitable('Parent', f, 'Position', [1230 400 465 58]);
t_bh = uitable('Parent', f, 'Position', [1230 335 465 58]);
t_bm = uitable('Parent', f, 'Position', [1230 270 465 58]);
t_bn = uitable('Parent', f, 'Position', [1230 205 465 58]);

set(t_ah, 'Data', data(1:2,:));
set(t_ah, 'RowName', {'a_h','a_h_rand'});

set(t_am, 'Data', data(3:4,:));
set(t_am, 'RowName', {'a_m','a_m_rand'});

set(t_an, 'Data', data(5:6,:));
set(t_an, 'RowName', {'a_n','a_n_rand'});

set(t_bh, 'Data', data(7:8,:));
set(t_bh, 'RowName', {'b_h','b_h_rand'});

set(t_bm, 'Data', data(9:10,:));
set(t_bm, 'RowName', {'b_m','b_m_rand'});

set(t_bn, 'Data', data(11:12,:));
set(t_bn, 'RowName', {'b_n','b_n_rand'});

foregroundColor = [1 1 1];
backgroundColor = [0 .6 0;0.6 0 0];
set(t_ah, 'ForegroundColor',foregroundColor);
set(t_am, 'ForegroundColor',foregroundColor);
set(t_an, 'ForegroundColor',foregroundColor);
set(t_bh, 'ForegroundColor',foregroundColor);
set(t_bm, 'ForegroundColor',foregroundColor);
set(t_bn, 'ForegroundColor',foregroundColor);
```

```
490
491 set(t_ah, 'BackgroundColor',backgroundColor);
492 set(t_am, 'BackgroundColor',backgroundColor);
493 set(t_an, 'BackgroundColor',backgroundColor);
494 set(t_bh, 'BackgroundColor',backgroundColor);
495 set(t_bm, 'BackgroundColor',backgroundColor);
496 set(t_bn, 'BackgroundColor',backgroundColor);
```

2. `init_population.m`
The program used to initialize the population with randomly generated chromosomes.

```
1  clear init_pop fitness_vals;
2  clc;
3  %%
4  set_save_dir;
5
6  global old_min;
7  old_min = 1e10;
8  loop_count = 0;
9  error_tolerence = 1e5;
10
11 while old_min > error_tolerence
12
13     loop_count = loop_count + 1;
14     fprintf('Loop %d\n',loop_count);
15
16     max_limits = [0.1,  0.3, 44,  -0.1, 40,  4, 50, 15,  0.2, 40,...
17     -2, 60, 15, 6,  0.2, 55, 2, 70, 95, 2, 3,  2, 45 0.3 2 2 2];
18     min_limits = [-0.1, 0.1, 38, -0.2, 20, -4, 20,  5, -0.2, 30,...
19     -3, 40,  8, 4, -0.2, 50, 0, 50, 85, -2, 1.5, -2, 35 0.0 -2 ...
           -2 -2];
20
21     num_var = 27;
22     pop_size = 200;
23     init_pop = zeros(pop_size,num_var);
24
25     for i=1:num_var
26         r_min = min_limits(i);
27         r_max = max_limits(i);
28
29         r = (r_max-r_min).*rand(pop_size,1) + r_min;
30
31         init_pop(:,i) = r;
32     end
33
34     %%
35
36     pop_size = length(init_pop);
37     fitness_vals = zeros(pop_size,1);
38
```

```
    for i=1:pop_size
        fitness_vals(i) = GA_HH_fitness_NEW(init_pop(i,:));
    end

    init_range_min = min(init_pop);
    init_range_max = max(init_pop);

    num_var = 27;
end
```

3. `HH_solve.m`

The program used to solve the HH model.

```
function [RK, t] = GA_HH_simulate(alphas, betas, current)
% Time Step ms / Change this to 0.1 for IZ and 0.04 for HH
dt=0.04;
t=0:dt:200; %Time Array ms
init_potential = -65;

    %% setting up alpha_h variables
    alpha_h_a = alphas(1);
    alpha_h_b = alphas(2);
    alpha_h_c = alphas(3);
    a_h_vals = [alpha_h_a,alpha_h_b,alpha_h_c];

    %% setting up alpha_m variables (TO EDIT)
    alpha_m_a = alphas(4);
    alpha_m_b = alphas(5);
    alpha_m_c = alphas(6);
    alpha_m_d = alphas(7);
    alpha_m_e = alphas(8);
    alpha_m_f = alphas(9);
    a_m_vals = [alpha_m_a,alpha_m_b,alpha_m_c,alpha_m_d,...
    alpha_m_e,alpha_m_f];

    %% setting up alpha_n variables (TO EDIT)
    alpha_n_a = alphas(10);
    alpha_n_b = alphas(11);
    alpha_n_c = alphas(12);
    alpha_n_d = alphas(13);
    alpha_n_e = alphas(14);
    alpha_n_f = alphas(15);
    a_n_vals = [alpha_n_a,alpha_n_b,alpha_n_c,alpha_n_d,...
    alpha_n_e,alpha_n_f];

    %% setting up beta_m variables
    beta_m_a = betas(1);
    beta_m_b = betas(2);
    beta_m_c = betas(3);
    b_m_vals = [beta_m_a,beta_m_b,beta_m_c];
```

```
38
39      %% setting up beta_n variables
40      beta_n_a = betas(4);
41      beta_n_b = betas(5);
42      beta_n_c = betas(6);
43      b_n_vals = [beta_n_a,beta_n_b,beta_n_c];
44
45      %% setting up beta_h variables
46      beta_h_a = betas(7);
47      beta_h_b = betas(8);
48      beta_h_c = betas(9);
49      beta_h_d = betas(10);
50      beta_h_e = betas(11);
51      b_h_vals = [beta_h_a,beta_h_b,beta_h_c,beta_h_d,beta_h_e];
52
53
54      V(1)=init_potential;
55
56      temp_vals_ah = [a_h_vals(1),a_h_vals(2),a_h_vals(3)];
57      temp_vals_am = [a_m_vals(1),a_m_vals(2),a_m_vals(3),...
58      a_m_vals(4),a_m_vals(5), a_m_vals(6)];
59      temp_vals_an = [a_n_vals(1),a_n_vals(2),a_n_vals(3),...
60      a_n_vals(4),a_n_vals(5), a_n_vals(6)];
61
62      temp_vals_bm = [b_m_vals(1),b_m_vals(2),b_m_vals(3)];
63      temp_vals_bn = [b_n_vals(1),b_n_vals(2),b_n_vals(3)];
64      temp_vals_bh = [b_h_vals(1),b_h_vals(2),b_h_vals(3),...
65      b_h_vals(4),b_h_vals(5)];
66
67      m(1)=am(temp_vals_am,V(1))/(am(temp_vals_am,V(1))+...
68      bm(temp_vals_bm,V(1))); % Initial m-value
69      n(1)=an(temp_vals_an,V(1))/(an(temp_vals_an,V(1))+...
70      bn(temp_vals_bn,V(1))); % Initial n-value
71      h(1)=ah(temp_vals_ah,V(1))/(ah(temp_vals_ah,V(1))+...
72      bh(temp_vals_bh,V(1))); % Initial h-value
73
74
75  %% RK4 Method
76  V(1)=init_potential; % Initial Membrane voltage
77  m(1)=am(temp_vals_am,V(1))/(am(temp_vals_am,V(1))...
78  +bm(temp_vals_bm,V(1))); % Initial m-value
79  n(1)=an(temp_vals_an,V(1))/(an(temp_vals_an,V(1))...
80  +bn(temp_vals_bn,V(1))); % Initial n-value
81  h(1)=ah(temp_vals_ah,V(1))/(ah(temp_vals_ah,V(1))...
82  +bh(temp_vals_bh,V(1))); % Initial h-value
83
84  len = length(t);
85
86  for i=1:len-1 % Loop through each step until time is finished
87  %     disp(i);
88      %4 step method of Runge-Kutta
89      K1=dt*HH(i,i,len,current,[V(i); n(i); m(i); h(i)],...
90      temp_vals_ah,temp_vals_am,temp_vals_an,temp_vals_bh,...
91      temp_vals_bm,temp_vals_bn);
```

```
92      k1=K1(1,1);n1=K1(2,1);m1=K1(3,1);h1=K1(4,1);
93      K2=dt*HH(i,i+(0.5*dt),len,current,[V(i)+(0.5*k1);...
94      n(i)+(0.5*n1);m(i)+(0.5*m1);h(i)+(0.5*h1)],temp_vals_ah,...
95      temp_vals_am,temp_vals_an,temp_vals_bh,temp_vals_bm,...
96      temp_vals_bn);
97      k2=K2(1,1);n2=K2(2,1);m2=K2(3,1);h2=K2(4,1);
98      K3=dt*HH(i,i+(0.5*dt),len,current,[V(i)+(0.5*k2);...
99      n(i)+(0.5*n2);m(i)+(0.5*m2);h(i)+(0.5*h2)],temp_vals_ah,...
100     temp_vals_am,temp_vals_an,temp_vals_bh,temp_vals_bm,...
101     temp_vals_bn);
102     k3=K3(1,1);n3=K3(2,1);m3=K3(3,1);h3=K3(4,1);
103     K4=dt*HH(i,i+dt,len,current,[V(i)+k3;n(i)+n3;m(i)+m3;h(i)+h3],...
104     temp_vals_ah,temp_vals_am,temp_vals_an,temp_vals_bh,...
105     temp_vals_bm,temp_vals_bn);
106     k4=K4(1,1);n4=K4(2,1);m4=K4(3,1);h4=K4(4,1);
107     %create next step for each variable
108     V(i+1)=V(i)+1/6*(k1+2*k2+2*k3+k4);
109     n(i+1)=n(i)+1/6*(n1+2*n2+2*n3+n4);
110     m(i+1)=m(i)+1/6*(m1+2*m2+2*m3+m4);
111     h(i+1)=h(i)+1/6*(h1+2*h2+2*h3+h4);
112 end
113 %set variables for graphing later
114 RK=V;
115 RKm=m;
116 RKn=n;
117 RKh=h;
118 clear V m n h;
119
120 function a=ah(vals,v) %Alpha value for variable h
121 a=vals(1)*exp(vals(2)*(v+vals(3)));
122
123 function a=am(vals,v) %Alpha for Variable m
124 a=vals(1)*(v+vals(2))/(vals(3)-vals(6)*exp(-(v+vals(4))/vals(5)));
125
126 function a=an(vals,v)%Alpha for variable n
127 a=vals(1)*(v+vals(2))/(vals(3)-vals(6)*exp(-(v+vals(4))/vals(5)));
128
129 function b =bh(vals,v) %beta value for variable h
130 b=vals(1)/(vals(2)+vals(5)*exp(-(vals(3))*(v+vals(4))));
131
132 function b=bm(vals,v) %Beta for variable m
133 b=vals(1)*exp(vals(2)*(v+vals(3)));
134
135 function b=bn(vals,v) %Beta for variable n
136 b=vals(1)*exp(-(v+vals(2))/vals(3));
137
138 function dydt = HH(index,i,len,current,y,...
139 temp_vals_ah,temp_vals_am,temp_vals_an,temp_vals_bh,...
140 temp_vals_bm,temp_vals_bn)
141 % Constants
142 ENa=55.17; % mv Na reversal potential
143 EK=-72.14; % mv K reversal potential
144 El=-49.42; % mv Leakage reversal potential
145 gbarNa=1.2; % mS/cm^2 Na conductance
```

```
gbarK=0.36; % mS/cm^2 K conductance
gbarl=0.003; % mS/cm^2 Leakage conductance

I=zeros(1,len);
I(round(length(I)*0.1):end)=current;

Cm = 0.01; %Membrane Capacitance
% Values set to equal input values
V = y(1);
n = y(2);
m = y(3);
h = y(4);
gNa=gbarNa*m^3*h;
gK=gbarK*n^4;
gl=gbarl;
INa=gNa*(V-ENa);
IK=gK*(V-EK);
Il=gl*(V-El);

%Hodgkin-Huxley Model Equation
dydt = [((1/Cm)*(I(index)-(INa+IK+Il))); an(temp_vals_an,V)*...
(1-n)-bn(temp_vals_bn,V)*n; am(temp_vals_am,V)*...
(1-m)-bm(temp_vals_bm,V)*m; ...
    ah(temp_vals_ah,V)*(1-h)-bh(temp_vals_bh,V)*h];
```

4. **fitness_calculation.m**

The main program used to calculate the chromosome fitness.

```

function e = GA_HH_fitness_NEW(HH_param_in)
    working_vals_ah = ...
        [HH_param_in(01),HH_param_in(02),HH_param_in(03)];
    working_vals_am = [HH_param_in(04),HH_param_in(05),...
    HH_param_in(06),HH_param_in(07),HH_param_in(08),HH_param_in(25)];
    working_vals_an = [HH_param_in(09),HH_param_in(10),...
    HH_param_in(11),HH_param_in(12),HH_param_in(13),HH_param_in(26)];
    working_vals_bm = ...
        [HH_param_in(14),HH_param_in(15),HH_param_in(16)];
    working_vals_bn = ...
        [HH_param_in(17),HH_param_in(18),HH_param_in(19)];
    working_vals_bh = [HH_param_in(20),HH_param_in(21),...
    HH_param_in(22),HH_param_in(23),HH_param_in(27)];
    current = HH_param_in(24);

    %%
    init_alphas = [working_vals_ah,working_vals_am,working_vals_an];
    init_betas = [working_vals_bm,working_vals_bn,working_vals_bh];

    %% Load reference data
    temp = load('phasic_bursting');
```

```
20      f = fieldnames(temp);
21      AP_IZ = temp.(f{1});
22      % Smoothing Spikes
23      AP_IZ = smooth(AP_IZ,10,'lowess');
24      AP_IZ = AP_IZ'+5;
25      time_IZ = temp.(f{2});
26      clear f temp
27
28      global old_min;
29      global dir;
30
31      %%
32      [AP_HH, time_HH] = GA_HH_simulate(init_alphas, ...
            init_betas,current);
33      e_regular = compute_error(AP_IZ,AP_HH);
34
35      %% Peak error calculation
36      [ymax_HH,imax_HH,ymin_HH,imin_HH] = extrema(AP_HH);
37      [ymax_IZ,imax_IZ,ymin_IZ,imin_IZ] = extrema(AP_IZ);
38
39      e_max = compute_peak_error(imax_IZ,ymax_IZ,imax_HH,...
40      ymax_HH,length(time_IZ));
41      e_min = compute_peak_error(imin_IZ,ymin_IZ,imin_HH,...
42      ymin_HH,length(time_IZ));
43
44      %%
45
46      e = e_regular + e_max + e_min;
47      PER_error = percentage_error(AP_IZ, AP_HH);
48
49      if (e < old_min)
50          old_min = e;
51          figure(123);
52          plot(time_IZ, AP_IZ,'r',time_HH,AP_HH,'b');
53          title(sprintf('Fitness = %f (%.2f%%)',e, PER_error));
54
55          fig_path = ...
                sprintf('%s\\intermediate_result_%d-%d.fig',dir,...
56          round(e), round(PER_error));
57          saveas(123,fig_path);
58
59          txt_path = sprintf('%s\\HH_param_for_figures.txt',dir);
60          fID = fopen(txt_path,'a');
61          fprintf(fID,'fitness = %f \tHH Parameters = ',e);
62          fprintf(fID,'%f, ',HH_param_in);
63          fprintf(fID,'\n');
64          fclose(fID);
65      end
66
67  function e = percentage_error(AP_IZ, AP_HH)
68      e =((AP_IZ - AP_HH)*(AP_IZ - AP_HH)')/(AP_HH*AP_HH')*100;
69
70
71  function error = compute_error(AP_IZ, AP_HH)
```

```
72      if max(AP_HH) > 100 || min(AP_HH)< -100
73          error = 1e10;
74      else
75          error = ((AP_IZ-AP_HH)*(AP_IZ-AP_HH)'/length(AP_IZ)).^2;
76      end
77
78  function peak_error = compute_peak_error(i1,y1,i2,y2,len)
79      compute_peak_error_external(i1,y1,i2,y2,len);
```

5. `GA_run.m`

A template program of the genetic algorithm used to run the simulations.

```
1   function run_S_GA()
2       clc;
3       %% Generate (RANDOM) initial population
4       new_init_population_new_data;
5       population = init_pop;
6       clear init_pop;
7
8       %% Define GA parameters
9       % Elite Count
10      EC = 0.1;
11      % Cross Over
12      CO = 0.7;
13      % stall_generations = 200;
14      fitness_limit = 200;
15
16      fitness_vals = zeros(pop_size,1);
17      stop_criterion_met = false;
18
19      GA_index = 0;
20
21      while ¬stop_criterion_met
22          GA_index = GA_index + 1;
23          fprintf('GA iteration = %d\n', GA_index);
24      %% individual selection
25          n_elites = round(pop_size * EC);
26          n_cross_overs = round(pop_size*(1 - EC)*CO);
27          n_mutants = pop_size - (n_elites + n_cross_overs);
28
29          if n_mutants < 0
30              n_mutants = 0;
31          end
32
33          elites = population(1:n_elites, :);
34          parent_select = randperm(pop_size, ceil(n_cross_overs*0.5));
35          parents = population(parent_select, :);
36          mutant_select = randperm(pop_size, ceil(n_mutants*0.5));
37          mutants = population(mutant_select, :);
38
```

```
39      %% reproduction
40      % Cross-Over
41      elite_children = cross_over(elites, ceil(n_cross_overs*0.5));
42      regular_children = cross_over(parents, ceil(n_cross_overs*0.5));
43
44      % Mutation
45      elite_mutants = mutate(elites, ceil(n_mutants*0.5));
46      regular_mutants = mutate(mutants, ceil(n_mutants*0.5));
47
48      population = [elites; elite_children; regular_children;...
49                    elite_mutants; regular_mutants];
50
51      temp_length = length(population);
52
53      if temp_length > pop_size
54          % population(pop_size+1:end, :) = [];
55          del = randperm(pop_size, temp_length-pop_size);
56          population(del, :) = [];
57      else
58          disp('Population size incorrect');
59          break;
60      end
61
62      %% Find the fitness
63          for i=1:pop_size
64              fitness_vals(i) = GA_HH_fitness_NEW(population(i,:));
65          end
66
67          data_mat = [population fitness_vals];
68          [¬, col] = size(data_mat);
69
70          % Sort the data matrix (in 'DESCENDING' order) by ...
                fitness values
71          sortrows(data_mat, -col);
72
73      %% Check STOP criterion
74          stop_criterion_met = check_stop_criterion...
75          (data_mat, col, fitness_limit);
76      end
77
78  function SCM = check_stop_criterion(data, fitness_col, ...
        fitness_limit)
79      if data(1, fitness_col) < fitness_limit
80          SCM = true;
81      else
82          SCM = false;
83      end
84
85  function children = cross_over(parents, n_children)
86      [n_parents, col] = size(parents);
87      children = zeros(n_children, col);
88
89      index = 0;
90
```

```
    while index < n_children
        parent_select = randperm(n_parents, 2);
        temp_parents = parents(parent_select, :);

        break_point = randi([1,col-1], 1, 1);
        child1 = [temp_parents(1, 1:break_point),...
        temp_parents(2, break_point+1:end)];
        child2 = [temp_parents(2, 1:break_point),...
        temp_parents(1, break_point+1:end)];

        children(index+1, :) = child1;
        children(index+2, :) = child2;
        index = index+2;
    end

function children = mutate(parents, n_children)

    [n_parents, col] = size(parents);

    % Selection
    if n_children ≤ n_parents
        selection = randperm(n_parents, n_children);
        children = parents(selection, :);
    else
        selection = randperm(n_parents, n_children - n_parents);
        children = [parents; parents(selection, :)];
    end


    % Mutation using loops
    for i = 1:n_children
        temp = randi([1, col],1);
        element = children(i, temp);
        element = (2*element).*rand(1,1) - element;
        children(i, temp) = children(i, temp) + element;
    end
```

CURRICULUM VITAE

Saliya Kirigeeganage
Electrical and Computer Engineering
University of Louisville, Louisville, KY 40292
skkiri01@louisville.edu

Education

2018 **Ph.D. Electrical Engineering**, University of Louisville, Louisville, KY.
2013 **M.Sc. Physics**, University of Louisville, Louisville, KY.
2011 **B.Sc. Physics**, University of Peradeniya, Peradeniya, Sri Lanka.

Professional Experience

Research Engineer, February 2016 - Present

- Portable Tocodynamometer
  - Programmed a portable tocodynamometer product used to measure contractions in pregnant women.
  - Developed embedded software running on a microcontroller that reads force sensor data and transmits via Bluetooth to use with mobile devices.
- Wireless Medical Monitor for Assisted Living Facilities
  - Designed system architecture for a wearable medical device used to improve nursing response time.
  - Programmed a wireless access point client that reads Bluetooth data from the wearable device.
  - The data is then transferred to a server via TCP/IP where data is processed, status displays are generated, and pages are sent to staff.
  - Programming was performed using Python and JavaScript.
- Therapeutic Feeding Device

  - Programmed a therapeutic baby bottle product used to improve infant feeding. The design consisted of low power SPI sensors, Atmel 328P microcontroller, and a Bluetooth LE transceiver.
  - Programming was performed in C and a proprietary language used by the Bluetooth module.

- Smart Grid Appliance Interface
  - Programmed a smart grid communication interface based on Renesas RX231 microcontroller for General Electrics hybrid electric water heater.
  - The interface bridged a control protocol developed by the Energy Power Research Institute with the proprietary GE control interface.

Instructor/Teaching Assistant, August 2011 - Present

- Instructor for ECE computer tools - Matlab (ECE 322)
- Instructor for Introduction to electrical engineering (ECE 252)
- Graduate teaching assisted for multiple ECE and Physics courses
  - Computer design (ECE 510/511)
  - Integrated circuit design (ECE 533/534)
  - Introduction to embedded systems (ECE 412)
  - Introduction to VLSI systems Lab (ECE 515)
  - Fundamentals of Physics lab (PHYS 223)
  - Introduction to modern physics (PHYS 300/301)
  - Atomic and nuclear lab (PHYS 351)
  - Elementary astronomy (PHYS 108)

Graduate Research Assistant, August 2012 - August 2015

- Neural Stimulator for spinal cord stimulation
  - Performed finite element analysis simulations of the electric field distributions of the spinal cord using COMSOL.
  - Performed electro-chemical simulations using COMSOL to visualize the charge distributions in implanted electrodes under spinal cord stimulation.
  - Programmed MATLAB software to optimize stimulus timing in spinal cord stimulation.
  - Designed a digital to analog converter PCB to drive an electrode array.
  - Programmed software (VHDL and C) to drive an electrode array through a digital to analog converter.

  - Developed an android interface for a stimulator using MIT App inventor.

- Computational modeling of Boron-Nitride sheets
  - Performed particle swarm optimization simulations on Boron-Nitride sheets to tune band gaps.
  - Developed FORTRAN programs to perform simulations and optimize workflow.

Software Developer, January 2017 - April 2017

- Developed a web scraping application for a real estate company to extract relevant data from a large web based database as the database updates.
- The user interface and the back end of the application was developed using Python.

Publications and Presentations

- **S. Kirigeeganage**, J. Naber, D. Jackson, R. Keynton, T. Rousell. "Pulse position optimization to minimize charge accumulation in biological tissue during spinal cord stimulation".(In preparation).
- **S. Kirigeeganage**, D. Jackson, J. Zurada, J. Naber. "Optimizing the Hodgkin Huxley parameter equations using a genetic algorithm to simulate the bursting behavior ". *2018 Sensors, 18, 3051.* (In Submission)
- **S. Kirigeeganage**, D. Jackson, J. Zurada, J. Naber. "Modeling the Bursting Behavior of the Hodgkin-Huxley Neurons Using Genetic Algorithm Based Parameter Search". *2018 IEEE International Symposium on Signal Processing and Information Technology.* (Accepted for publication)
- **S. Kirigeeganage**, D. Jackson, J. Naber, K. Heyl. "Wireless Incontinence Monitoring System for Improved Heath". *2018 KY Nanotechnology and Additive Manufacturing Symposium.*
- **S. Kirigeeganage**, J. Naber, D. Jackson, R. Keynton, T. Rousell. "Simulating Spiking Neurons Using a Simple Mathematical Model". *2016 COMSOL Conference.*
- **S. Kirigeeganage**, J. Naber, D. Jackson, R. Keynton, T. Rousell. "Spinal Cord Stimulation Modeling Based on an Extracellular Electrical Neural Model".*2015 Research Louisville* (Poster).