Neural Network Processing: Modelling and Simulation of Neurons and Neural networks

Manu Jayadharan
MS student in Mathematics
Indian Institute of Science Education and Research, Mohali
manujhuhaha@gmail.com

July 28, 2015

Department of Avionics

Indian Institute of Space Science and Technology

Thiruvanthapuram - 695 547, Kerala.

August 19, 2015

CERTIFICATE

This is to certify that this project report entitled "**Neural Network Processing: Modelling and Simulation of Neurons and Neural networks**' by **Mr. Manu Jayadharan**, MS student in Mathematics, Indian Institute of Science Education and Research, Mohali is a record of his work carried out under my supervision and guidance in partial fulfillment of the requirements for the internship in Avionics department from **"18.05.2015" to "28.07.2015".** He has done an excellent work and can lead to a publication in some reputed conference or journal. Further the simulation engine he has developed can be extended for various topological studies among neurons. I wish him all the best for his research career.

Deepak

Dr. Deepak Mishra
Associate Professor
Department of Avionics
Indian Institute of Space Science and Technology Trivandrum

Dr. Deepak Mishra
Assistant Professor
Department of Avionics
Indian Institute of Space Science and Technology
Department of Space, Govt. of India
Thiruvananthapuram-695 547

Acknowledgement

I would like to express my gratitude towards National Network for Mathematical and Computational Biology(NNMCB) for giving me an opportunity to work in the field I am interested in as a part of their programme. I am also thankful to Dr.Deepak Mishra who was my official guide for his invaluable suggestions and guidance through out this programme. And for last but not least I would like to thank the authors of the numerous references I made for completing this project.

Abstract

In this report I am preseting the work I have done as a part of an internship programme organised by National Network for Mathematical and Computational Biology. The first part of the report being a very brief introduction to definition and structure of neurons and neural signal processing so that a more general reader could go through the remainig material, advanced readers can skip this section. Next part is on modelling and simulation of a neurons and neural network. First chapter in this section is the modelling and simulation of a single neuron is covered using the classical Hodgkin-Huxley model. In the next chapter, the coupling of two neurons is modelled and simulated using the Leaky Integreate and Fire(LIF) model. In the final chapter, I have made an attempt to create something called a "Variable Topology Neural Network Simulator" which could simulate neural networks of any topology with any kind of synaptic connection with or without latency of synaptic conductance. This simulator is based on the exact solution of mathematical formulation of a network based on LIF model[see Exact simulation of Integrate and Fire models with synaptic conductances,Brette,2006] which makes it computationally highly efficient. Also no advanced softwares is used for simulations, all simulation codes are written in $Fortran95$ which is preferred over other advanced languages because of its computational efficiency which comes in handy while simulating a large nerwork of neuron. All the numerical packages were self written and most of the codes are incorporated with the use of *gnuplot* and $ffmpeg$ to form images and animations to visualise the results.

Part I

Introduction to Neural Signal Processing

0.1 Neuron

Neurons can be considered as the most fundamental and core unit of the nervous system. Neurons are excitable cells which facilitate the propogation of information through electrical and chemical interactions within an organism. Structure of a Neuron can be seen in figure 0.1.1 The sole purpose of Neural signal processing is to understand how information is represented, transmitted and stored in the nervous system and develop mechanisms to record and interpret these signals.

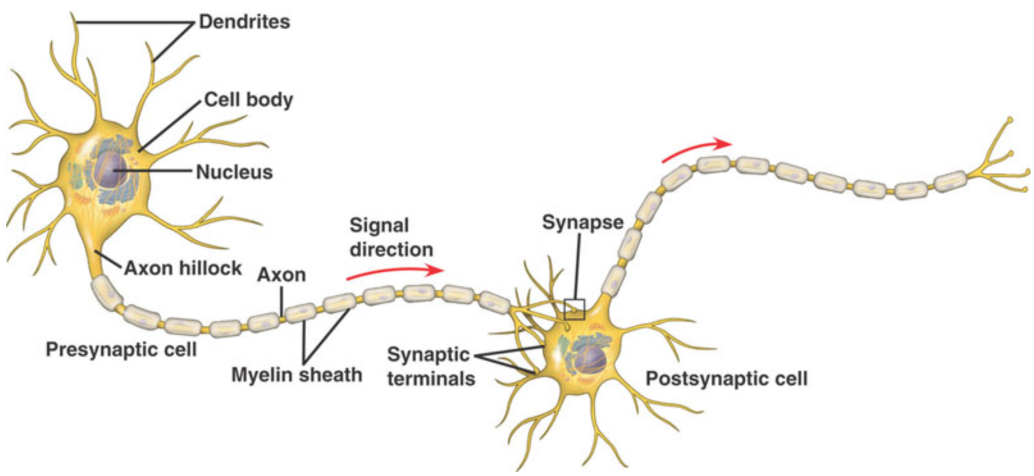


Figure 0.1.1: Two neurons having a synaptic connection
source:internet

0.2 Membrane Potential

Neurons usually maintain a difference in electrical potential across its cell membrane. Unequal distribution of charged ions(Na^+, K^+, Ca^{2+}, Cl^-etc). Cell membrane of neurons has selectively permeable ion channels which facilitates teh movement of corresponding ions.

Membrane potential, $V_m = V_{inside} - V_{outside}$

Resting potential refers to the stable membrane potential of a neuron when there exists an equilibrium in the flow of ions across the membrane. Generally for a typical neuron, resting potential is around -67mV

0.3 Action Potential

Signals are propogated across an excitable neuron with the help of sudden spikes in the membrane potential called propogating spikes or action potential. Formation of a spike is facilitated by slight change in permeability of various ion channells which will lead to depolarization of the cell membrane and a series of feed back mechanisms.

0.4 Synaptic connection

Interaction between two neurons happens through connection mechanism called a synapse. The junction at which facilitates this connection is called a synaptic junction. The connection may be electrical(electric synapse) or through release and binding of chemicals called neural transmitters(chemical synapse).

Part II

Modelling and simulation of neurons and neural network

Chapter 1

Hodgkin-Huxley model

1.1 About the model

Hodgkin-Huxley Model is the classical model for the initiation and propogation of action potential across a neuron. Alan Lloyd and Andrew Huxley received the Nobel prize in physiology/Medicine for the same. H&H model is based on the dynamic behaviour of the conductance of different ions across the membrane. The model consists of a set of differential equations which is able to mimic the behaviour of the neurons taking account of the differential behaviour of Na^+ and K^+ion channels and ohmic conductace of leakage current. The parameters of the differential equations are derived mostly from emperical data attained from series of Voltage clamp experiments conducted on squid giant axon.

1.2 Hodkin-Huxley's explanation of formation of Action Potential

Formation of action potential starts when there is a change in the permeability of some ion channels which leads to slight depolarization of the cell membrane. As result more, Na^+ion channels opens and there will be current formation due to inward flow of Na^+ions. This leads to further depolarization and subsequent opening of more Na^+ion channels. This positive feedback continues and the membrane potential, V_m approaches the reverse potential of Na^+ion. Subsequently Na^+ion channels inactivates and K^+ion channels start opening and hyperpolarization happens due to which V_m approaches towards the reverse potential of K^+. Finally V_m retains the resting potential with the additional help of sodium-potassium pump.

1.3 Equations of H&H model

$$I_m = I_C + I_l + I_{Na} + I_K$$

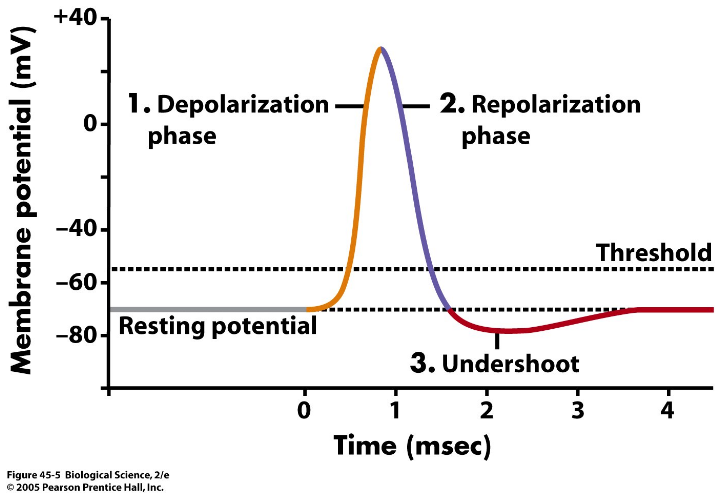


Figure 1.2.1:

where, I_m is the total membrane currnet, $I_C$is the current due to current due to capacitance of the cell membrane, I_l is the leakage current and $I_{Na}$and $I_K$refers to the sodium and potassium current.

$$I_c = C_m \frac{dV_m}{dt}$$

$$I_l = g_l(V_m - E_l)$$

$$I_{Na} = g_{Na}(V_m - E_{Na})$$

$$I_K = g_K(V_m - E_K)$$

where, g_i refers to corresponding conductance and E_i refers to corresponding reverse potential.

$$g_{Na} = g'_{Na} m^3 h$$

$$g_K = g'_K n^4$$

where, g'_i refers to the maximum ionic conductance of i-ion, m and n are the gating variables which corresponds to activation and inactivation of sodium channels and n refers to the opening of potassium channels.

Modelling and Parameters of the gating variables is given in Appendix A.

1.4 Simulation of Hodgkin-Huxley model

1.4.1 Coding

Simulation is done by solving the equations of hodgkin huxley model with a constant current supply for a fixed amount of period. Also parameters and scaled so that the resting potential V_m becomes zero.

Lot of open sourced and paid softwares are available for the simulation and visualisation of the Model which is not used in this work. All codes were written in $Fortran95$ and all packages are self written. Numerical integration is first done with Euler method which is later improved using runge-kutta of order 4. The code also incorporates the facility of automatically plotting the simulated data and making an mp4 animation using gnuplot and ffmpeg. complete code with animated videos can be found at `https://www.dropbox.com/sh/7bd3hueoui1kcfq/AAB2iR21nXP1owCtnhOmUf65a?dl=0`

The dropbox folder also contains an executable file named *hodgkin* which ask for the applied current which is to be used for the simulation. One of the final version of the code is also given in Appendix A.

---

compilation instruction in linux:∼ f95 -o filename hodgkin_huxley_simulator.f95

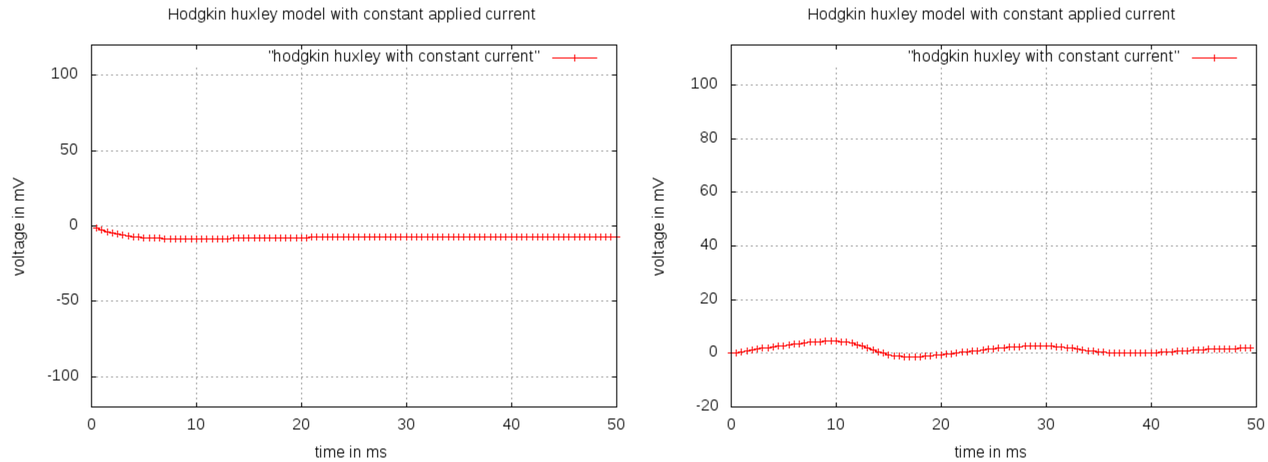


Figure 1.4.1: Simulation of the neuron with applied current $I_{app} = 0mA$(left) and $I_{app} = 3.6mA$(right): no spiking is formed because the voltage failed to reach the threshold required for firing

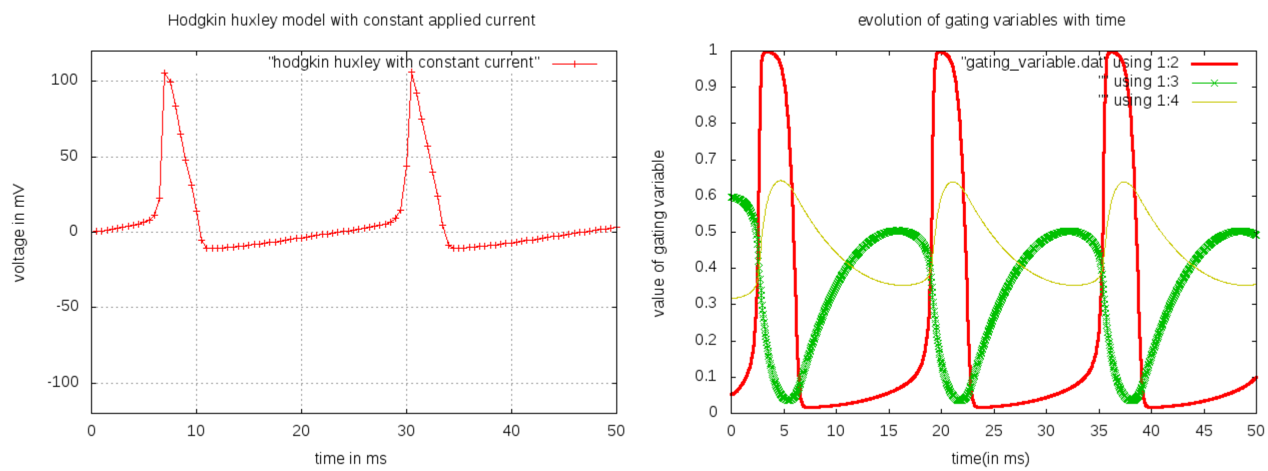
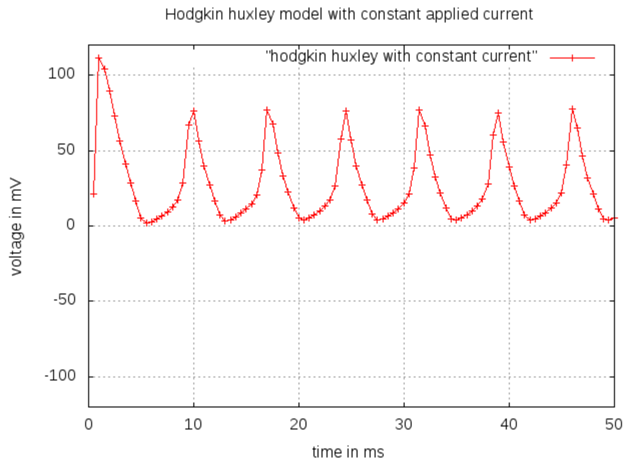


Figure 1.4.2: Applied current $I_m = 4.0mV$. figure on the right shows evolution of gating variables. bold line:m,thin line:n,line with dots:h



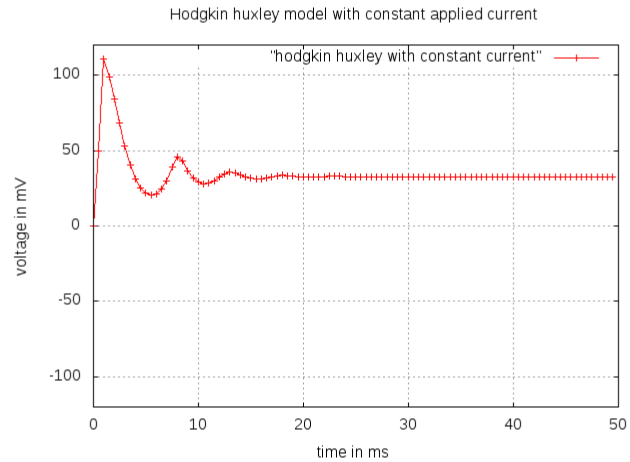


Figure 1.4.3: Applied current $I_m = 50.0mV$ (left), $I_m = 100.0mV$(right). Subsequent firing of neurons on application of high currnet is hindered by the the refractory period.

1.5 Further scope for modification

Other ion conductances like Ca^{+}can be also incorporated into the Hodgkin-Huxley model to get a better realization of the biological model.

Chapter 2

Coupling between two neurons

2.1 Leaky Integrate and Fire (LIF) model

LIF model is a comparitively simple model for a neuron in which the ionic conductances are not explicitely incorperated, instead an ohmic leaky conductance is assumed for this model. It assumes the neuron to be a simple RC circuit. Instead of modelling the spike internally, in this model the spikes are artificially introduced when the membrane potential reaches a threshold voltage V_{th} and resetting it to V_{reset}. The simplicity and computational efficiency of the model makes it a powerful tool to develop more complicated models and networks. The basic equation for an LIF model can be given by

$$C_m \frac{dV}{dt} = -g_l(V - E_l) + I_{applied} \tag{2.1.1}$$

where $C_m$is the membrane capacitance, g_l and E_l are the leakage conductance and reverse potential respectively.

2.2 Modelling the coupling

Coupling of two neurons here refers to the electrical interaction between two neurons having an electrical connection. LIF model can be used to model this situation by introducing a term for ohmic conductance, g_c between two neurons[see The Effects of Potassium Currents on the Synchronization of Electrically Coupled Neural Oscillators , Middleton ,2005] to get the following coupled differential equations:

$$\begin{cases} C_m \frac{dV_1}{dt} = & -g_l(V_1 - E_l) + I_{applied} + g_c(V_2 - V_1) \\ C_m \frac{dV_2}{dt} = & -g_l(V_2 - E_l) + I_{applied} + g_c(V_1 - V_2) \end{cases} \tag{2.2.1}$$

An explicit spike is added at $V_{th} = 1mV$.

Eqution (2.2.1) after some algebraic manipulations and substitutions will give

$$\begin{cases} \frac{dv_1}{dt'} = & -v_1 + I + g'_c(v_2 - v_1) \\ \frac{dv_2}{dt'} = & -v_2 + I + g'_c(v_1 - v_2) \end{cases} \tag{2.2.2}$$

where $t' = \frac{t}{(C_m/g_l)}$, $v_i = \frac{(V_{reset}-V_i)}{V_{reset}-V_{th}}$, $g'_c = \frac{g_c}{g_l}$, $I = \frac{I_{applied}+g_l(E_l-V_{reset})}{g_l(V_{th}-V_{reset})}$

Theory of weakly coupled oscillators can be used to reduce equation (2.2.2) to form a single phase equation which can then be solved analytically and can be used to study the existance and stability of the phase-locking states of the coupled neurons. See THE EFFECTS OF POTASSIUM CURRENTS ON THE SYNCHRONIZATION OF ELECTRICALLY COUPLED NEURAL OSCILLATORS , MIDDLETON ,2005. In this work, Synchronious and anti-synchronious behaviour of the coupling explained in this publication is reproduced by numerical simulation of equation (2.2.2).

2.3 Simulation of Coupling of two neurons

2.3.1 Coding for the simulation

Simulation is done by solving the coupled equations in (2.2.2) using RK4.

A constant current $I_{applied}$ is applied to the neurons through out the simulation.

Artificial peaks were produced once the voltage once reaches the threshold of $1mV$.

All codes for this simulation were written in $Fortran95$ and all packages are self written. Numerical integration is done using runge-kutta of order 4. The code also incorporates the facility of automatically plotting the simulated data and making an mp4 animation using gnuplot and ffmpeg. A module is written and used for this by the name "plotting_module" complete code with animated videos can be found at `https://www.dropbox.com/sh/qs4f3on3oo1fcvz/AAAu5XrPbo_nFdw3pGyEBUDea?dl=0`

compilation instruction in linux:∼ f95 -c plotting_module.f95	
:∼ f95 -o filename lifmodel.f95	

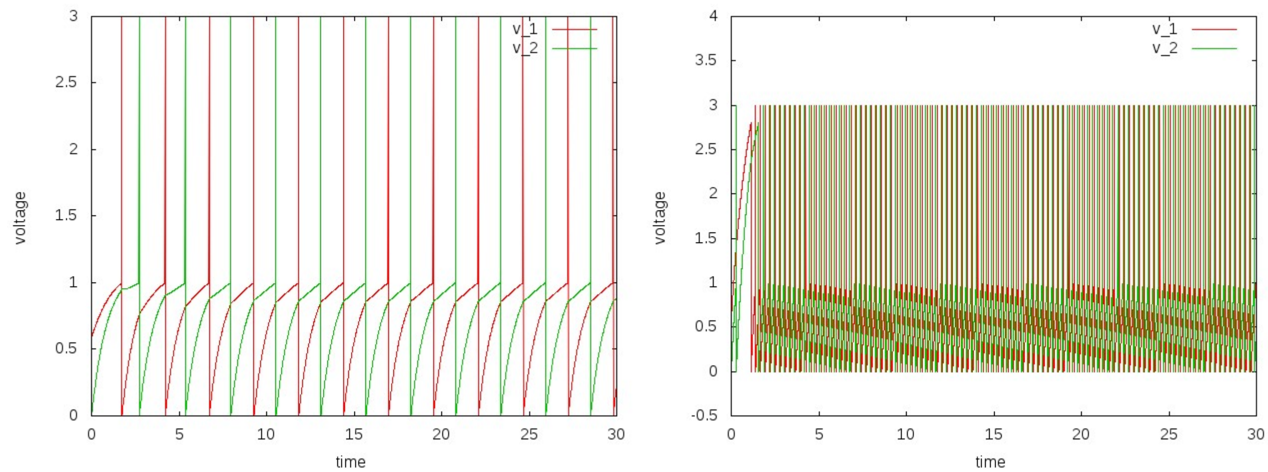


Figure 2.3.1: Simulation of equations (2.2.2)for a couple of neurons using $I_{applied} = 1.2mA$(left) and $I_{applied} = 2.0mA$(right). In the left image a stable anti-synchronous behaviour and a synchronising behaviour and increase in frequency can be seen in the right image. This can be explanied by the change in the intensity of applied current using the work done in The Effects of Potassium Currents on the Synchronization of Electrically Coupled Neural Oscillators , Middleton ,2005.

Chapter 3

Modelling of Neural Network

The complexity of neural networks arises not from the specialisation of neurons, but from their topology and nature of interconnections. Understanding how the topology of neural network affect signal propogation across a network seemed to be a very interesting feild of study as it might give some insight into how similar action potentials can be used to propogate complex information from and to the brain.

3.1 Variable Topology Neural Network Simulator

Variable Topology Neural Network Simulator(VTNNS) is a simulator which is based on a Mathematical model that can simulate or mimic the behaviour of signal propogation in a neural network with any specified topology and synaptic relation. VTNNS should be based on an algorithm which gives high computational efficiency so that signal propogation in large network of neurons can be simulated in feasable amount of time. My attempt is to make a simplified but effective VTNNS based on the LIF model. VTNNS could be used to get a realization of whats happening in a large and complex neural network once we get an idea of the nature of synaptic connections. Modelling is done in such a way that any kind of synaptic interaction can be incorporated into the VTNNS. Efficient improvement from the basic code written is suggested whenever possible as grey notes.

3.1.1 Mathematical Model

LIF model of a single neuron with time constant τ can be described by the following equation:

$$\tau \frac{dV}{dt} = -(V - V_0) \tag{3.1.1}$$

Now after accounting for the synaptic interactions in this model, we will get

$$\tau\frac{dV}{dt} = -(V - V_0) - g^+(t)(V - E^+) - g^-(t)(V - E^-) \tag{3.1.2}$$

where g^+ and $g^-$are the total excitory and inhibitory conductance from other neurons in the network relative to the leak conductance and $E^+$and $E^-$are the excitory and inhibitory reversal potential respectively.

Here only one kind of excitory and inhibitory synaptic relation is assumed. More synaptic relation can be introduced by adding more E^i coressponding to different ionic species.

Synaptic conductance g is assumed to follow exponential decay with time constant τ_s

$$\tau_s\frac{dg^i}{dt} = -g^i \tag{3.1.3}$$

The above model can be simulated either using time driven simulations(slowly progressing with time using numerical integration methods) or by event-driven simulation(progresses by going from one firing event to another firing event)

We are using the latter for our simulations, the reason being the computational efficiency of the latter relative to the former[see Exact simulation of Integrate and Fire models with synaptic conductances,Brette,2006] which comes in handy for simulation of large network of neurons.

We assume that both the excitory and inhibitory conductance has same time constant τ_s, this is a trade-off we make to get an exact solution of the equation (3.1.2) which can then be used to develop the simulator.

In equation (3.1.2), we assume $V_0 = 0$ and express the time constant in units of τ to get :

$$\frac{dV}{dt} = -V + (g^+(t) + g^-(t))(E_s(t) - V) \tag{3.1.4}$$

where $E_s(t) = \frac{g^+(t)E^+ + g^-(t)E^-}{g^+(t)+g^-(t)}$.

$E_s$can be seen as the effective synaptic reverse potential at time t which dynamically depends on the synaptic conductance $g(t)$.

In equation (3.1.3) and (3.1.4), we substitute $g^+ + g^- = g$ to get:

$$\frac{dV}{dt} = -V + (E_s(t) - V)g \tag{3.1.5}$$

$$\tau_s\frac{dg}{dt} = -g \tag{3.1.6}$$

equation (3.1.5) can be solved to get

$$V(t) = -\rho(1-\tau_s, \tau_s g(t))\tau_s E_s g(t) + exp(\tau_s(g(t)-g(0))-t)(V(0)+\rho(1-\tau_s, \tau_s g(0))\tau_s E_s g(0) \tag{3.1.7}$$

where, $\rho(a, b) = e^b x^{-a}\gamma(a, b)$ with $\gamma(a, b) = \int_0^b e^{-t}t^{a-1}dt$ which is the incompete gamma integral.

Complete derrivation of the solution is given is given in Appendix-C .

Fast computing numerical libraries are available for efficient calculation of the incomplete gamma integrals.

3.1.2 Designing the simulator

Here we explore a proper algorithm for the simulation of the spiking and its propogation in a neural network. Basic model of the simulator takes topology of the network and initial conditions of the neurons as input and gives a sorted table of possible firing of neurons which can then be plotted for analysis.

- Each neuron in the network has three parameters which need to be updated suitably in the simulation namely: V, g ,E_s and t_0 , where t_0 is the time of last update of the neuron and V, g ,E_s being the state variables of the neuron at t_0
- When a neuron reaches the threshold voltage V_{th}, V is reset to V_{reset}. Both V_{th} and $V_{reset}$are fixed according to the values of time constants and reverse potentials we give.
- We need functions/subroutines to update the parameters of a neuron when it spikes and also to update the parameters of other neurons after the spiking. Also subroutines shoud be defined to find the next firing time of each neuron, which can be finite or ∞in the case of no upcoming spike.
- Synaptic relation between neurons in the network can be defined in the form of a weight matrix W of dimension same as the number of neurons in the network such that $(i,j)^{th}$entry w_{ij} quantitatively represent the change in the synaptic conductance of j^{th}neuron due to the firing of the i^{th} neuron. w_{ij}can be positive(excitory) or negetive(inhibitory).

By the above setup, we assume that the synaptic connection of a neuron to another neuron can either be excitory or inhibitory. In case of different kind of excitory and inhibitory connection possible between neurons, a suitable multidimensional array can be used to represent all kind of synaptic relations possible and hence take the model more close to reality.

VTNNS is modelled in two ways: one without time delay for the transmission of the signal, and one with the time delay. Basic algorithm for both are sketched in the following section.

3.1.2.1 VTNNS without time delay

Algorithm 3.1 without time delay

---

1. Maintain a sorted table of firing time of various neurons in the network.

2. Updating the neuron after it fires: If a neuron is to be first at time t and if $t_0$is the last time of update of the neuron,

   (a) $V \to V_{reset}$

   (b) $g \to g * exp(\frac{-(t-t_0)}{\tau_s})$

3. Updating neuron j after i neuron fires: If a neuron fires at time t, then for each other neuron with last update time t_0,

   (a) $V \to V(t - t_0)$

   (b) $g \to g * exp(\frac{-(t-t_0)}{\tau_s})$

   (c) $E_s \to \frac{gE_S+pw_{ij}+q|w_{ij}|}{g+|w_{ij}|}$ where $p = \frac{E^+-E^-}{2}$and $q = \frac{E^++E^-}{2}$

   (d) $g \to g + |w_{ij}|$

4. Update the firing time table of neurons

3.1.2.2 VTNNS with time delay

Algorithm 3.2 without time delay

1. Suitable time delay should be predefined as t_{lag}.
2. Maintain a sorted table of firing time of neurons and table of time for updating neurons due to synaptic conductance from each neuron.
3. Find t which is the time for next neuron firing and t_{update}which is the time for next update of neuron due to synaptic connection.
4. If $t < t_{update}$, and i is the nueron to be fired with $t_0$being its last update time then,
   (a) $V \to V_{reset}$
   (b) $g \to g * exp(\frac{-(t-t_0)}{\tau_s})$
   (c) if next update time of $i > t + t_{lag}$then, set next update time of i to $t + t_{lag}$
5. If $t_{update} < t$ and t_{update} is the synaptic update due to spike coming from neuron i , then for each other neuron j with last update time t_0,
   (a) $V \to V(t_{update} - t_0)$
   (b) $g \to g * exp(\frac{-(t_{update}-t_0)}{\tau_s})$
   (c) $E_s \to \frac{gE_S+pw_{ij}+q|w_{ij}|}{g+|w_{ij}|}$ where $p = \frac{E^+-E^-}{2}$and $q = E^+ + E^-$
   (d) $g \to g + |w_{ij}|$
6. Update the firing time table of neurons

Note: step 4.(c) automatically incorporates a refractory period for the propogation of spikes since even if one of the neuron fires rapidly only one transmission of the spike through a synapse is possible until the time delay is reached.

3.1.3 Finding the next firing time of a neuron

For each neuron, next firing time is defined as the time at which the voltage V reaches . From equation (3.1.7), it can be seen that V first increses and then decreases, so intitial part of the curve is concave in nature and if V_{th} is in this part of the curve, Newton raphson method can be used to firing time with assured convergance .

But in most cases, next firing time will be ∞ which means the neuron never spikes. To save the computional expenses in cases where the neuron never spikes, we use a series of spiking tests to check whether the neurons spikes before going to the process of finding out the firing time.

First of all, E_S should exceed V_{th} otherwise no spiking is possible. Assume $E_s > V_{th}$. From equations (3.1.5) and (3.1.6) it can be concluded whether a neuron spikes or not solely depends on the initial conditions $V_0$and g_0. If a neuron fires with initial condition (V_0, g_0) , then it fires for any other initial condition (V_1, g_0) with $V_0 < V_1$. So for each g, there exists a minimum voltage $V_{min}(g)$ for which the the neuron fires. So the set of points

$$C = \{V_{min}(g), g\}$$

gives a minimum spiking curve , so that if the initial conditions (V_0, g_0) lies above C , then the neuron is guaranteed to fire. Consider the trajectory in the phase space of solutions of $V(g)$ starting on C from $(V_{0,}g_0)$. This trajectory should be same as C and also should tangential to the threshold $V = V_{th}$, otherwise there would be a trajectory below it which hits the threshold which is not possible. So the minimum firing potential $V_{min}(g)$ can be found by substituting $\frac{dV_{min}}{dg} = 0$ at V_{th} with conductance g_{min}in the equation

$$\frac{dV_{min}}{dg} = \tau_s(1 + 1/g)V_{min} - \tau_s E_s$$

to get:

$$0 = (1 + 1/g_{min})V_{th} - E_s$$

$$g_{min} = \frac{1}{(E_s/V_t) - 1} \tag{3.1.8}$$

Since conductance g decreases with time, there is possiblity of spike in future only if the initial conductance $g_0 > g_{min}$. Once this condition is satisfied, to make sure the neuron fires, we have to check whether $V(g_{min})V_{th}$. $V(g_{min})$ can be found by using equation (3.1.7) to calculate $V(t)$ for t such that $g(t) = g_{min}$.

$$V(g_{min}) = -\rho(1-\tau_s, \tau_s g_{min})\tau_s E_s g_{min} + (\frac{g_{min}}{g_0})^{\tau_s} exp(\tau_s(g_{min}-g_0)-t)(V_0+\rho(1-\tau_s, \tau_s g_0)\tau_s E_s g_0 \tag{3.1.9}$$

Algorithm 3.3 Spike tests

1. Check $E_s > V_t$, if yes then
2. Check $g_0 > g_{min}$, if yes then
3. Check $V(g_{min}) > V_{th}$

Neuron will fire iff the initial conditions passes all the three spike tests above

3.2 Coding for VTNNS

Simulator with and without time lag in synaptic conductance is coded separatley . We have used the algorithms to make two VTNNS setup: one in which the synaptic interaction is very strong and a few neurons can be set to fire once and check the propogation of the signal in the network, the second in which one of the neurons in the network act as an oscilattor(source of signal) with a specified frequency. The frequency of the latter can be modelled as a random process following some distribution like poisson distribution. Both the setups can be used depending on which specific biolophysiological situation of a nerual network that we want to simulate.

All codes are written in $Fortran95$ considering the efficiency of the same to handle huge arrays and comutational efficiency while doing huge calculations. This give us an advantage to simulate large network within feasible computation time. Disadvantage being the difficulty in debugging and the code getting lengthy. Apart from the fast computing library for finding incomplete gamma integral, all other functions and packages for the VTNNS is self-written. Also facility of automatically plotting the simulated data in the form of raster plot is incorporated into the simualaor. Complete codes are available at

`https://www.dropbox.com/sh/bkju25ktl9jip29/AABDjSxazoR0TcBQp13Xizema?dl=0`

One of fortran code for VTNNS with time delay in synaptic conductance is given in Appendix-C

---

compilation instruction in linux:∼ sudo cp libmincog.a /usr/local/lib
:∼sudo cp lib_randomseed_gen.a /usr/local/lib
:∼f95 -o filename VTNNS_with_delay.f95 -lrandom_seed_gen -lmincog numerical.o

---

3.3 Simulation of different topology of neural networks.

To demonstrate the functioning of VTNNS, neural networks of some common simple topologies are simulated. Results in the cases with and without time delay in synaptic conductance is compared in most cases. In all cases following parameters are fixed: $E^{+} = 74.0$, $E^{-} = -6.0$, $\tau = 20ms$, $\tau_s = 5ms$, $V_{th} = 20.0$, $V_{reset} = 14.0$, time delay in transmission of the spike= $0.02ms$, oscillator time period of $0.2ms$. At the beginning of the simulation g_0 is randomly selected from $[0, 0.015]$, and V_0 is randomly selected from $[10, 16]$.

3.3.1 Star Topology

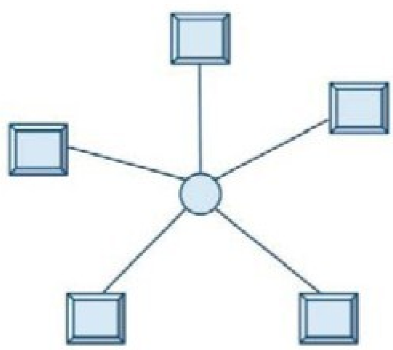


Figure 3.3.1: star topology

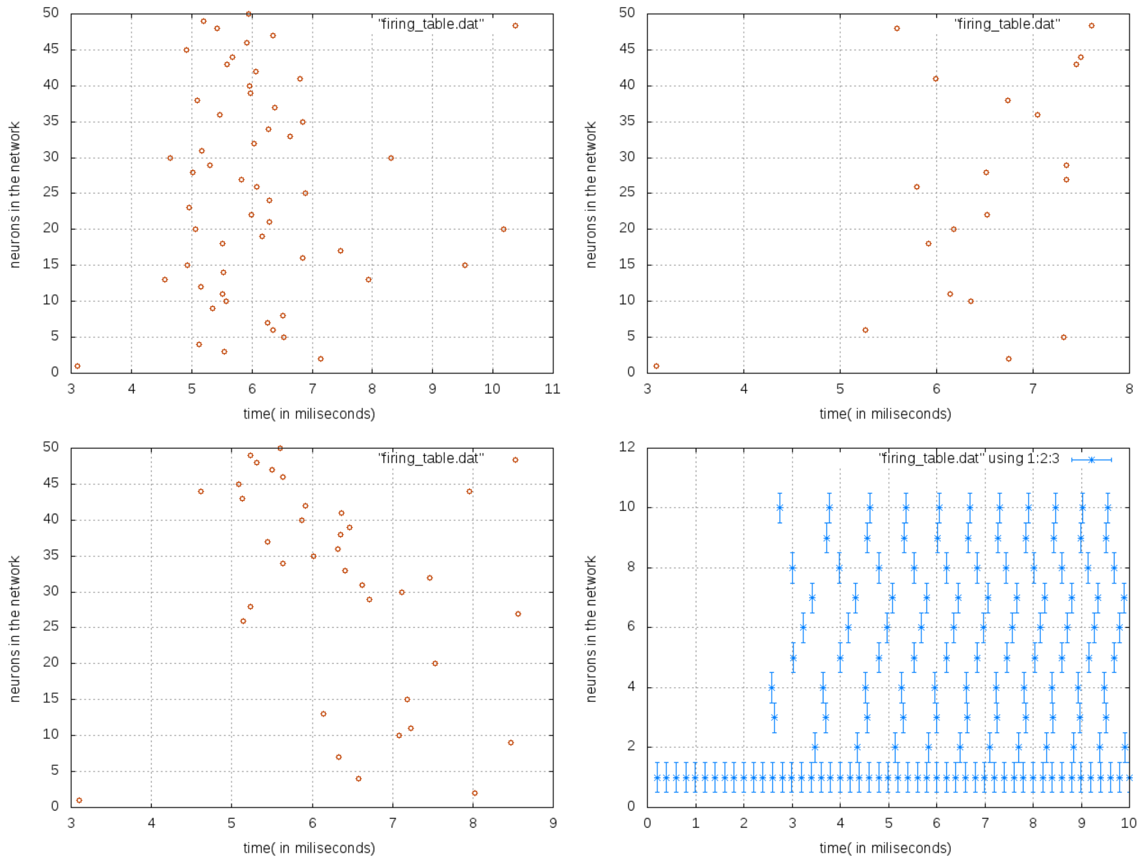


Figure 3.3.2: Raster plot for neural network with star topology simulated using VTNNS without time delay in propogation of spike(top left) and with time delay(top right). $w_{1j} = 1.2$(neuron 1 is connected to all other neurons with an excitory sypansis) . As expected once the first neuron fires, some of the other neurons fire in a scattered manner. There is a delay in the firing pattern of neurons in the right image due to the delay in the propogation of spike. Image on bottom left shows the same simulation with parameter w_{1j} increased to 1.6 for neurons 25 to 50, as a result clear shift in the firing timing of neurons from 25 to 50 can be observed. Image on bottom right shows the simulation in which neuron 1oscillates with time period $0.2ms$, no kind of synchronising behavior is observed even with the change of the oscillation frequency with and without delay in transmission of spike.

3.3.2 Fully connected Topology

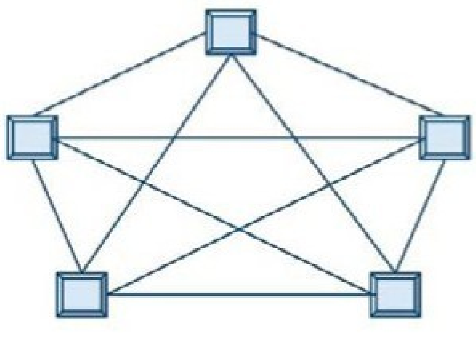


Figure 3.3.3: fully connected topology

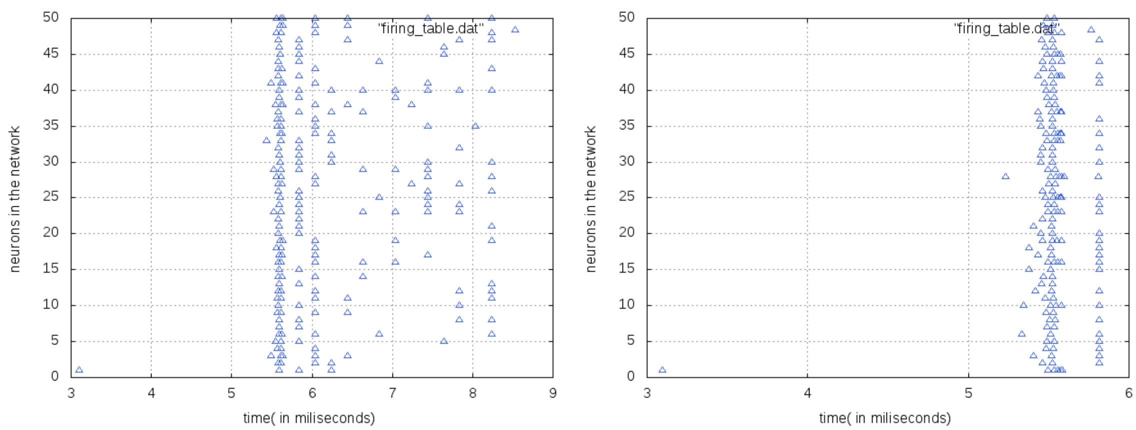


Figure 3.3.4: Raster plot for neural network with fully connected topology simulated using VTNNS without time delay in propogation of spike(top left) and with time delay(top right). $w_{ij} = 1.2$(each pair of neurons is connected to each other using an excitory synapse) . Neurons in the network are seen to be firing in a synchronous manner. This is expected because the topology of the network is in such a way that each pair of neurons is coupled to each other in terms of excitory synaptic relation. In the right image, repeated firing of neurons is restricted, this is due to the refractory period in the firing of neurons included in the algorithm for VTNNS with time delay in propogation of spike.

3.3.3 Ring topology

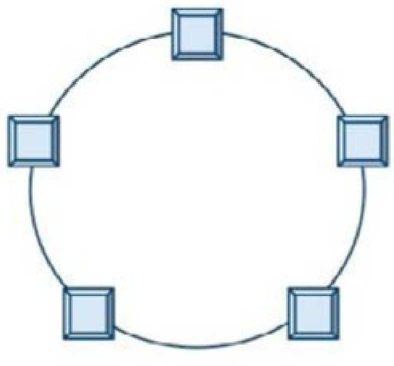


Figure 3.3.5: ring topology

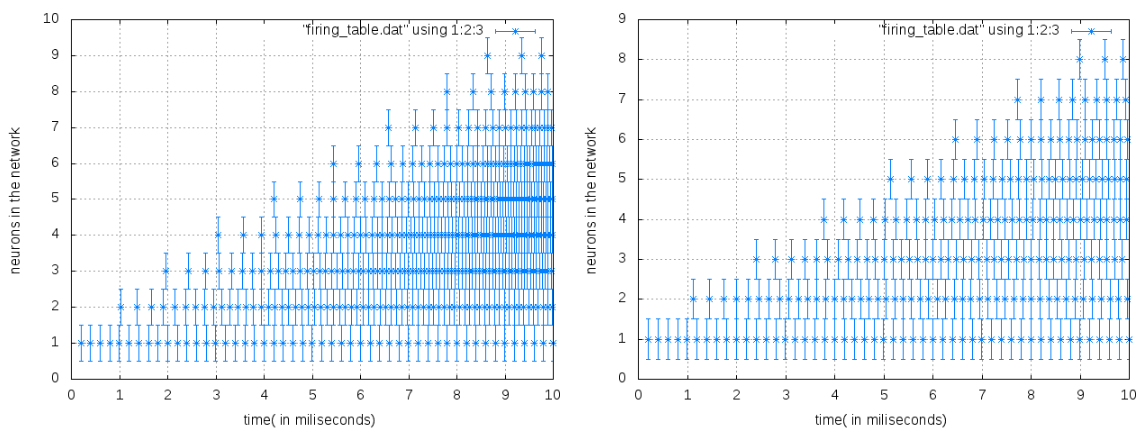


Figure 3.3.6: Raster plot for neural network with ring topology simulated using VTNNS without time delay in propogation of spike(left) and with time delay(right). $w_{i,i+1} = 1.0$ for $i = 1, .., 9$ and $w_{10,1} = 1.0$(each neuron is connected to the next neuron in the order in a circle with an excitory sypansis) . IN both cases a triangular shape is obtained by the plot. Very close synchronising behaviour is also visible in both the cases.

3.3.4 Miscellaneous

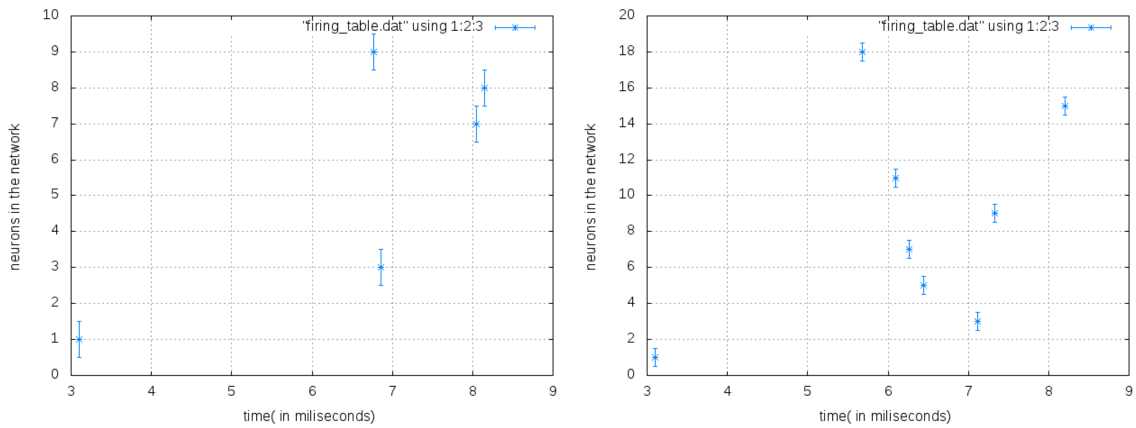


Figure 3.3.7: A wheel spoke kind of toplogy is considered by joining the peripheral neurons in a star shaped Neural netowork using inhibitory synapse.Raster plot of the same is simulated using VTNNS without time delay in propogation of spike(left) and with time delay(right). $w_{1j} = 1.0$(neuron 1 is connected to all other neurons with an excitory sypansis) and $w_{i,i+1} = -1.0$(peripheral neurons in the network are connected with inhibitory synapse in one direction). Less firing happens in both the cases due to influence of negetive sypase on adjacent neurons. In left image, even in the presence of inhibitory relation between adjacent neurons, adjacent neurons fire which is explained by the lack of time delay in the propogation of the spike. Adjacents neurons didnt fire in the right image where time delay of spikes are considered.

3.3.5 Random Network

VTNNS can be easily used to make a random network containing large number of neurons. Probabilistic distributions and functions could be used to define the synaptic relatin between these neurons and also spiking could also be introduced at various nodes following some particular distribution.

3.4 Improving the VTNNS

- Various kinds of synapstic transmitters can be incorporated in the model using multidimensional arrays for the weight matrix explaining synaptic relations.
- Synaptic relation between neurons in the network can be made to evolve with time which can be done by using functions to change entries of the weight matrix considering the current state of a neuron and the nature of the incomign spikes. This needs further work, proper experiments might

be needed to fit functions which actually mimics the behaviour of the neural transmitters and their binding to various receptors in the synaptic junction.

- Each neuron or different section of neurons in the network can be assigned different synaptic time constants which again requires experiments to get an idea of how these values should be assigned to make it useful.
- A more probabilistic model can be used to model VTNNS in which case the whole algorithm need to be rewritten.
- I case of using VTNSS for some studies, various graphical interfaces can be used so that VTNNS will automatically generate an image of the toplogy that it is being simulated and also an animation which gives a clear view on how the neurons are behaving inside the simulated network.

Part III

Appendix

Appendix A

Modelling on variables for Hodgkin Huxley

Parameters of the equation in the model are normalised so that the resting potential of the nureon is 0mV .

In the hodgkin-huxley model, the gating variables of the neurons are assumed to satisfy the following differential equation:

$$x' = \frac{x_\infty(V) - x}{\tau_x(V)}$$

where $x \in \{n, m, h\}$ and all the functions of V in the equation are found from experimental data as follows.

$$\tau_x(V) = \frac{1}{\alpha_x(V) + \beta_x(V)} \qquad x_\infty(V) = \alpha_x(V)\tau_x(V)$$

and

$$\alpha_m(V) = \frac{2.5 - 0.1V}{exp(2.5 - 0.1V) - 1} \quad \alpha_n(V) = \frac{0.1-0.01V}{exp(1-0.1V)-1} \quad \alpha_h(V) = 0.07(exp(-\frac{V}{20})$$

$$\beta_m(V) = 4(exp(-\frac{V}{18}) \quad \beta_n(V) = 0.125(exp(-\frac{V}{80}) \quad \beta_h(V) = \frac{1}{exp(3 - 0.1V) + 1}$$

Fortran95 code for Hodgkin-Huxley Simulation

program hodgkin_huxley_simulator

! this program once compiled and executed ask for the input constant current and then simulate the voltage according to hodgkin huxley mode.

!should be run in a linux based system with a terminal and ffmpeg and gnuplot should be installed to get the animation. administrative previlages are assumed.

!the function written for gate variables can be modified and can be used as such also finding the limiting values of respective gating variables

!Euler's method is used for solving the equations, can be easily updated to RK4 for reducing the error

!a folder named animation1 will be created in the parent folder where plotted images will be saved by the name hodgkin_huxley.mp4

!output video will be created with name

implicit none

real::voltage,tau_m,m_infty,n_infty,tau_n,h_infty,tau_h, I_k,n_initial_k,n_final_k,step_s,a_n,b_n,fi

real::a_m,b_m,a_h,b_h,I_na,I_l,m_initial,m_final,h_initial,h_final,I_total,voltage_final,dV,x_begin,

real , parameter:: V_k=-12.14,V_Na=115.0,V_l=10.6,G_Na=120,G_K=36,G_l=0.03,c_m=1.0

integer:: i,j,k,mn,mm

```
write(*,*) "please write the applied current in the real format:"
read(*,*) I_m !reading the value of external current while running the program
voltage=0.0
n_initial_k= alpha_n(voltage)/(alpha_n(voltage)+beta_n(voltage))
m_initial= alpha_m(voltage)/(alpha_m(voltage)+beta_m(voltage))
h_initial= alpha_h(voltage)/(alpha_h(voltage)+beta_h(voltage))
n_final_k=0.0
m_final=0.0
h_final=0.0
initial_pp=.03
final_pp=50.0
step_s=(final_pp-initial_pp)/1200
open(11,file="hodgkin huxley with constant current")
call execute_command_line('mkdir "animation1"')
mn=0
x_begin=0.0
x_close=50.0
open(211,file="gating_variable.dat")
do i=1,1200
mm= mod(i,12)
if(i==1 .or. mm==0 ) then
write(11,*) initial_pp,voltage
open(12,file='gnucommand')
write(12,*) 'set terminal png'
write(12,*) 'set grid'
write(12,*) 'set xlabel "time in ms"'
write(12,*) 'set ylabel "voltage in mV"'
write(12,*) "set title 'Hodgkin huxley model with constant applied current'"
if (mn<10) then
write(12,112) 'set output "animation1/',0,0,mn,'.png"'
112 format(A23,i1,i1,i1,A5)
else if(mn<100 .and. mn>=10) then
write(12,113) 'set output "animation1/',0,mn,'.png"'
113 format(A23,i1,i2,A5)
else
write(12,114) 'set output "animation1/',mn,'.png"'
114 format(A23,i3,A5)
end if
write(12,*) 'set yrange [-20:115]'
write(12,*) 'set xrange [',x_begin,':',x_close,']'
write(12,*) 'plot "hodgkin huxley with constant current" w lp'
write(12,*) 'replot'
write(12,*) 'set output'
write(12,*) 'exit'
close(12)
```

```
call execute_command_line('gnuplot "gnucommand"')
mn=mn+1
end if
I_l = G_l*(voltage-V_l)
final_pp=initial_pp+step_s
a_n= alpha_n(voltage)
b_n= beta_n(voltage)
a_m= alpha_m(voltage)
b_m= beta_m(voltage)
a_h= alpha_h(voltage)
b_h= beta_h(voltage)
n_final_k = n_initial_k+ step_s*(potassium_open(initial_pp,n_initial_k,a_n,b_n))
m_final = m_initial+ step_s*(sodium_open(initial_pp,m_initial,a_m,b_m))
h_final = h_initial+ step_s*(sodium_close(initial_pp,h_initial,a_h,b_h))
i_na= G_Na*(m_initial**3)*(h_initial)*(voltage-V_na) !sodium current
i_k=G_K*(n_initial_k**4)*(voltage-V_k) !potassium current
voltage_final=voltage + step_s*(voltage_function(I_m,I_k,I_na,I_l))
!
!
write(211,*) initial_pp,m_initial,h_initial,n_initial_k
m_initial=m_final
h_initial=h_final
n_initial_k=n_final_k
voltage=voltage_final
initial_pp=final_pp
end do
close(11)
close(211)
call execute_command_line('ffmpeg -framerate 25 -i animation1/%03d.png
-c:v libx264 -r 30 -pix_fmt yuv420p hodgkin_huxley.mp4')
call execute_command_line('rm -f "gnucommand"')
call execute_command_line("rm -f 'hodgkin huxley with constant current'")
contains
function alpha_m(vol)
real:: alpha_m, vol
alpha_m=(2.5-0.1*vol)/(exp(2.5-0.1*vol)-1)
end function alpha_m
function alpha_n(vol)
real:: alpha_n, vol
alpha_n=(0.1-0.01*vol)/(exp(1.0-0.1*vol)-1)
end function alpha_n
function beta_h(vol)
real:: beta_h, vol
beta_h=1.0/(1.0+exp(3-0.1*vol))
end function beta_h
function alpha_h(vol)
```

```
real:: alpha_h, vol
alpha_h=0.07*exp(-Vol/20.0)
end function alpha_h
function beta_m(vol)
real:: beta_m, vol
beta_m=4.0*exp(-vol/18)
end function beta_m
function beta_n(vol)
real:: beta_n, vol
beta_n=0.125*exp(Vol/80)
end function beta_n
function potassium_open(x,y,a_n,b_n) !differential function for gate variable
real:: potassium_open,x,y,a_n,b_n,volt
potassium_open= a_n*(1-y) - b_n*y
end function potassium_open
function sodium_open(x,y,a_m,b_m) !differential function for gate variable
real:: sodium_open,x,y,a_m,b_m,volt
sodium_open= a_m*(1-y) - b_m*y
end function sodium_open
function sodium_close(x,y,a_h,b_h) !differential function for gate variable
real:: sodium_close,x,y,a_h,b_h,volt
sodium_close= a_h*(1-y) - b_h*y
end function sodium_close
function voltage_function(I_m,I_k,I_na,I_l) ! gives the function corresponding to dV/dt
real:: I_m,I_k,I_na,I_l,voltage_function
voltage_function= (I_m-(I_k+I_na+I_l))/c_m
end function voltage_function
end program hodgkin_huxley_simulator
```

Appendix B

Fortran95 code for simulation of coupled neurons

```
program lifmodel
use plotting_module
!coupling of two neurons will be simulated and graphed using rk4 and gnuplot resp.
!a normalised form as given in collin thesis is being employed so that the V_reset is zero and V_threshold=1.0 and some other normalisations are made ,
!please refer the report or collin thesis.
implicit none
real , parameter :: g_c=0.2
```

```
real :: v_1,v_2,beta ,delta,v_threshold,v_reset,I_app,I_total,k1,k2,k3,k4,l1,l2,l3,l4
real:: time_initial,time_range,step_size,dummy1,dummy2,error1
integer::n1,n2,n3
beta = 0.2
v_1=0.59
v_2=0.0
I_total = 3.0
step_size = 0.001
error1=0.001
time_initial=15.0
time_range = 45.0
open(1,file="datapoints.dat")
do n2=0,14
write(1,*) n2,v_1,v_2,0.0
end do
n1 = int((time_range-time_initial)/step_size)
do n2= 1,n1
if(abs(v_1-1.0) <error1) Then
v_1=3.0
else if(abs(v_1-3.0) <0.2) then
v_1=0.0
end if
if(abs(v_2-1.0) <error1) then
v_2=3.0
else if(abs(v_2-3.0) <0.2) then
v_2=0.0
end if
write(1,*) time_initial, v_1,v_2,I_total
k1 = neuron1_voltage(v_1,v_2,I_total)
l1= neuron2_voltage(v_1,v_2,I_total)
k2 = neuron1_voltage(v_1+0.5*step_size*k1,v_2+0.5*step_size*l1,I_total)
l2 = neuron2_voltage(v_1+0.5*step_size*k1,v_2+0.5*step_size*l1,I_total)
k3 = neuron1_voltage(v_1+0.5*step_size*k2,v_2+0.5*step_size*l2,I_total)
l3 = neuron2_voltage(v_1+0.5*step_size*k2,v_2+0.5*step_size*l2,I_total)
k4 = neuron1_voltage(v_1+step_size*k3,v_2+step_size*l3,I_total)
l4 = neuron2_voltage(v_1+step_size*k3,v_2+step_size*l3,I_total)
v_1 = v_1 + (step_size/6)*(k1+ 2.0*k2 + 2*k3 + k4)
v_2 = v_2 + (step_size/6)*(l1+ 2.0*l2 + 2*l3 + l4)
time_initial= time_initial + step_size
end do
close(1)
call graph_video(0.0,40.0)
contains
function neuron1_voltage(v_1,v_2,I)
real:: neuron1_voltage,v_1,v_2,I
neuron1_voltage = -1.0*v_1 + I + g_c*(v_2 - v_1)
```

```
    end function neuron1_voltage
    function neuron2_voltage(v_1,v_2,I)
    real:: neuron2_voltage,v_1,v_2,I
    neuron2_voltage = -1.0*v_2 + I + g_c*(v_1 - v_2)
    end function neuron2_voltage
    end program lifmodel
```

Fortran95 code for plotting_module

```
module plotting_module
    implicit none
    contains
    subroutine graph_video(x_initial,x_final)
    !will generate a video of the graph into the folder name
    !file name should be datafile.dat
    real:: x_initial,x_final,dummy1,step_size,x_left,x_right
    integer:: i,j,k,l,frame_rate
    step_size= (x_final - x_initial)/100
    x_left = x_initial
    x_right=10.0
    call execute_command_line('mkdir "animation1"')
    !write(*,*) step_size
    do i=1,100
    open(12,file='gnucommand')
    write(12,*) 'set terminal png'
    write(12,*) 'set grid'
    if (i<10) then
    write(12,112) 'set output "animation1/',0,0,i,'.png"'
    112 format(A23,i1,i1,i1,A5)
    else if(i<100 .and. i>=10) then
    write(12,113) 'set output "animation1/',0,i,'.png"'
    113 format(A23,i1,i2,A5)
    else
    write(12,114) 'set output "animation1/',i,'.png"'
    114 format(A23,i3,A5)
    end if
    write(12,*) 'set ytics 1'
    write(12,*) 'set ylabel "voltage"'
    write(12,*) 'set xlabel "time"'
    write(12,*) 'set xrange [',x_left,':',x_right,']'
    write(12,*) 'set yrange [-0.5:4.0]'
    write(12,*) 'plot "datapoints.dat" using 1:2 w l title "v_1" , "" using 1:3 w
l title "v_2" &
    , "" using 1:4 w l title "I_applied"'
    !120 format(A5,F10.10,A1,F10.10,A21)
    write(12,*) 'replot'
```

```
write(12,*) 'set output'
write(12,*) 'exit'
close(12)
! write(*,*) dummy1
call execute_command_line('gnuplot "gnucommand"')
x_left = x_left + step_size
x_right = x_right + step_size
end do
call execute_command_line('ffmpeg -framerate 4 -i animation1/%03d.png
-c:v libx264 -r 30 -pix_fmt yuv420p out.mp4')
! call execute_command_line('rm -f "gnucommand"')
end subroutine graph_video
end module plotting_module
```

Appendix C

Solution for the coupled differential equation

Proof. We need to solve the system of equations :

$$\frac{dV}{dt} = -V + (E_s(t) - V)g \tag{3.4.1}$$

$$\tau_s \frac{dg}{dt} = -g \tag{3.4.2}$$

From equation (3.4.1) we write V as a function of g as

$$\frac{dV}{dg} = \tau_s(1 + \frac{1}{g})V - \tau_s E_s$$

And it follows that

$$\frac{d}{dg}(V(exp(-\tau_s(g + log\, g))) = -\tau_s E_s exp(-\tau_s(g + log\, g))$$

Now integrating between $g(0)$ and $g(t)$, we get

$$\frac{V(t)exp(-\tau_s g(t))}{g(t)^{-\tau_s}} - \frac{V(0)exp(-\tau_s g(0))}{g(0)^{\tau_s}} = -\tau_s E_s \int_{g(0)}^{g(t)} \frac{exp(-\tau_s g)}{g^{\tau_s}} dg$$

now by substituting $\tau_s g = h$ in the aboove equation will be

$$\begin{aligned}\frac{V(t)exp(-\tau_s g(t))}{g(t)^{-\tau_s}} - \frac{V(0)exp(-\tau_s g(0))}{g(0)^{\tau_s}} &= -\,\tau_s^{\tau_s} E_s \int_{\tau_s g(0)}^{\tau_s g(t)} \frac{exp(-h)}{h^{\tau_s}} dh \\ &= -\,\tau_s^{\tau_s} E_s(\gamma(1-\tau_s, \tau_s g(t)) - \gamma(1-\tau_s, \tau_s g(0)))\end{aligned}$$

where $\gamma(a.b) = \int_0^b exp(-t)t^{a-1}dt$ which is also called the incomplete gamma integral.

Also since g also follows exponential decay, $g(t) = g(0)e^{-t/\tau_s}$ we have,

$g(0)^{-\tau_s} exp(t-\tau_s g(0)e^{-t/\tau_s})V(t) = V(0)e^{-\tau_s g(0)} g(0)^{-\tau_s} - \tau_s^{\tau_s} E_s(\gamma(1-\tau_s, \tau_s g(t)) - \gamma(1-\tau_s, \tau_s g(0)))$

If we define $\rho(a,b) = e^b b^{-a}\gamma(a,b)$, we can write

$$\tau_s^{\tau_s} E_s(\gamma(1-\tau_s, \tau_s g(0)) = \tau_s E_s g(0)\rho(1-\tau_s, \tau_s g(0))e^{-\tau_s g(0)} g(0)^{-\tau_s}$$
$$\tau_s^{\tau_s} E_s(\gamma(1-\tau_s, \tau_s g(t)) = \tau_s E_s g(t)\rho(1-\tau_s, \tau_s g(t))g(0)^{\tau_s} exp(t-\tau_s g(0)e^{-t/\tau_s})$$

So we get

$$e^{t-\tau_s g(t)}(V(t)+\tau_s E_s g(t)\rho(1-\tau_s, \tau_s g(t))) = e^{t-\tau_s g(0)}(V(0)+\tau_s E_s g(0)\rho(1-\tau_s, \tau_s g(0)))$$

From which we get,

$$V(t) = -\rho(1-\tau_s, \tau_s g(t))\tau_s E_s g(t) + exp(\tau_s(g(t)-g(0))-t)(V(0)+\rho(1-\tau_s, \tau_s g(0))\tau_s E_s g(0)$$

□

Fortran code for VTNNS with time delay in synaptic conductance

```
program VTNNS_with_time_delay
    use numerical
    implicit none
    integer ,parameter:: num=50
    real(8) , parameter :: V_th=20.0 , V_reset=14.0
    real(8),parameter:: time_bound= 1.0
    real(8) , parameter :: tau=20.0, tau_s=5.0/tau
    real(8),parameter:: E_plus = 74.0, E_minus= -6.0
    real(8),parameter:: weight_plus=1.2,weight_plus_plus=1.6, weight_minus=-
1.19
    real(8),parameter:: adjuster=0.01
    !num is the number of neurons in neural network
    real(8):: neural_network(num,4),weight_matrix(num,num),firing_table(num)
    real(8):: random1,random2,dummy_zero,testing,testing_argument(4),h,dummy_time
    real(8):: next_fire_time,printing_array(num+1)
    integer:: l
    neural_network=0.0
    weight_matrix=0.0
    do l =1,num-1
    ! weight_matrix(l,l+1)=weight_plus
```

```
weight_matrix(1,l+1)=weight_plus
! weight_matrix(l+1,1)= weight_plus
end do
do l =1,num
call init_random_seed()
!assigning a value between -75 and -53 V(0) for all neurons
call random_number(random1)
neural_network(l,1)= 16.0- 6.0*random1
end do
neural_network(1,1)= 15.0
do l=1,num
call init_random_seed()
call random_number(random1)
call random_number(random2)
random1=0.3*random1
random2=random2*0.018
random2=0.0
neural_network(l,2)= random1+random2
neural_network(l,3)=(random1*E_plus+random2*E_minus)/(random1+random2)
end do
neural_network(1,2)=1.2
neural_network(1,3)=74.0
open (1,file="firing_table.dat")
call firing_time_updater(neural_network,firing_table,dummy_zero,diff_central)
testing_argument(1)=15.0
testing_argument(2)=1.2
testing_argument(3)=74.0
testing_argument(4)= 0.15488245509844328
next_fire_time=minval(firing_table)
do while(next_fire_time<time_bound)
printing_array=0.0
do l =1,num
if(firing_table(l)==next_fire_time) then
write(3,*) "passed with time",next_fire_time
write(1,*) 20.0* next_fire_time,l
printing_array(1)= next_fire_time
printing_array(l+1)=1.0
write(2,*) printing_array
call outgoing_updater(neural_network,l,next_fire_time)
call incoming_updater(neural_network,l,next_fire_time,weight_matrix)
end if
end do
call firing_time_updater(neural_network,firing_table,next_fire_time,diff_central)
write(4,*) firing_table
next_fire_time= minval(firing_table)
end do
```

```
close(1)
call execute_command_line('gnuplot "gnucommand"')
contains
!————————————— ————- ————-
!gives the value of g afer giving the initial g value and time
function g_function(g_0,time)
implicit none
real(8):: g_0,time,g_function
g_function = g_0* exp(-time/tau_s)
end function g_function
!—————— —————————————— ——————————-
!gives the exact solution for voltage given the time and initial conditions
function voltage_function(arg_array)
!arg_array = (V_0,g_0,E_s,time)
implicit none
!gamma_1,gamma_2 refers to different version of gamma integral in equa-
tion
!dummy_1 and dummy_2 refers to dummy var for the subroutine incog
real(8) ::arg_array(4), voltage_function,g_t,gamma_1,gamma_2,dummy_1,dummy_2,a,b,c
g_t = g_function(arg_array(2),arg_array(4))
a = 1-tau_s
b= tau_s*g_t
c= tau_s*arg_array(2)
call incog(a,b,gamma_1,dummy_1,dummy_2)
call incog(a,c,gamma_2,dummy_1,dummy_2)
gamma_1 = gamma_1*exp(b)*(b**(-a))
gamma_2 = gamma_2*exp(c)*(c**(-a))
voltage_function = (-tau_s*arg_array(3)*g_t*gamma_1) + exp(-arg_array(4)+
tau_s&
*(g_t-arg_array(2)))*(arg_array(1)+tau_s*arg_array(3)*arg_array(2)*gamma_2)
end function voltage_function
!—————— ——————————- —————————— ——————————
!special voltage function in which one of the gamma integral is given as an
argument
function voltage_function_special(arg_array)
! arg_array=(V_0,g_0,E_s,gamma,time)
implicit none
!gamma_1,gamma_2 refers to different version of gamma integral in equa-
tion
!dummy_1 and dummy_2 refers to dummy var for the subroutine incog
real(8) ::arg_array(5), voltage_function_special,V_0,g_0,E_s,time,g_t
real(8) :: gamma,gamma_1,gamma_2,dummy_1,dummy_2,a,b,c
g_t = g_function(arg_array(2),arg_array(5))
a = 1-tau_s
b= tau_s*g_t
! c= tau_s*g_0
```

```
call incog(a,b,gamma_1,dummy_1,dummy_2)
! call incog(a,c,gamma_2,dummy_1,dummy_2)
gamma_1 = gamma_1*exp(b)*(b**(-a))
gamma_2=arg_array(4)
!gamma_2 = gamma_2*exp(c)*(c**(-a))
voltage_function_special = (-tau_s*arg_array(3)*g_t*gamma_1) + exp(-
arg_array(5)+ tau_s&
*(g_t-arg_array(2)))*(arg_array(1)+tau_s*arg_array(3)*arg_array(2)*gamma_2)
end function voltage_function_special
!——- ————— ———— —————- ——-
!subroutine to update the state of a neuron after it is fired.
subroutine outgoing_updater(Neural_network,i,fire_time)
implicit none
!i refers to the index of the neuron which is going to be fired
integer :: i
real(8) :: Neural_network(num,4), fire_time
neural_network(i,1) = V_reset
neural_network(i,2)=neural_network(i,2) *&
exp((neural_network(i,4)-fire_time)/tau_s)
neural_network(i,4)= fire_time
end subroutine outgoing_updater
!——————- —————— —————————
!subroutine to update other neurons after one neuron fires.
subroutine incoming_updater(neural_network,i,fire_time,weight_matrix)
implicit none
integer:: j,i
real(8) :: Neural_network(num,4), fire_time,w_dummy,alpha,beta,weight_matrix(num,num)
real(8):: arg_array(4)
alpha= (E_plus - E_minus)/2.0
beta= (E_plus + E_minus)/2.0
do j = 1,num
if (j .ne. i) then
arg_array(1:3)=neural_network(j,1:3)
arg_array(4)=fire_time-neural_network(j,4)
neural_network(j,1) =voltage_function(arg_array)
neural_network(j,2)=neural_network(j,2) *&
exp((neural_network(j,4)-fire_time)/tau_s)
w_dummy = weight_matrix(i,j)
neural_network(j,3) = (neural_network(j,2)*neural_network(j,3) + alpha*w_dummy
+ beta*abs(w_dummy))/&
(neural_network(j,2)+abs(w_dummy))
neural_network(j,2)= neural_network(j,2) + abs(w_dummy)
neural_network(j,4)= fire_time
w_dummy=0.0
end if
end do
```

```
end subroutine incoming_updater
!———— ———— ———— —————-
!to update the spike firing time table
subroutine firing_time_updater(neural_network,firing_table,firing_time,diff_function)
implicit none
real(8),external:: diff_function
!real(8),external::voltage_function
!real(8) , external :: voltage_function_special
real(8):: neural_network(num,4),firing_table(num),firing_time
real(8)::dummy_voltage,dummy_time,guess, error,h,dummy_1,dummy_2,diff,arg_array(5),g_star
real(8):: dummy_a,dummy_b,dummy_c,dummy_gamma_1,dummy_gamma_2,dummer_1,dummer_2,a
real(8):: ar_arr(4),random_dummy
integer::k,q
do k=1,num
firing_table(k)=1000000.0
if(neural_network(k,3)>V_th) then
g_star= V_th/(neural_network(k,3)-v_th)
write(*,*) "g_star is", g_star,neural_network(k,2)
if(neural_network(k,2)>g_star) then
dummy_a = 1-tau_s
dummy_b= tau_s*g_star
dummy_c= tau_s*neural_network(k,2)
call incog(dummy_a,dummy_b,dummy_gamma_1,dummer_1,dummer_2)
call incog(dummy_a,dummy_c,dummy_gamma_2,dummer_1,dummer_2)
dummy_gamma_1 = dummy_gamma_1*exp(dummy_b)*(dummy_b**(-
dummy_a))
dummy_gamma_2 =dummy_gamma_2*exp(dummy_c)*(dummy_c**(-dummy_a))
dummy_voltage = (-tau_s*neural_network(k,3)*g_star*dummy_gamma_1)
&
+((g_star/neural_network(k,2))**tau_s)* exp( tau_s&
*(g_star-neural_network(k,2)))*(neural_network(k,1)&
+tau_s*neural_network(k,3)*neural_network(k,2)*dummy_gamma_2)
if(dummy_voltage>V_th) then
guess = 0.0
arg_array_2(1:3) = neural_network(k,1:3)
arg_array_2(4)=guess
error=voltage_function(arg_array_2)-V_th
q=0
do while(abs(error)>0.001)
if (q==20) then
q= q+1
arg_array(1:3)=neural_network(k,1:3)
arg_array(4)=dummy_gamma_2
arg_array(5)=guess
h=0.0000001
diff= diff_function(voltage_function_special,arg_array,5,h)
```

```
random_dummy= error/diff
if(random_dummy>0) then
if(neural_network(k,1)>V_th) then
guess= adjuster
exit
else
guess=100000.0
exit
end if
end if
guess = guess- error/diff
arg_array(5)= guess
error= voltage_function_special(arg_array)-V_th
end do
firing_table(k)=neural_network(k,4) + guess
end if
end if
end if
end do
end subroutine firing_time_updater
!———- ————— ———————— —————- ———
end program VTNNS_with_time_delay
```

Bibliography

[1] The Effects of Potassium Currents on the Synchronization of Electrically Coupled Neural Oscillators , Middleton ,2005

[2] Lecture presentations of Dr.Caleb Kemere, Rice university

[3] Exact simulation of Integrate and Fire models with synaptic conductances,Brette,2006

[4] Lecture notes on Hodgkin Huxley model by Dr.Stephen coombes, university of Nottingham