Variable Topology Neural Network Simulator

Manu Jayadharan,
MS student in Mathematics,
IISER Mohali

Abstract

The complexity of neural networks can be ascribed to their topology and nature of interconnections rather than specialization of neurons. So understanding how the topology of neural networks affect signal propagation across the network plays a key role in developing an insight into how similar action potentials can be used to propagate complex information from and to the brain. In this work, we make an attempt to lay down a computationally efficient algorithm to make a simplified Variable Topology Neural Network Simulator(VTNNS) that can simulate or mimic the behavior of signal propagation in a neural network with any specified topology and synaptic relation. The simulator is based on the exact solution of mathematical formulation of a network based on LIF model (see [1]) which gives it high computational efficiency. The model could incorporate various kind of synaptic interactions. Choice of LIF model is justified by the existence of closed form solution and ease of implementation and inference. Never the less, more realistic models can be used for modeling VTNNS. A more complex version of this prototype could possibly be used to simulate biological phenomenons like the effect of drug interaction in the central nervous system and other practical applications which calls for signal tracking within a biological neural network.

1 Mathematical Model

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

$$\tau \frac{dV}{dt} = -(V - V_0) \tag{1}$$

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

$$\tau \frac{dV}{dt} = -(V - V_0) - g^{+}(t)(V - E^{+}) - g^{-}(t)(V - E^{-}) \tag{2}$$

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

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

$$\tau_s \frac{dg^i}{dt} = -g^i \tag{3}$$

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 conductance, Brette, 2006] which comes in handy for simulation of large network of neurons.

We assume that both the excitatory and inhibitory conductance has same time constant τ_s, this is a trade-off we make to get an exact solution of the equation (2) which can then be used to develop the simulator.

In equation (2), we assume $V_0 = 0$ and express the time constant in units of τ to get :

$$\frac{dV}{dt} = -V + (g^+(t) + g^-(t))(E_s(t) - V) \tag{4}$$

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) and (4), we substitute $g^+ + g^- = g$ to get:

$$\frac{dV}{dt} = -V + (E_s(t) - V)g \tag{5}$$

$$\tau_s \frac{dg}{dt} = -g \tag{6}$$

equation (5) can be solved to get

$$\begin{aligned} V(t) &= -\rho(1-\tau_s, \tau_s g(t))\tau_s E_s g(t) \\ &\quad + exp(\tau_s(g(t) - g(0)) - t)(V(0) \\ &\quad + \rho(1-\tau_s, \tau_s g(0))\tau_s E_s g(0) \end{aligned} \tag{7}$$

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 incomplete gamma integral.

Complete derivation of the solution is given is given in Appendix-A .

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

2 Designing the Simulator

Here we explore a proper algorithm for the simulation of the spiking and its propagation 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.

1. 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

2. 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.

3. 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 should be defined to find the next firing time of each neuron, which can be finite or ∞in the case of no upcoming spike.

4. 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(excitatory) or negative(inhibitory).

VTNNS is modeled 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.

2.1 VTNNS with time delay

Algorithm 1 with 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 neuron 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 propagation 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.

2.2 VTNNS without time delay

Algorithm 2 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
```

2.3 Finding the next firing time of a neuron in the network

For each neuron, next firing time is defined as the time at which the voltage V reaches . From equation (7), it can be seen that V first increases and then decreases, so initial 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 convergence .

But in most cases, next firing time will be ∞ which means the neuron never spikes. To save the computational 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 (5) and (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:

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

$$g_{min} = \frac{1}{(E_s/V_t) - 1} \tag{8}$$

Since conductance g decreases with time, there is possibility 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 (**??**) to calculate $V(t)$ for t such that $g(t) = g_{min}$.

$$\begin{aligned} V(g_{min}) &= -\rho(1 - \tau_s, \tau_s g_{min})\tau_s E_s g_{min} \\ &\quad + (\frac{g_{min}}{g_0})^{\tau_s} exp(\tau_s(g_{min} - g_0) - t) \\ &\quad (V_0 + \rho(1 - \tau_s, \tau_s g_0)\tau_s E_s g_0 \end{aligned} \tag{9}$$

Algorithm 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 Coding for VTNNS

Simulator with and without time lag in synaptic conductance is coded separately . 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 propagation of the signal in the network, the second in which one of the neurons in the network act as an oscillator(source of signal) with a specified frequency. The frequency of the latter can also be modeled as a random process following some distribution like Poisson distribution. Both the setups can be used depending on which specific bio-physiological situation of a neural network that we want to simulate.

All codes are written in $Fortran95$ considering the efficiency of the same to handle huge arrays and computational 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 simulator. 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-B.

4 Results(or Simulation of different topology of Neural networks)

To demonstrate the functioning of VTNNS, neural networks of some common simple typologies 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 $[10, 16]$.

4.1 Star Topology

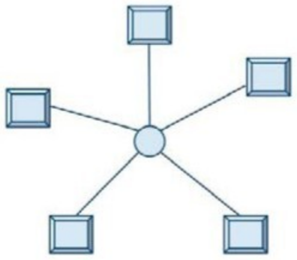

Figure 1: star topology

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

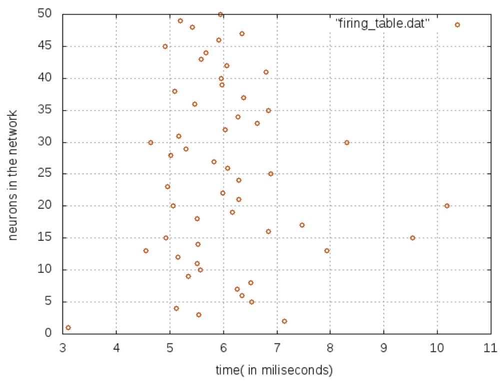

Figure 2: Raster plot for neural network with star topology simulated using VTNNS without time delay in propagation of spike, $w_{1j} = 1.2$.

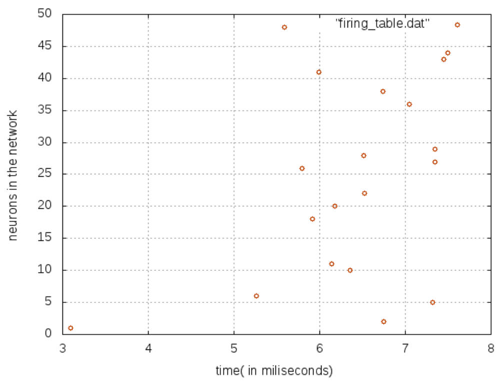

Figure 3: Raster plot for neural network with star topology simulated using VTNNS with time delay in propagation of spike, $w_{1j} = 1.2$

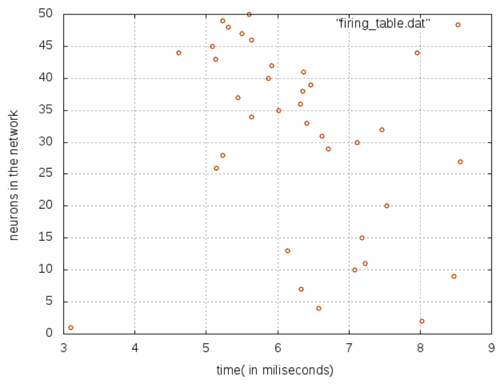

Figure 4: Raster plot for neural network with star topology simulated using VTNNS with time delay in propagation of spike, $w_{1j} = 1.2$ for neurons $1 - 24$ and $w_{1j} = 1.6$ for neurons $25 - 50$.

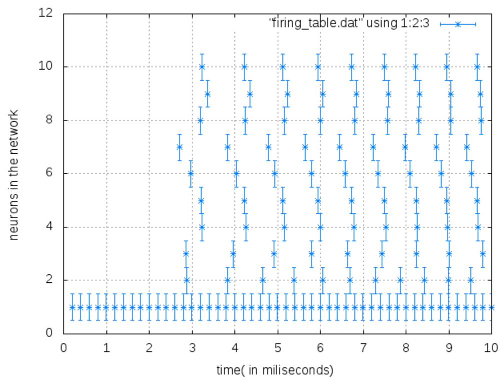

Figure 5: Raster plot for simulation in which neuron oscillates with time period $0.2ms$

In *Figure 2* and *Figure 3*, once the first neuron fires, some of the other neurons fire in a scattered manner as expected. There is a delay in the firing pattern of neurons in *Figure 3* due to the delay in the propagation of spike. *In Figure 4*, w_{1j} is 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. *Figure 5* shows the simulation in which neuron 1oscillates with time period $0.2ms$, no kind of synchronizing behavior is observed even with the change of the oscillation frequency with and without delay in transmission of spike.

4.2 Fully connected Topology

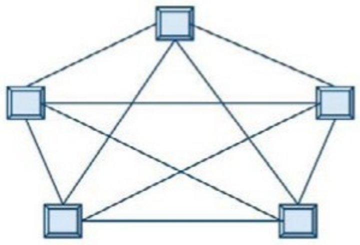

Figure 6: fully connected topology

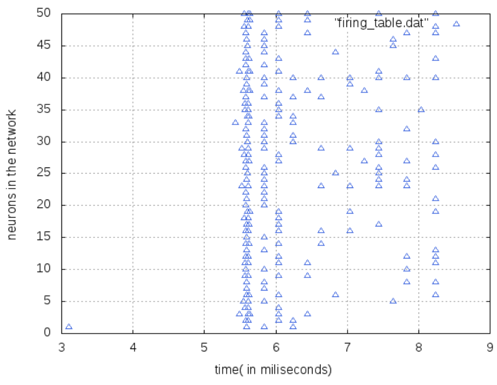

Figure 7: Raster plot for neural network with fully connected topology simulated using VTNNS without time delay in propagation of spike,$w_{ij} = 1.2$(each pair of neurons is connected to each other using an excitatory synapse).

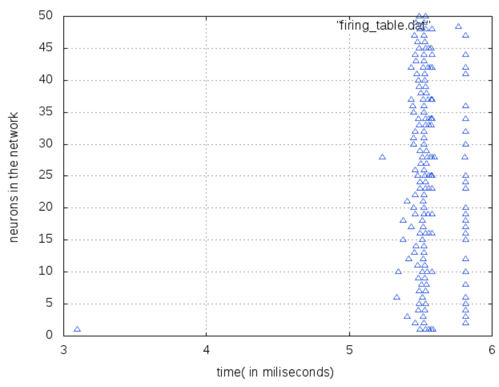

Figure 8: Raster plot for neural network with fully connected topology simulated using VTNNS with time delay in propagation of spike,$w_{ij} = 1.2$.

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 excitatory 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 propagation of spike.

4.3 Ring topology

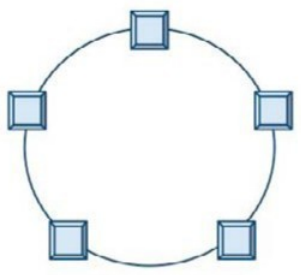

Figure 9: ring topology

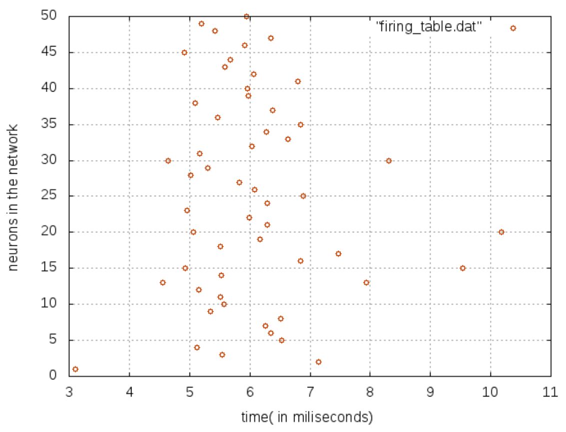

Figure 10: Raster plot for neural network with ring topology simulated using VTNNS without time delay in propagation of spike.$w_{i,i+1} = 1.0$ for $i = 1,..,9$ and $w_{10,1} = 1.0$(each neuron is connected to the next neuron in the order in a circle with an excitatory sypansis).

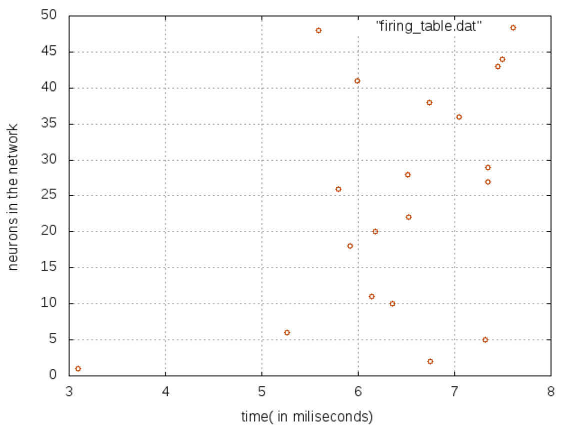

Figure 11: Raster plot for neural network with ring topology simulated using VTNNS with time delay in propagation of spike.$w_{i,i+1} = 1.0$ for $i = 1,..,9$ and $w_{10,1} = 1.0$.

In both cases a triangular shape is obtained by the plot. Very close synchronizing behavior is also visible in both the cases.

References

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

Appendix A

Solution for the coupled differential equation

Proof.

We need to solve the system of equations :

$$\frac{dV}{dt} = -V + (E_s(t) - V)g \tag{10}$$

$$\tau_s \frac{dg}{dt} = -g \tag{11}$$

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

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

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,

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

Appendix B

Fortran code for VTNNS with time delay in synaptic conductance

```fortran
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 equation
!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 equation
!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,arg_array_2(4)
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.
```