Spiking Neurons

Computer Exercises using the NEURON Simulator

1st Baltic-Nordic Summer School on Neuroinformatics

May 2013, Kaunas, Lithuania.

Author: Dr Bruce Graham, Computing Science & Mathematics, University of Stirling, U.K.

URL: www.cs.stir.ac.uk/~bpg/

Email: b.graham@cs.stir.ac.uk

Computer Exercises using NEURON

Table of Contents

Overview

These exercises make use of the **NEURON** simulator (www.neuron.yale.edu). It is not necessary to be familiar with NEURON before attempting these exercises, but previous experience will allow you to progress faster and to try some of the optional, advanced exercises. Prewritten code is available for all of the main exercises, but it is also suggested that you try to write your own **hoc** code for some of them, rather than use the supplied code. For this you will need some familiarity with **hoc**, and a simple text editor eg Notepad. The NEURON documentation available via your NEURON installation will be very useful here.

A number of the exercises are based on examples from the book, "**Principles of Computational Modelling in Neuroscience**" by Sterratt, Graham, Gillies and Willshaw (CUP, 2011). The book will be referred to as **PCNM** in the following text.

Code files

The following NEURON code files can be downloaded from the **computer exercises web page: simpcell.hoc, FIcurve.hoc, EIosc.hoc, stein.hoc, amit-brunel.hoc, STDPtiming.hoc, STDPnetseq.hoc, stdpB.mod, AAM-SW.hoc.**

Code for several figures from PCNM is provided in the form of **nrnzip** files that can be downloaded from the web page and run simply by double-clicking on them: **fig-5-9, fig-8-6, fig-9-8, fig-9-10.**

Other tutorial material

If you would like more experience with NEURON before trying these exercises, then the following tutorial material is available.

NEURON GUI tools: Neuron models and small networks can be constructed in NEURON using in-built GUI tools, with no programming required. Links to relevant tutorials on these tools, available on the NEURON website, are provided on the computer exercises web page.

Learning Hoc: To gain basic experience in constructing NEURON simulations by programming them in the scripting language "hoc", try the series of tutorials by Gillies and Sterratt. The tutorials are available in PDF format on the exercises web page, and all associated hoc code is also provided.

1. Frequency-Input Current (F-I) Firing Curve of a Neuron

A simple compartmental model neuron is stimulated by a constant current injection to the cell body. The cell body is a cylinder 30um in diameter and 30um long, and contains sodium, potassium and leak ion channels, and so can produce action potentials if stimulated to threshold. A 1000um long and 2um thick dendrite is attached to the cell body and is modelled as 11 compartments, each of which contains only leak ion channels.

Neuron construction:

You can either choose to use the predefined network simulation available in **FIcurve.hoc**, or you can try to construct it yourself by writing your own hoc file.

If you are constructing your own neuron, then your code should construct a cell of type **SimpCell** using the **SimpCell** template (in simpcell.hoc) . Add an IClamp point process to the soma. Give it a delay of 0 msecs, duration of 100 msecs and an amplitude of 0.1 nA.

The exercise is to determine the firing rate response of this neuron to different magnitudes of current injection (which approximates a constant stream of EPSPs due to input from other neurons, which are not modelled). This is the **F-I curve** of our neuron.

Do the following:

1. Double-click on **FIcurve.hoc** or your own **hoc** file to run the simulation in NEURON.
2. Bring up a PointProcessManager window (via Tools->PointProcesses->Managers), click SelectPointProcess and choose an APcount process. Then click Show and select Parameters. This point process will count the number of action potentials the neuron fires.
3. Open up the IClamp in a window (via Tools->PointProcesses->Viewers) and set the amp to 0.1 nA (if necessary).
4. Create a voltage graph for the soma (via Graph->Voltage axis).
5. Open a RunControl window (Tools->RunControl) and set Tstop to 100 msecs.
6. Press Init & Run – this will run a simulation of stimulating the cell with the IClamp electrical current for 100msecs. The membrane voltage in the cell body over this time is plotted in the voltage Graph window.
7. Note the action potential count (n) in the PointProcessManager window (it may be zero!).
8. Increase the IClamp current amplitude (amp) by a small amount eg 0.1 nA, and repeat steps 6 and 7. Keep incrementing amp by, say 0.1 each time, then by larger amounts when you get bored! There is no need to go beyond an amp of 3.0.
9. Draw a sketch of amp (x-axis) versus n (y-axis).
10. **ALTERNATIVE** to steps 6-9 is to create a Grapher window in your GUI (Graph->Grapher) with the following settings:
    a. Indep Begin: 0; Indep End: 3; Steps 30
    b. Independent Var: IClamp[0].amp
    c. X-expr: IClamp[0].amp
    d. Generator: run()
    e. In the plot window, right-click and set “Plot what?” to APCount[0].n

    Now you just need to press Plot in the Grapher window to run a number of simulations and automatically plot the complete F-I curve! You can try altering the number of current steps (Steps) and the length of simulation (Tstop in RunControl) to get a smoother curve.

Dr B. Graham, Computing Science & Maths, University of Stirling, U.K.

2. Type I and Type II F-I Curves

The Frequency-Current (F-I) firing curve of a neuron is classified as either Type I, if the frequency of firing increases smoothly from zero, once the firing threshold is reached, or as Type II, if the neuron suddenly starts firing with a finite frequency after the threshold.

Question 1: What is the type of the F-I curve of the neuron in exercise 1? (Looking at the figure below will help with answering this).

In this exercise we will see that the presence or absence of particular ion channels in the membrane can determine the type of the F-I curve of a neuron. The code and GUI are already provided for you using the example code from the PCMN book.

Running the book code:

Double-click on **fig-5-9.nrnzip** to run the code associated with figure 5.9 in PCNM. This demonstrates Type I and Type II firing in a neuron, controlled by the presence or absence of the potassium A-type current.

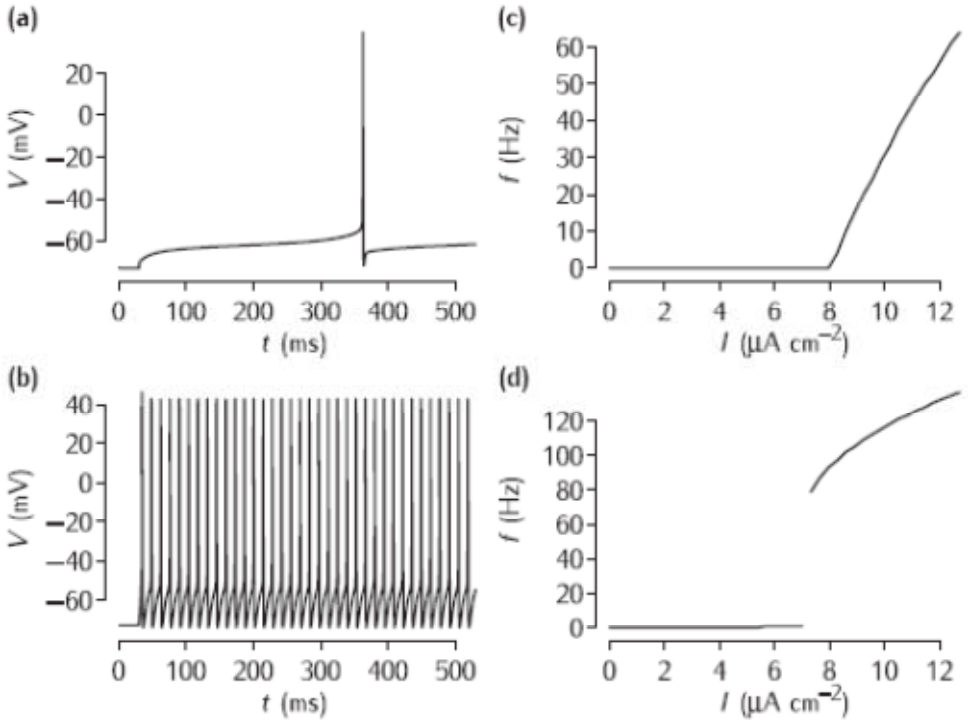


Figure 5.9: (a) First spike in a Type I neuron. (b) Start of spiking in a Type II neuron. (c) f-I curve for Type I. (d) f-I curve for Type II.

Once the simulation file is open, these are the steps to reproduce each panel in the figure:

Panel (a)

1. In the top **Parameters** window, click on the checkbox next to **Type I parameters**
2. Click on **Set Type I threshold current**
3. Click on **Init & Run** in the **Run Control** window
4. The voltage plot corresponding to Panel (a) above is in Graph[0] in the GUI. The plot windows below this graph show sodium, potassium and leak conductances and current.

Panel (b)

1. In the top **Parameters** window, click on the checkbox next to **Type II parameters**
2. Click on **Set Type II threshold current**
3. Click on **Init & Run** in the **Run Control** window
4. The voltage plot should now look like Panel (b)

Panel (c)

1. In the top **Parameters** window, click on the checkbox next to **Type I parameters**
2. Click on **Plot** in the **Grapher** window
3. Simulations for a number of levels of current injection will be run and the firing frequency will be plotted against current in the **Grapher** window.
4. Once the simulations have run, to make sure you can see these results, right-click in this graph area and select **View... -> View = Plot**

Panel (d)

1. In the top **Parameters** window, click on the checkbox next to **Type II parameters**
2. Click on **Plot** in the **Grapher** window
3. Simulations for a number of levels of current injection will be run and again the firing frequency will be plotted against current in the **Grapher** window.
4. Once the simulations have run, to make sure you can see these results, right-click in this graph area and select **View... -> View = Plot**

Question 2: What cell parameters are different between the Type I and Type II examples?

Supplementary Question 2 (for later research): Can you explain why the difference in firing characteristics arises?

3. Simple Excitation-Inhibition (E-I) Oscillator

A network of two neurons forming an E-I oscillator is modelled.

Each neuron (defined in **simpcell.hoc**) contains a cell body that is a cylinder 30um in diameter and 30um long, and contains sodium, potassium and leak ion channels, and so can produce action potentials if stimulated to threshold. A 1000um long and 2um thick dendrite is attached to the cell body and is modelled as 11 compartments, each of which contains only leak ion channels.

The excitatory cell connects to a synapse on the dendrite of the inhibitory cell, and the inhibitory cell connects back to a synapse on the dendrite of the excitatory cell. There is a connection delay between each cell. The excitatory cell also is stimulated by a constant current injection to the cell body.

Network construction:

You can either choose to use the predefined network simulation available in **EIosc.hoc**, or you can try to construct it yourself by writing your own hoc file.

If you are constructing your own network, then your code should construct two cells of type **SimpCell** and connect them together as described above. The **SimpCell** template (in simpcell.hoc) contains suitable synapses to use and a procedure, **connect2target(..)** that can be called to construct network connections. Both connections should have a delay of 2 msecs; the excitatory connection should have a weight of 0.02 and the inhibitory connection a weight of 0.01. Create an **IClamp** current injection object situated in the soma of your excitatory cell. Give it a delay of 0 msecs, duration of 1000 msecs and an amplitude of 0.8 nA. It will also be useful to create a GUI **xpanel()** that allows you to change the connection weights and delays (have a look in **EIosc.hoc** if you need help with this).

The exercise is to determine the firing patterns of these two cells for different current injection amplitudes to the excitatory cell, different connection weights (excitatory and inhibitory) and different connection delays.

Do the following:

1. Double-click on **EIosc.hoc** or your own hoc file to run the simulation in NEURON.
2. Open up the IClamp in a window (via Tools->PointProcesses->Viewers) and make sure amp is set to 0.8 nA.
3. Create two separate voltage graphs, one for the excitatory cell soma and one for the inhibitory cell soma (Graph->Voltage axis; right-click on plot area and select Plot what?, then Show->Object refs to select the cell somas; use the right-click Delete option to remove the v(.5) plot from your inhibitory cell graph).
4. Open a RunControl window and set Tstop to 1000. Press Init & Run – this will run a simulation of stimulating the excitatory cell with the IClamp electrical current for 1000msecs.
5. For this network configuration you should observe a characteristic oscillating firing pattern in both cells. Set the E->I weight to 0 and rerun to see what happens when the cells are not connected. Now reset the E->I weight to 0.02 before continuing to step 6.

6. This network behaviour is actually very sensitive to the current injection amplitude to the excitatory cell. Try changing amp in the IClamp window by small amounts, keeping within a range of between 0.6 and 1.0 nA. *Note any different firing patterns that emerge.*
7. Now reset the IClamp amp to 0.8. Try changing the E->I weight in the Weight window in a range between 0.01 and 0.08. *Again, note any different firing patterns.*
8. Reset the E->I weight to 0.02. Try changing the I->E weight in a range from 0.004 to 0.02. *Note what happens now.*
9. Reset the I->E weight to 0.005. Try changing the E-> delay between 0 to 4 msecs and *note what happens.*

Question 3: What is the behaviour of this circuit and how do the various parameters (weights, delays etc) affect this behaviour?

4. Excitation-Inhibition Balance in an I&F Neuron

This exercise runs the code behind figure 8.6 of PCNM that explores the firing characteristics of a single integrate-and-fire neuron being driven by a number of excitatory and inhibitory inputs. This was originally explored by Stein (*Biophysical Journal* 5:173-194, 1965). With a large, balanced number of excitatory and inhibitory inputs (NE=300, NI=150) the cell fires irregularly. With small excitation only (NE=18, NI=0) the cell fires at the same mean rate, but much more regularly.

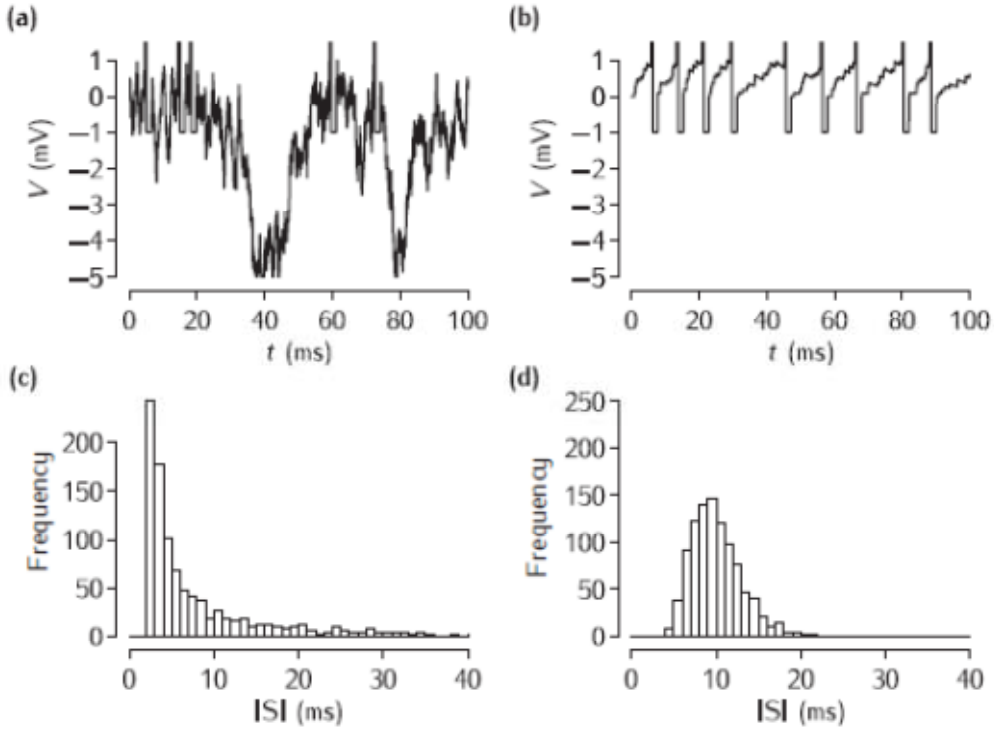


Figure 8.6 (a, c) Large, balanced excitation and inhibition (NE=300, NI=150). (b, d) Small excitation (NE=18, NI=0)

Running the book code:

Double-click on **fig-8-6.nrnzip** to run the code associated with figure 8.6 in the book.

To reproduce the data behind panel **(a)** and **(c)**:

1. In the **RunControl** window click on **Init&Run**
2. A trace of the membrane potential appears in the top window and at the end of the simulation the histogram appears in the lower window.

To reproduce the data behind panel **(b)** and **(d):**

1. In the **Parameters** window change **NE** to 18 and change **NI** to 0.
2. In the **RunControl** window click on **Init&Run**.

Do the following:

Set the parameters to the large, balanced condition (NE=300, NI=150) and do the following:

1. Explore the effect of changing the strength of the excitatory and inhibitory inputs by changing **JE** and **JI** in the **Parameters** window. Make small changes to one or other input

strength and note what effect it has on the regularity of firing. For each value of JE that you try, find a value of JI that results in (a) irregular firing, and a value of JI that results in (b) regular firing.

2. Return the strengths to their original values (JE=0.1, JI=-0.2). Now, in the same way, explore changing the number of excitatory and inhibitory inputs.

Question 4: How sensitive is irregular firing to the balance between excitation and inhibition?

ADVANCED: Examining the NEURON code

The code file corresponding to this simulation is **stein.hoc**. You can load **stein.hoc** into your favourite text editor to examine the code behind these simulations. This code demonstrates constructing I&F neurons as point processes (**IntFire1** in this case), and the use of **NetStim** objects to provide noisy spike trains as inputs. The strange part of this is that the I&F cell has to be inserted into a "dummy" compartmental neuron, but this "dummy" neuron takes no other part in the simulation. This is because NEURON was developed around the concept of simulating compartmental model neurons, and I&F point neurons have been added later to the simulator.

There is other interesting and useful code in this example, showing the use of vectors to collect spike times and calculate statistics, such as the distribution of interspike intervals. You should look up the NEURON documentation to find out more about the different components of this simulation.

Advanced Exercise: You might like to explore the different I&F neuron models available with NEURON. The model used (**IntFire1**) contains only a single time constant for the membrane. The other two models available (**IntFire2** and **IntFire4**) introduce further time constants for synaptic inputs. As a further exercise you could try altering the code in **stein.hoc** to use either of these I&F models instead, and explore the effect of synaptic time constants on the cell's firing characteristics. You can run **stein.hoc** directly by double-clicking on it. You will need to create a suitable graph to plot the I&F output eg. Plot **ifn.M** on a **state** graph.

5. Excitation-Inhibition Balance in a Network of I&F Neurons

This exercise runs the code behind figure 9.8 of PCNM that explores the firing characteristics of a network of integrate-and-fire neurons, based on the work of Amit & Brunel (*Network: Computation in Neural Systems* 8:373-404, 1997). The network contains randomly connected populations of excitatory and inhibitory neurons, which also receive external excitatory inputs. In an infinitely large network like this, with balanced excitation and inhibition, the cells should fire essentially randomly and uncorrelated with each other. In finite sized networks, as in the simulation, episodes of strong correlations in firing between groups of neurons appear. These become more evident the smaller the network.

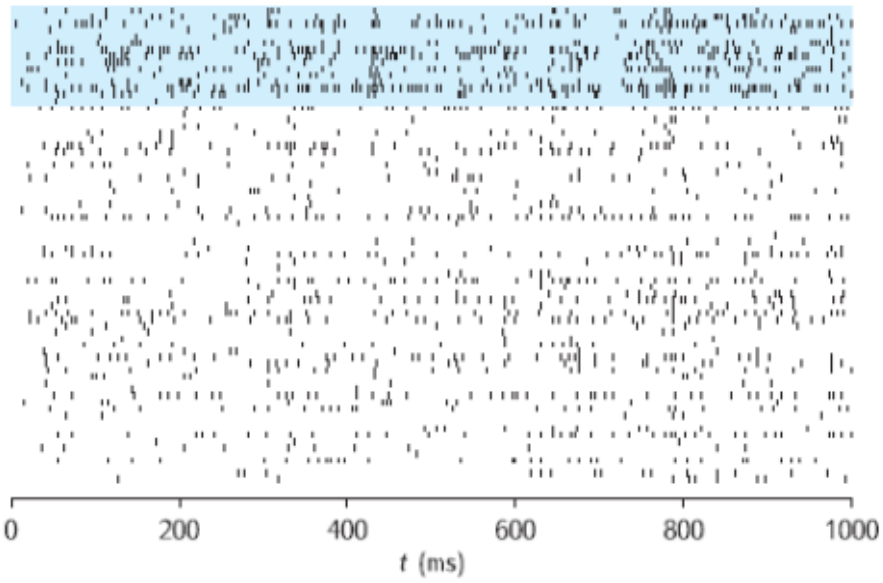


Figure 9.8 Raster plot of activity from 15 inhibitory (shaded at top) and 60 excitatory neurons exhibiting largely irregular firing.

Running the book code:

Double-click on **fig-9-8.nrnzip** to run the code associated with figure 9.8 in the book. This simulation is set up to run a network of half the size (scale=0.5) of the network used to produce the results in the figure. It will still take several minutes to set up and run on a decent PC.

Do the following:

Try several smaller networks i.e. reduce the scale parameter below 0.5.

Question 5: How do the network firing characteristics change as the size of the network is reduced?

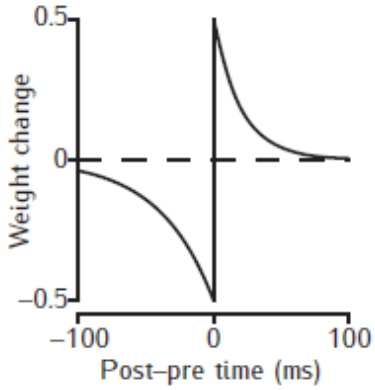
Examining the code:

You can load **amit-brunel.hoc** into your text editor to examine the code behind these simulations. This code provides an example of setting up a randomly connected network of neurons. Note that this file is not runnable by itself. You need to run **fig-9-8.nrnzip** to generate a simulation.

6. STDP in Action

There are two exercises to illustrate the effects of spike-time-dependent plasticity (STDP). They make use of a simple, saturating version of STDP implemented in **NMODL** in the file **stdpB.mod** (equation 7.37 in PCNM with leading terms fixed at e_i=e_j=1):

$$\frac{dw_{ij}}{dt} = \epsilon_i \epsilon_j \left[(w^{\mathrm{LTP}} - w_{ij}) \sum_k A^{\mathrm{LTP}} \exp(-(t - \tilde{t}^{\mathrm{pre}})/\tau^{\mathrm{LTP}}) \delta(t - t_k^{\mathrm{post}}) \right.$$
$$\left. - (w_{ij} - w^{\mathrm{LTD}}) \sum_l A^{\mathrm{LTD}} \exp(-(t - \tilde{t}^{\mathrm{post}})/\tau^{\mathrm{LTD}}) \delta(t - t_l^{\mathrm{pre}}) \right].$$



This figure shows schematically the magnitude of the weight change as a function of the time the postsynaptic spike follows a presynaptic input. LTP takes place if the postsynaptic spike occurs after the presynaptic spike, with the magnitude of the weight increase decaying with a time constant of 17 ms. LTD occurs when the presynaptic spike follows the postsynaptic spike, decaying with a time constant of 34 ms.

Copy the files **STDPtiming.hoc**, **STDPnetseq.hoc, simpcell.hoc** and **stdpB.mod** to a folder called e.g. STDP and run **mknrndll** from the NEURON start menu on this folder to create a new version of NEURON that incorporates stdpB.mod. Now the two STDP **hoc** files can be run by double-clicking on them.

6a. Phase precession of spike timing (STDPtiming.hoc)

In this exercise, a single neuron receives excitatory input from 10 spike trains through synapses that can undergo STDP. Each spike train consists of regular spikes spaced at 30 msec intervals. However, each spike train is offset by increments of 1 msec, so that the first spike in the 10th spike train occurs 10 msecs after the first spike in the first train. The individual EPSPs from the different spike trains summate, so that initially the cell fires its own action potential near the end of the sequence of spikes from all the inputs. Depending on the STDP parameters, the timing of this postsynaptic spike may evolve over the simulation, relative to the input spike times.

Do the following:

1. Run this simulation by double-clicking **STDPtiming.hoc**. Open a **Run Control** window and set **Tstop** to 200msecs. Create a voltage plot for the soma of the postsynaptic cell.
2. Press **Init&Run** to see the effect described above (you might want to make the voltage graph as big as possible). *Note any change in the timing of the postsynaptic spike relative to the inputs over the course of the simulation*.
3. Plot the weights (press the **Plot weights** button in the **Weights & Delays** window) and *note the pattern of weights across the inputs 0 to 9*. In this simulation, all the weights start at 0.012 and can saturate at a maximum of 0.12 (determined by the weight multiplier) and a minimum of 0.0012 (determined by the weight divisor). Only LTP, and not LTD, is possible, with a potentiation rate of 0.05.

4. Try setting the depression rate to 0.2 and rerunning the simulation and replotting the weights. *Note any difference you see in the voltage trace and the weight distribution.*
5. Try changing both the LTP and LTD rates individually and together to see if the "phase precession" effect on the postsynaptic spike is quicker or slower to emerge. You might need to run the simulation for more than 200msecs.

Question 6a: How and why does the timing of the postsynaptic spike change, and why does a pattern emerge of weights across the different inputs?

6b. Sequence learning (STDPnetseq.hoc)

In this exercise, 10 neurons are connected all-to-all by excitatory synapses that can undergo STDP. Each cell also receives a single external input that is strong enough to make it fire its own AP. These external inputs arrive at an interval of 10 msecs, so that the cells in the network fire in sequence at 10 msec intervals. The connections between cells are set to a weak value (0.001) so that they do not cause the receiving cell to fire. Nonetheless, STDP will change the strength of these synapses depending on the spike times of the cells. The aim of the exercise is to see what pattern of connection weights develops over time as the cells spike in sequence.

Do the following:

1. Run this simulation by double-clicking **STDPsequence.hoc**. Open a **Run Control** window and set **Tstop** to 1000 msecs.
2. Create a voltage plot that will plot the voltages from each of the 10 cells (or at least the first few cells) in a different colour on the same plot.
3. Press **Init&Run** to run the simulation for 1000 msecs.
4. The Shape window contains a color matrix representation of all the connection weights. Initially they are all the same. Once your simulation has finished, click the Update plot button, then resize the Shape window a little to get it to actually refresh the colours. Each column shows the weights of the connections a neuron makes to all others in the network, working from the bottom up. The hotter the colour, the stronger the connection. *Note any pattern you can see in the weights that have developed across all the neurons.*
5. The initial simulation contains only LTP. Set the depression rate to 0.2 as well and rerun the simulation. Update the Shape window as above. *Can you see any difference in the pattern of weights now?*
6. Try different LTP (Potentiation) and LTD (Depression) rates.
7. Try also changing the delay between successive inputs. *How do you think this might interact with the time windows of LTP and LTD (see the STDP figure above)?*

Question 6b: What patterns of weights emerge? Can you explain why? And how do the patterns vary with different rates of LTP and LTD and with different delays between spikes?

Advanced exercise:

Try changing the inputs to each cell so that they are noisy spike trains with no particular phase relationship to each other (change the "start" and "noise" values for each input NetStim object). Also make them such that a few of the cells get high frequency input (around 100 Hz) while the others get much lower frequency input (eg 20 Hz). Run simulations of this with both LTP and LTD rates set at various values, and examine the weight matrix that results. *Do the faster firing cells become more strongly connected to each other?*

Dr B. Graham, Computing Science & Maths, University of Stirling, U.K.

7. Associative Memory in a Network of Spiking Neurons

This exercise runs the code behind figure 9.10 in PCMN that explores autoassociative recall of stored patterns in an associative memory neural network. The network consists of 100 two-compartment Pinsky-Rinzel neurons connected by all-to-all excitatory and inhibitory connections. Stored patterns consist of a random selection of 10 active neurons from the network population of 100 excitatory neurons. Fifty patterns have been stored in the network by binary Hebbian learning.

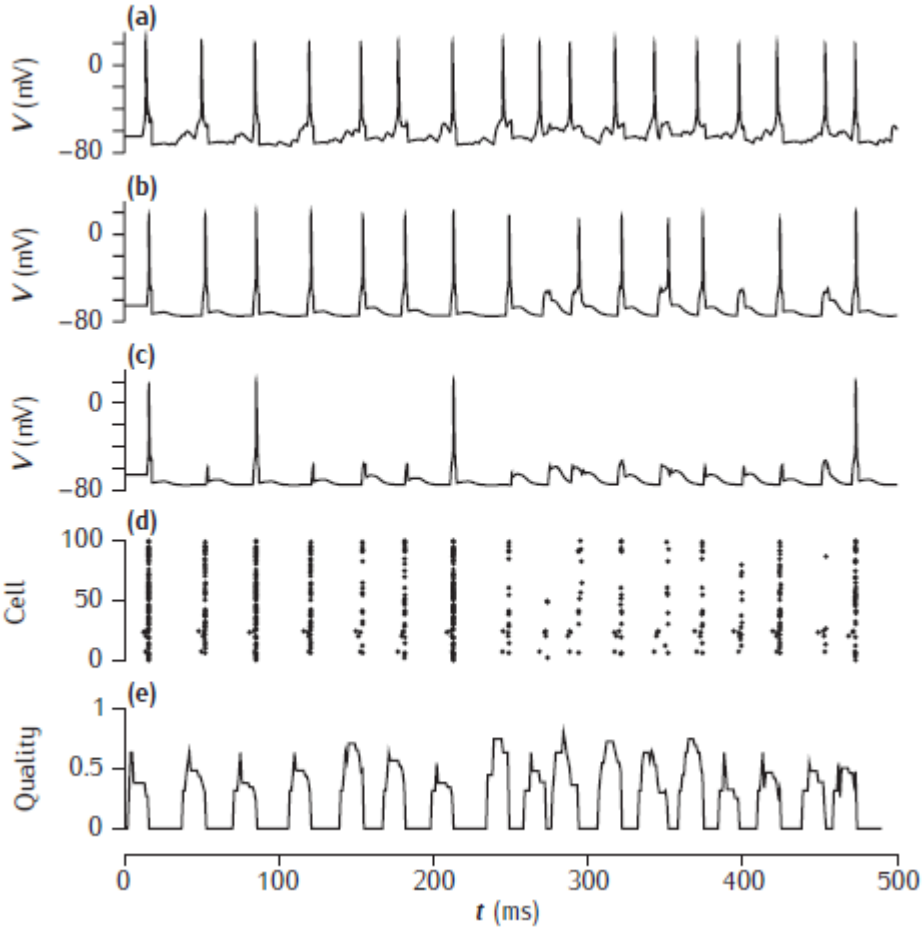


Figure 9.10 (a) Example cue cell. (b) Pattern cell. (c) Non-pattern cell. (d) Raster plot of all cells. (e) Quality of recall.

Running the book code:

Double-click on **fig-9-10.nrnzip** to run the code associated with figure 9.10 in the book. This runs a simulation in which 4 of the cells in one of the stored patterns are made active by external 500Hz excitatory input. Hopefully the network activity will largely be the recall of the entire stored pattern. Actually, recall is not perfect: all the pattern cells do fire regularly (eg top two voltage plots), but there is also occasional activity of neurons who do not belong to the pattern (eg third voltage plot). Click **Spike Plot** to see a raster plot of the entire network activity.

Do the following:

1. Try reducing the inhibitory weight by small amounts. You should start to see more spurious (non-pattern) cells firing, but they will tend to fire later than the pattern cells. Additionally, the pattern cells will start to fire in bursts.

2. Return the inhibitory weight to its original value, and try changing the inhibitory delay instead. What happens?
3. Try changing the excitatory weights and delays as well.

Question 7: How sensitive does pattern recall seem to be to these network parameters?

Examining the code:

You can load **AAM-SW.hoc** into your text editor to examine the code behind these simulations. Note that this file is not runnable by itself. You need to run **fig-9-10.nrnzip** to generate a simulation.

Changing the simulation:

You can extract all the code files for this simulation if you put **fig-9-10.nrnzip** into suitable "uncompress" software. You can then edit **AAM-SW.hoc** into your text editor to make changes to the simulation (suggested exercises below). To run your new simulation you will need to run **mknrndll** on the folder containing all your files. Then you can run the simulation by double-clicking on **mosinit.hoc**.

Advanced exercise:

Try changing the cue pattern (CPATT) and/or the number of cue cells (CFRAC; the default value of 0.3 results in 4 from 10 cells being selected for the cue) by editing these values in **AAM-SW.hoc** and running the new simulation by double-clicking on **mosinit.hoc**. The default voltage plots may no longer be for pattern and non-pattern cells as before – you can examine the stored patterns by loading **pattsN100S10P50.dat** into your text editor and trying to find indices of pattern cells to plot in your voltage graphs.

Very advanced exercise:

If you want a real challenge, implement inhibition in this network with an explicit population of inhibitory interneurons (rather than have direct inhibitory connections between the excitatory cells). Any simple model spiking cell eg simpcell.hoc or an I&F model, would be suitable to use for your inhibitory neurons.