

Chapter 1

A tour of the NEURON simulation environment

Modeling and understanding

Modeling can have many uses, but its principal benefit is to improve understanding.

The chief question that it addresses is whether what is known about a system can account for the behavior of the system. An indispensable step in modeling is to postulate a *conceptual model* that expresses what we know, or think we know, about a system, while omitting unnecessary details. This requires considerable judgment and is always vulnerable to hindsight and revision, but it is important to keep things as simple as possible. The choice of what to include and what to leave out depends strongly on the hypothesis that we are studying. The issue of how to make such decisions is outside the primary focus of this book, although from time to time we may return to it briefly.

The task of building a *computational model* should only begin after a conceptual model has been proposed. In building a computational model we struggle to establish a match between the conceptual model and its computational representation, always asking the question: would the conceptual model behave like the simulation? If not, where are the errors? If so, how can we use NEURON to help understand why the conceptual model implies that behavior?

Introducing NEURON

NEURON is a simulation environment for models of individual neurons and networks of neurons that are closely linked to experimental data. NEURON provides numerically sound, computationally efficient tools for conveniently constructing, exercising, and managing models, so that special expertise in numerical methods or programming is not required for its productive use. Increasing numbers of experimentalists and theoreticians are incorporating it into their research strategies. As of this writing, more than 460 scientific publications have reported work done with NEURON on topics that range from the molecular biology of voltage-gated channels to the operation of networks containing thousands of neurons (see **Research reports that have used NEURON** at <http://www.neuron.yale.edu/neuron/bib/usednrn.html>).

In the following pages we introduce NEURON by going through the development of a simple model from start to finish. This will require us to consider each of these steps:

1. State the question that we are interested in
2. Formulate a conceptual model
3. Implement the model in NEURON
4. Instrument the model, i.e. attach signal sources and set up graphs
5. Set up controls for running simulations
6. Save the model with instrumentation and run controls
7. Run simulation experiments

8. Analyze results

Since our aim is to provide an overview, we have chosen a simple model that illustrates just one of NEURON's strengths: the convenient representation of the spread of electrical signals in a branched dendritic architecture. We could do this by writing instructions in NEURON's programming language `hoc`, but for this example we will employ some of the tools that are provided by its graphical user interface. Later chapters examine `hoc` and the graphical tools for constructing models and managing simulations in more detail, as well as many other features and applications of the NEURON simulation environment (e.g. complex biophysical mechanisms, neural networks, analysis of experimental data, model optimization, customization of the user interface).

1. State the question

The scientific issue that motivates the design and construction of this model is the question of how synaptic efficacy is affected by synaptic location and the anatomical and biophysical properties of the postsynaptic cell. This has been the subject of too many experimental and theoretical studies to reference here. Interested readers will find numerous relevant publications in NEURON's on-line bibliography (cited above), and may retrieve working code for several of these from ModelDB (<http://senselab.med.yale.edu/senselab/modeldb/>).

2. Formulate a conceptual model

Most neurons have many branches with irregularly varying diameters and lengths (Fig. 1.1 A), and their membranes are populated with a wide assortment of ionic channels that have different ionic specificities, kinetics, dependence on voltage and second messengers, and spatial distributions. Scattered over the surface of the cell may be hundreds or thousands of synapses, some with a direct effect on ionic conductances (which may also be voltage-dependent) while others act through second messengers. Synapses themselves are far from simple, often displaying stochastic and use-dependent phenomena that can be quite prominent, and frequently being subject to various pre- and postsynaptic modulatory effects. Given all this complexity, we might well ask if it is possible to understand anything without understanding everything. From the very onset we are forced to decide what to include and what to omit.

Suppose we are already familiar with the predictions of the basic ball and stick model (Rall 1977; Jack et al. 1983), and that experimental observations motivate us to ask questions such as: How do synaptic responses observed at the soma vary with synaptic location if dendrites of different diameters and lengths are attached to the soma? What happens if some parts of the cell have active currents, while others are passive? What if a neuromodulator or shift of the background level of synaptic input changes membrane conductance?

Then our conceptual model might be similar to the one shown in Fig. 1.1 B. This model includes a neuron with a soma that gives rise to an axon and two dendritic trunks, and a single excitatory synapse that may be located at any point on the cell.

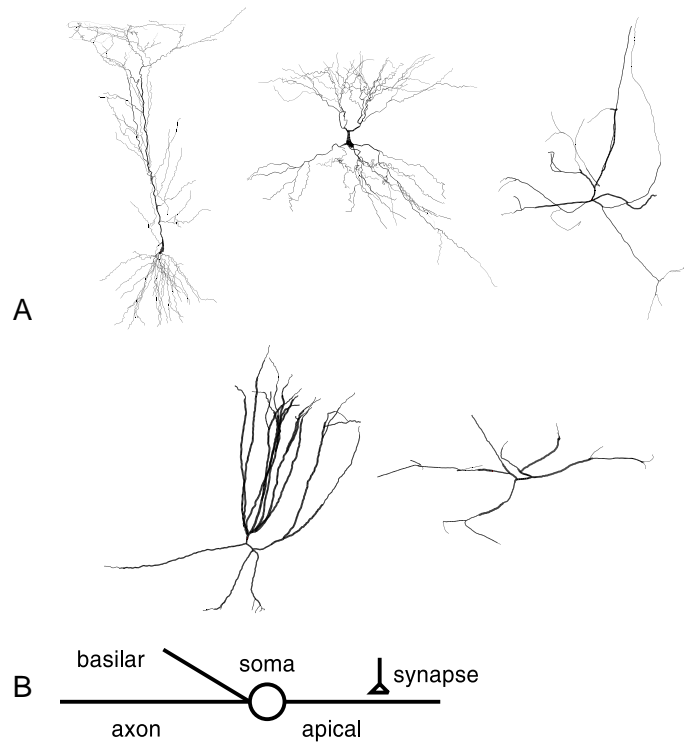


Figure 1.1. A. Clockwise from top left: Ca1 and Ca3 pyramidal neurons (from D.A. Turner); calbindin-, parvalbumin-, and calretinin-positive interneurons (from A.I. Gulyás). B. Our conceptual model neuron. The conductance change synapse can be located anywhere on the cell.

Although deliberately more complex than the prototypical ball and stick, the anatomical and biophysical properties of our model are much simpler than the biological original (Table 1.1). The axon and dendrites are simple cylinders, with uniform diameters and membrane properties along their lengths. The dendrites are passive, while the soma and axon have Hodgkin-Huxley sodium, potassium, and leak currents, and are capable of generating action potentials (Hodgkin and Huxley 1952). A single synaptic activation causes a localized transient conductance increase with a time course described by an alpha function

$$g_s(t) = \begin{cases} 0 & \text{for } t < t_{act} \\ g_{max} \frac{(t-t_{act})}{\tau_s} e^{-\frac{(t-t_{act})}{\tau_s}} & \text{for } t \geq t_{act} \end{cases} \quad \text{Eq. 1.1}$$

where t_{act} is the time of synaptic activation, and g_s reaches a peak value of g_{max} at $t = \tau_s$ (see Table 1.2 for parameter values). This conductance increase mechanism is just slightly more complex than the ideal current sources used in many theoretical studies (Rall 1977; Jack et al. 1983), but it is still only a pale imitation of any real synapse (Bliss and Lømo 1973; Ito 1989; Castro-Alamancos and Connors 1997; Thomson and Deuchars 1997).

Table 1.1. Model cell parameters

	Length μm	Diameter μm	Biophysics
soma	30	30	HH g_{Na} , g_K , and g_{leak}
apical dendrite	600	1	passive with $Rm = 5,000 \Omega \text{ cm}^2$, $E_{pas} = -65 \text{ mV}$
basilar dendrite	200	2	same as apical dendrite
axon	1000	1	same as soma
$Cm = 1 \mu\text{f} / \text{cm}^2$			
cytoplasmic resistivity = $100 \Omega \text{ cm}$			
Temperature = 6.3°C			

Table 1.2. Synaptic mechanism parameters

g_{max}	0.05 μS
τ_s	0.1 ms
E_s	0 mV

3. Implement the model in NEURON

With a clear picture of our model in mind, we are ready to express it in the form of a computational model. Instead of writing instructions in NEURON's programming language `hoc`, for this example we will employ some of the tools that are provided by NEURON's graphical user interface.

We begin with the **CellBuilder**, a graphical tool for constructing and managing models of individual neurons. At this stage, we are not considering synapses, stimulating electrodes, or simulation controls. Instead we are focussing on creating a representation of the continuous properties of the cell. Even if we were not using the **CellBuilder** but instead were developing our model entirely with `hoc` code, it would probably be best for us to follow a similar approach, i.e. specify the biological attributes of the model cell separately from the specification of the instrumentation and control code that we will use to exercise the model. This is an example of modular programming, which is related to the "divide and conquer" strategy of breaking a large and complex problem into smaller, more tractable steps.

The **CellBuilder** makes it easier for us to create a model of a neuron by allowing us to specify its architecture and biophysical properties through a graphical interface. When we are satisfied with the specification, the **CellBuilder** will generate the corresponding `hoc` code for us. Once we have a model cell, we will be ready to use other graphical tools to attach a synapse to it and plot simulation results (see **4. Instrument the model** below).

The images in the following discussion were obtained under MSWindows; the appearance of NEURON under UNIX, Linux, and MacOS is quite similar.

Starting and stopping NEURON

No matter what a program does, the first thing you have to learn is how to start and stop it. To start NEURON under UNIX or Linux, just type `nrngui` on the command line and skip the remainder of this paragraph. Under MSWindows, double click on the `nrngui` icon on your desktop (Fig. 1.2 left); if you don't see one there, bring up the NEURON program group (i.e. use Start / Program Files / NEURON) and select the `nrngui` item (Fig. 1.2 right). If you are using MacOS, open the folder where you installed NEURON and double click on the `nrngui` icon.

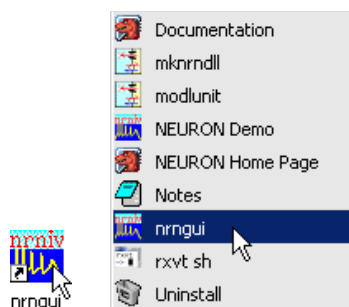


Figure 1.2. Under MSWindows, start NEURON by clicking on the `nrngui` icon on the desktop (left) or selecting the `nrngui` item in the NEURON program group (right).

You should now see the NEURON Main Menu (Fig. 1.3 top), which offers a set of menus for bringing up graphical tools for creating models and running simulations. If you are using UNIX or Linux, a "banner" that includes the version of NEURON you are running will be printed in the xterm where you typed `nrngui`, and the prompt will change to `oc>` to indicate that NEURON's `hoc` interpreter is running. Under MacOS and MSWindows, the "banner" and `oc>` prompt will appear in a new console window (Fig. 1.3 bottom).

There are three different ways to exit NEURON; you can use whichever is most convenient.

1. type `^D` (i.e. control D) at the `oc>` prompt
2. type `quit()` at the `oc>` prompt
3. click on File in the NEURON Main Menu, scroll down to Quit, and release the mouse button (Fig. 1.4)

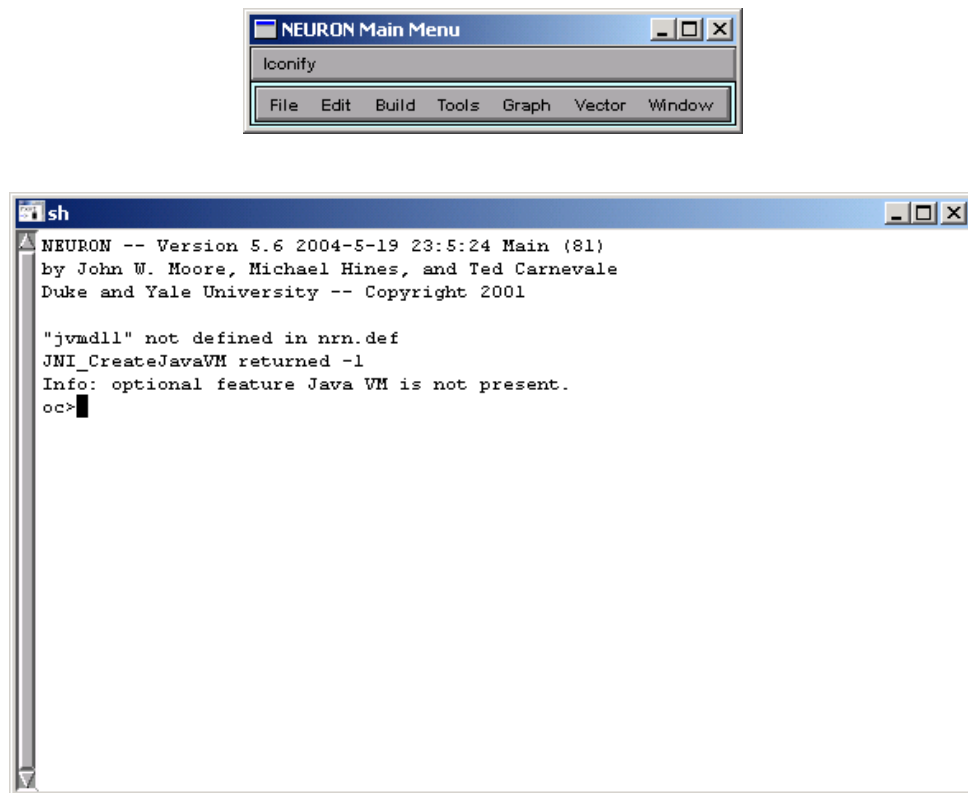


Figure 1.3. Top: The NEURON Main Menu toolbar. Bottom: NEURON's "banner" and `oc>` prompt in an MSWindows console window.

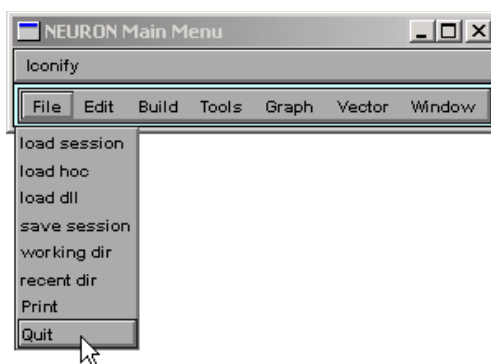


Figure 1.4. One way to exit NEURON is to click on File / Quit in the NEURON Main Menu toolbar.

Bringing up a CellBuilder

To get a CellBuilder just click on Build in the NEURON Main Menu, scroll down to the CellBuilder item, and release the mouse button (Fig. 1.5).

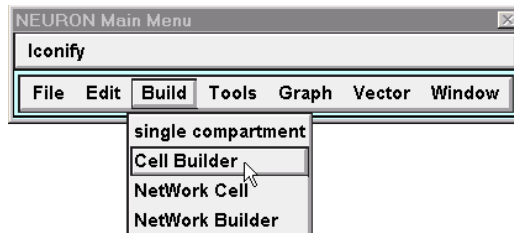


Figure 1.5. Using the NEURON Main Menu to bring up a CellBuilder.

Across the top of the CellBuilder is a row of radio buttons and a checkbox, which correspond to the sequence of steps involved in building a model cell (Fig. 1.6). Each radio button brings up a different page of the CellBuilder, and each page provides a view of the model plus a graphical interface for defining properties of the model. The first four pages (Topology, Subsets, Geometry, Biophysics) are used to create a complete specification of a model cell. On the Topology page, we will set up the branched

architecture of the model and give a name to each branch, without regard to diameter, length, or biophysical properties. We will deal with length and diameter on the Geometry page, and the Biophysics page is where we will define the properties of the membrane and cytoplasm of each of the branches.



Figure 1.6. Top panel of the CellBuilder

The Subsets page deserves special comment. In almost every model that has more than one branch, two or more branches will have at least some biophysical attributes that are identical, and there are often significant anatomical similarities as well. Furthermore, we can almost always apply the `d_lambda` rule for compartmentalization throughout the entire cell (see below). We can take advantages of such regularities by assigning shared properties to several branches at once. The Subsets page is where we group branches into subsets, on the basis of shared features, with an eye to exploiting these commonalities on the Geometry and Biophysics pages. This allows us to create a model specification that is compact, efficient, and easily understood.

Enter the specifications of the model cell

- **Topology**

We start by using the Topology page to set up the branched architecture of the model. As Fig. 1.7 shows, when a new CellBuilder is created, it already contains a branch (or "section," as it is called in NEURON) that will serve as the root of the branched

architecture of the model (the root of a tree is the branch that has no parent). This root section is initially called "soma," but we can rename it if we desire (see below).

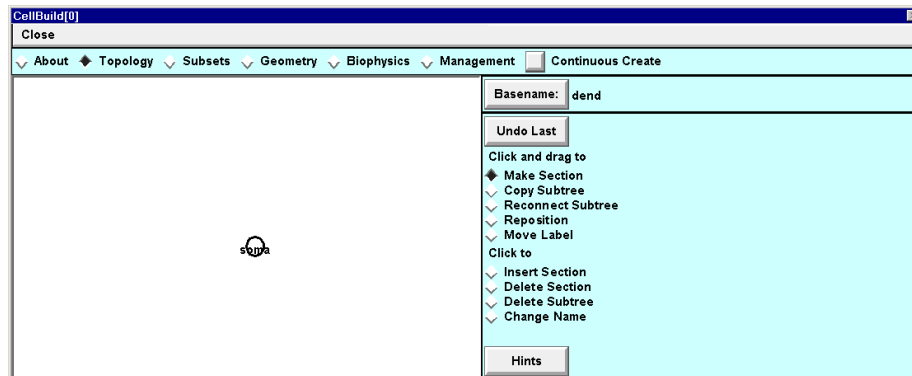
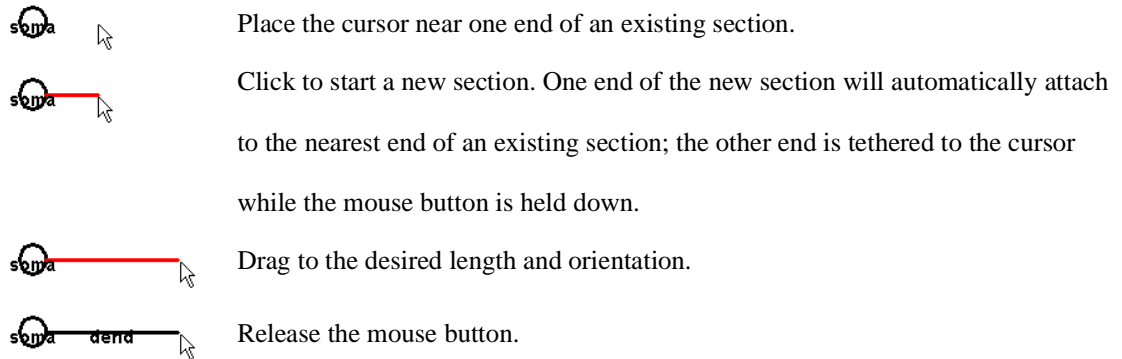


Figure 1.7. The Topology page. The left panel shows a simple diagram of the model, which is called a "shape plot." The right panel contains many functions for editing the branched architecture of a model cell.

The Topology page offers many functions for creating and editing individual sections and subtrees. We can make the section that will become our apical dendrite by following the steps presented in Fig. 1.8. Repeating these actions a couple more times (and resorting to functions like Undo Last, Reposition, and Delete Section as needed to correct mistakes) gives us the basilar dendrite and axon.

Figure 1.8. Making a new section. Verify that the Make Section radio button is *on*, and then perform the following steps.



Our model cell should now look like Fig. 1.9. At this point some minor changes would improve its appearance: moving the labels away from the sections so they are easier to read (Fig. 1.10), and then renaming the apical and basilar dendrites and the axon (Figs. 1.11 and 12). The final result should resemble Fig. 1.13.

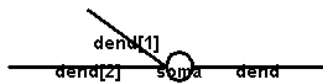


Figure 1.9. The model after all sections have been created.

Figure 1.10. To change the location of a label,

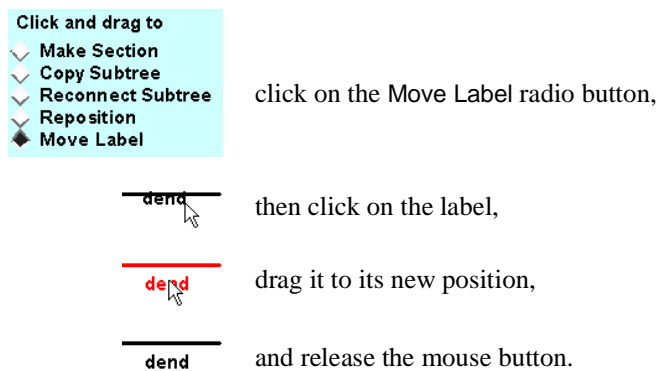


Figure 1.11. Preparing to change the name of a section. Each section we created was automatically given a name based on "dend." To change these names, we must first change the base name as shown here.

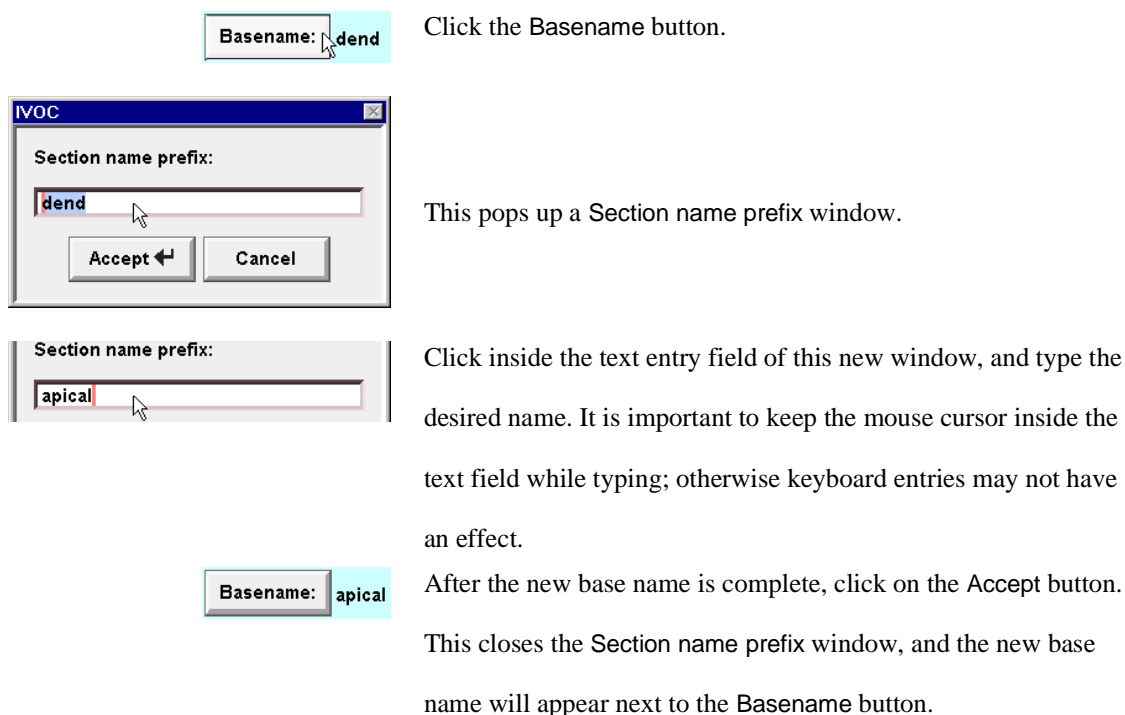


Figure 1.12. Changing the name of a section.

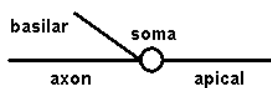
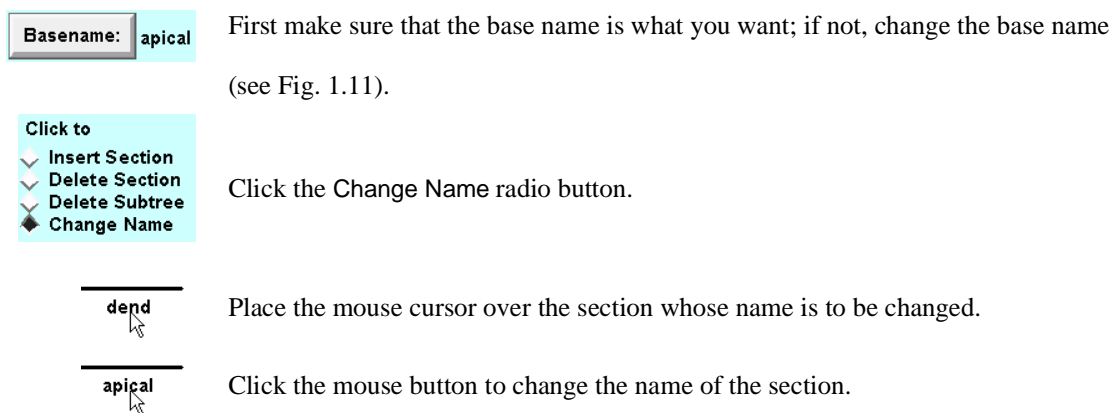


Figure 1.13. The shape plot of the model with labels positioned and named as desired.

● Subsets

As mentioned above, the Subsets page (Fig. 1.14) is for grouping sections that share common features. Well-chosen subsets can save a lot of effort later by helping us create very compact specifications of anatomical and biophysical properties.

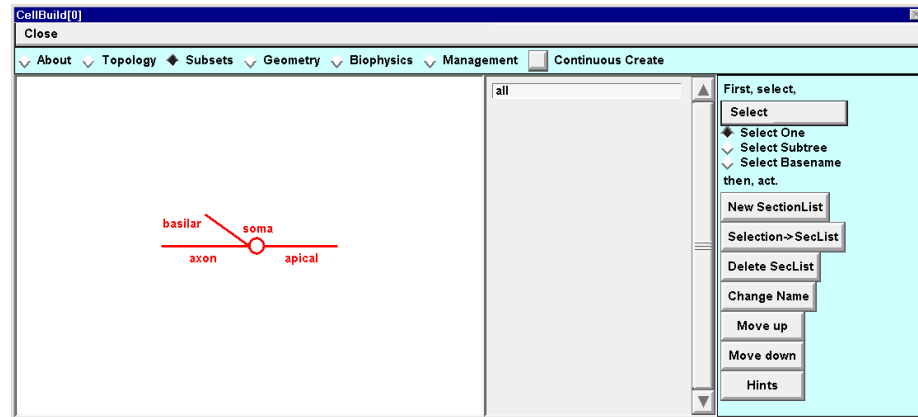


Figure 1.14. The Subsets page. The middle panel lists the names of all existing subsets. In the shape plot, the sections that belong to the currently selected subset are shown in red. When the Subsets page initially appears, it already has an all subset that contains every section in the model.

The properties of the sections in this particular example suggest that we create two subsets: one that contains the basilar and apical branches, which are passive, and another that contains the soma and axon, which have Hodgkin-Huxley spike currents. To make a subset called `has_HH` that contains the sections with HH currents, follow the steps in Fig. 1.15. Then make another subset called `no_HH` that contains the basilar and apical dendrites.

Figure 1.15. Making a new subset.



With the **Select One** radio button on (Fig. 1.14), click on the axon and soma sections while holding down the shift key. The selected sections will be indicated in red . . .

. . . and the list of subsets will change to show that **all** is not the same as the set {axon, soma}.

Next, click on the **New SectionList** button (a subset is a list of sections).

This pops up a window that asks you to enter a name for the new SectionList.

Click inside the text entry field of this new window and type the name of the new subset, then click on the **Accept** button.

The new subset name will appear in the middle panel of the CellBuilder.

● Geometry

In order to use the **Geometry** page (Fig. 1.16) to specify the anatomical dimensions of the sections and the spatial resolution of our model, we must first set up a strategy for assigning these properties. After we have built our (hopefully efficient) strategy, we will give them specific values.

The geometry strategy for our model is simple. Each section has different dimensions, so the length *L* and diameter *diam* of each section must be entered individually. However, for each section we will let NEURON decide how fine to make the spatial grid, based on a fraction of the length constant at 100 Hz (spatial accuracy and NEURON's tools for

adjusting the spatial grid are discussed in **Chapter 5**). Figure 1.17 shows how to set up this strategy.

Having set up the strategy, we are ready to assign the geometric parameters (see Figs. 1.18 and 19).

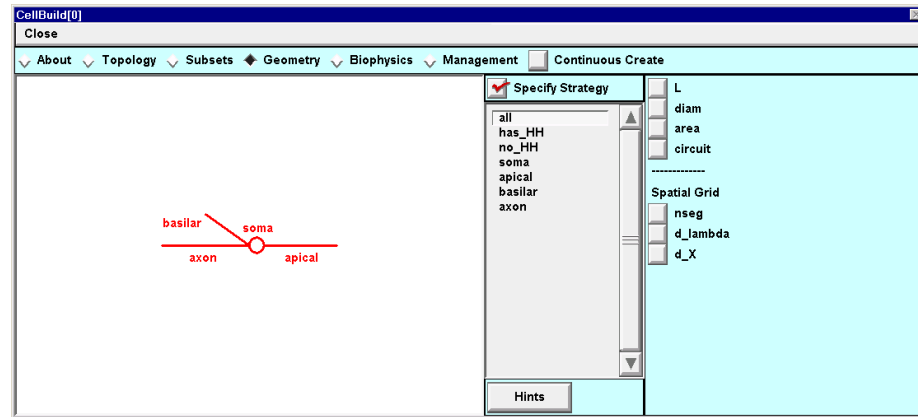
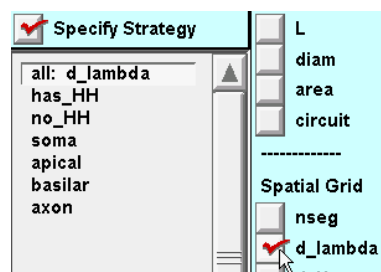


Figure 1.16. When the Geometry page in a new CellBuilder is first viewed, a red check mark should appear in the Specify Strategy checkbox. If not, clicking on the checkbox will toggle Specify Strategy *on*.

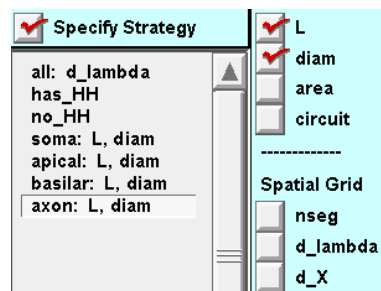
Figure 1.17. Specifying strategy for assignment of geometric parameters. First make sure that Specify Strategy contains a red check (see Fig. 1.16). Then proceed with the following steps.



For the all subset, toggle d_lambda on.



Select soma in the middle panel, and then toggle L and diam on.



Repeat for apical, basilar, and axon, and the result should resemble this figure.

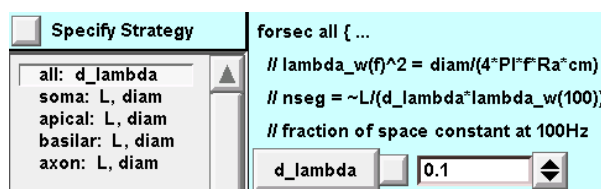


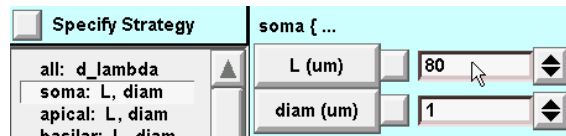
Figure 1.18. Assigning values to the geometric parameters. Toggling Specify Strategy off makes the middle panel show only the subsets and sections that we selected when setting up our strategy. Adjacent to each of these are the names of the parameters that are to be reviewed and perhaps changed. Here the subset all is selected; the right panel displays the current value of the parameter associated with it (d_lambda) and offers us the means to change this parameter if necessary. According to the d_lambda criterion for spatial resolution,

NEURON will automatically discretize the model, breaking each section into compartments small enough that none will be longer than `d_lambda` at 100 Hz.

The default value of `d_lambda` is 0.1, i.e. 10% of the AC length constant. This is short enough for most purposes, so we do not need to change it.

Discretization is discussed in **Chapter 5**.

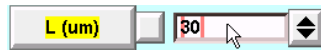
Figure 1.19. Assigning values to the geometric parameters *continued*.



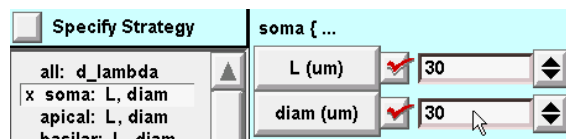
The length and diameter of each section must be changed from the default values.



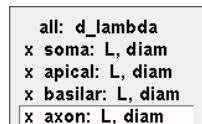
To set the length of the soma to 30 μm , first click inside the numeric field for L so that a red editing cursor appears.



Then use the backspace key to delete the old value, and finally type in the new value.



After doing the same for diam, the dimensions of soma should look like this. The checkboxes adjacent to the L and diam buttons indicate that these parameters have been changed from their default values. The x in the middle panel is another reminder that at least one of the parameters associated with soma has been changed.



After adjusting L and diam for the dendrites and the axon, the middle panel shows an x next to the name of each section.

- Biophysics

The Biophysics page (Fig. 1.20) is used to insert biophysical properties of membrane and cytoplasm (e.g. Ra, Cm, ion channels, buffers, pumps) into subsets and individual sections. As with the Geometry page, first we set up our strategy (Fig. 1.21), and then we review and adjust parameter values (Fig. 1.22). The CellBuilder will then contain a complete specification of our model.

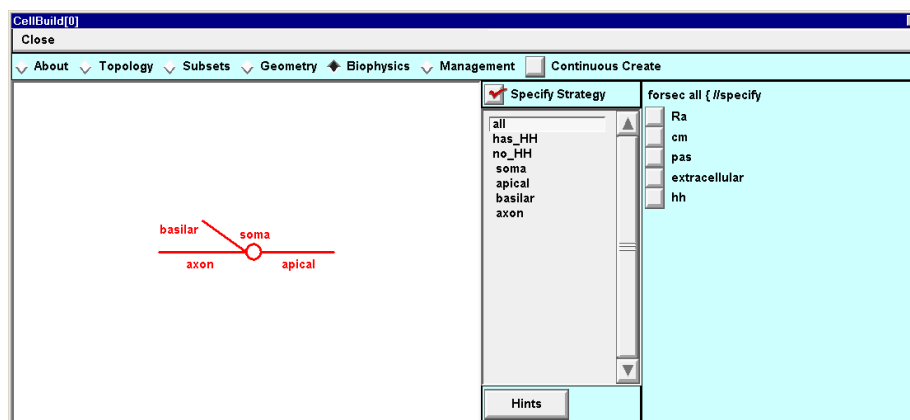
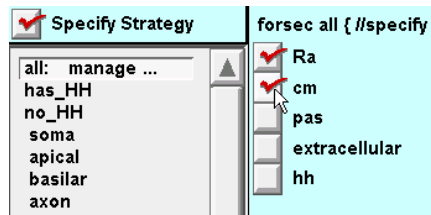
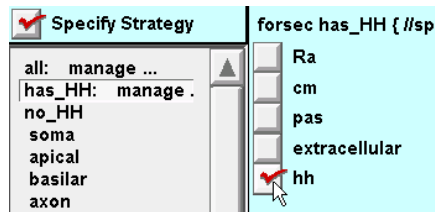


Figure 1.20. The Biophysics page, ready for specification of strategy. The right panel shows the mechanisms that are available to be inserted into our model. For this simple example, the number of mechanisms is deliberately small; adding new mechanisms is covered in **Chapter 9: How to expand NEURON's library of mechanisms**.

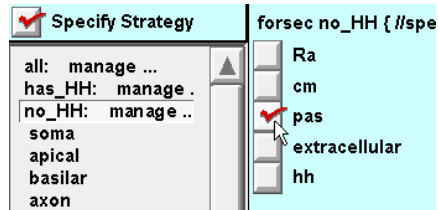
Figure 1.21. Specifying strategy for assignment of biophysical parameters. First make sure that Specify Strategy contains a red check, then proceed with the following steps.



For the all subset, toggle Ra (cytoplasmic resistivity) and cm (specific membrane capacitance) *on*.

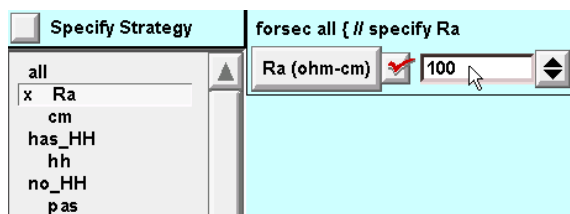


Select the has_HH subset in the middle panel, and then toggle HH *on*.

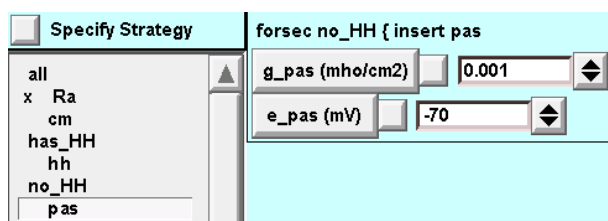


Finally select the no_HH subset and toggle pas *on*.

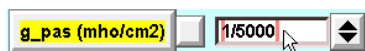
Figure 1.22. Assigning values to the biophysical parameters. Toggling *Specify Strategy* *off* shows a list of the names of the subsets that are part of the strategy. Beneath each subset are the names of the mechanisms that are associated with it. Clicking on a mechanism brings up a set of controls in the right panel for displaying and adjusting the parameters of the mechanism.



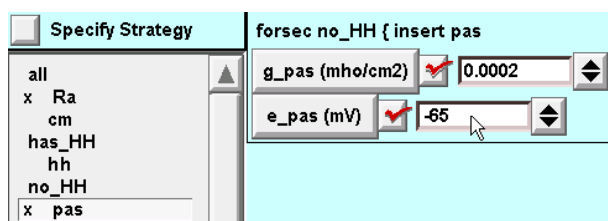
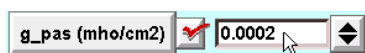
For the subset all, change the value of Ra from its default (80 Ω cm) to the desired value of 100 Ω cm.



The sections in the no_HH subset have a passive current whose parameters must be changed from their defaults (shown here).



The value of g_pas can be set by deleting the default and then typing 1/5000 (= 1/Rm).



The final values of g_pas and e_pas. Not shown: cm (all subset) and the parameters of the hh mechanism (has_HH subset), which have the desired values by default and do not need to be changed, although it is good practice to review them.

Save the model cell

After investing time and effort to set up our model, we would be wise to take just a moment to save it. The CellBuilder, like NEURON's other graphical windows, can be saved to disk as a "session file" for future re-use, as shown in Figures 1.23 and 1.24. For more information about saving and retrieving session files, including how to use the Print

& File Window Manager GUI tool to select and save specific windows, see **Using**

Session Files for Saving and Retrieving Windows at

<http://www.neuron.yale.edu/neuron/docs/saveses/saveses.html>

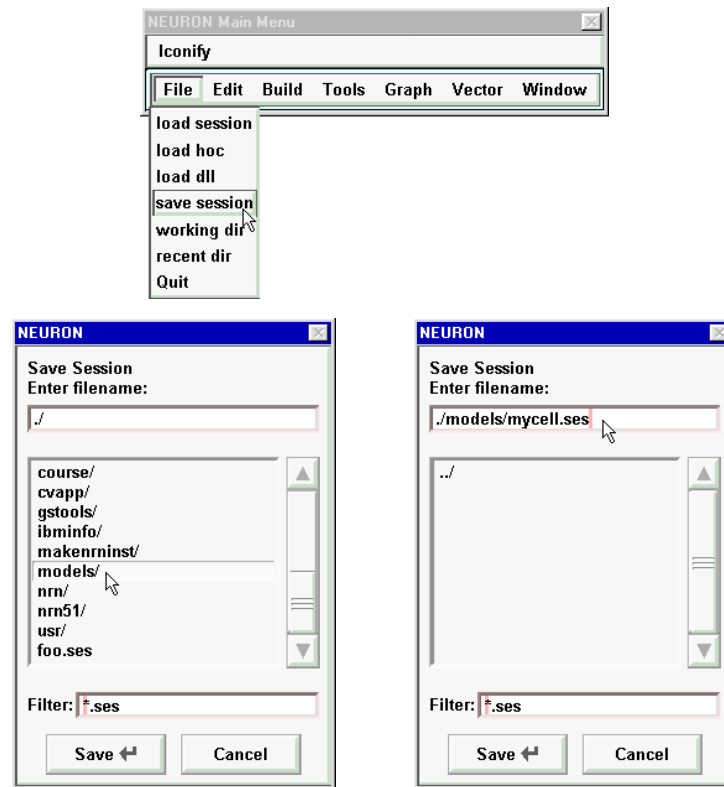


Figure 1.23. Top: To save all of NEURON's graphical windows into a session file, first click on File in the NEURON Main Menu and scroll down to save session. Bottom left: This brings up a directory browser that can be used to navigate to the directory where the session file will be saved. Bottom right: Click in the edit field at the top of the directory browser and type the name to use for the session file, then click on the Save button.

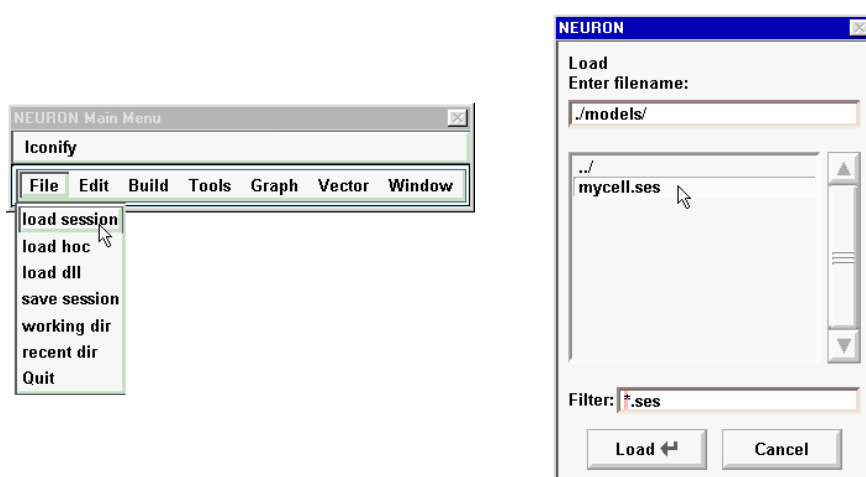


Figure 1.24. Left: To recreate the graphical windows that were saved to a session file, first click on File in the NEURON Main Menu and scroll down to load session. Right: Use the directory browser that appears to navigate to the directory where the session file was saved. Then double click on the session file that you want to retrieve.

Execute the model specification

Now that the CellBuilder contains a complete specification of the model cell, we could use the Export button on the Management page (see **Chapter 6**) to write out a hoc file that, when executed by NEURON, would create the model. However, for this example we will just turn Continuous Create *on* (Fig. 1.25). This makes the CellBuilder send its output directly to NEURON's interpreter without bothering to write a hoc file. The model cell whose specifications are contained in the CellBuilder is now available to be used in simulations.

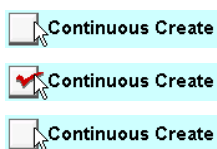


Figure 1.25. Continuous Create is initially *off*, but clicking on the adjacent button toggles it *on* and *off*.

If we make any changes to the model while Continuous Create is *on*, the CellBuilder will automatically send new code to the interpreter. This can be very convenient during model development, since it allows us to quickly examine the effects of any change. Automatic updates might bog things down if we were dealing with a large model on a slow machine. In such a case, we could just turn Continuous Create *off*, make whatever changes were necessary, and then cycle it *on* and *off* again.

4. Instrument the model

Signal sources

In the NEURON simulation environment, a synapse or electrode for passing current (current clamp or voltage clamp) is represented by a point source of current which is associated with a localized conductance. These signal sources are called "point processes" to distinguish them from properties that are distributed over the cell surface (e.g. membrane capacitance, active and passive ionic conductances) or throughout the cytoplasm (e.g. buffers), which are called "distributed mechanisms" or "density mechanisms."

We have already seen how to use one of NEURON's graphical tools for dealing with distributed mechanisms (the CellBuilder). To attach a synapse to our model cell, we turn to one of NEURON's tools for dealing with point processes: the PointProcessManager (Fig. 1.26). Using a PointProcessManager we can specify the type and parameters of the point process (Fig. 1.27) and where it is attached to the cell.

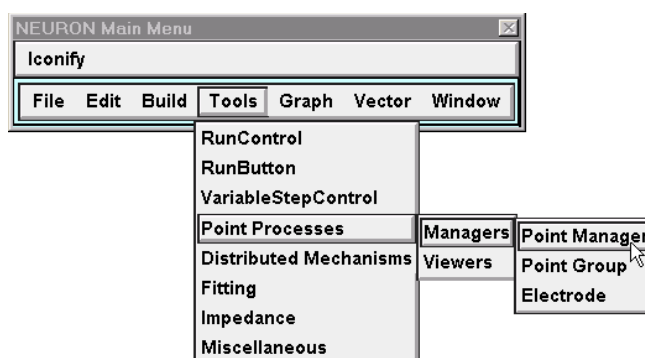
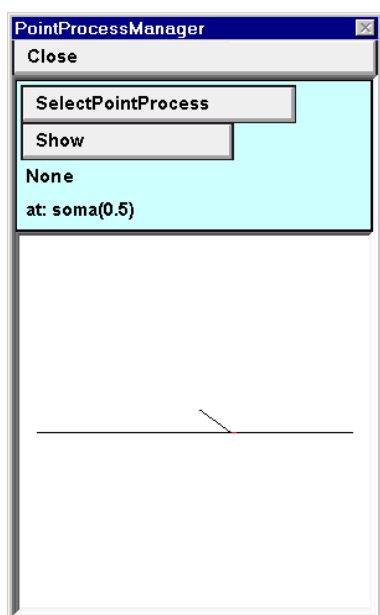
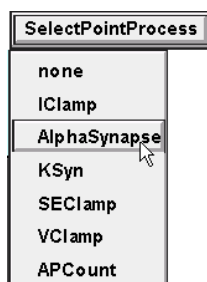


Figure 1.26. Bringing up a PointProcessManager in order to attach a synapse to our model cell. In the NEURON Main Menu, click on Tools / Point Processes / Managers / Point Manager, then proceed as shown in Fig. 1.27.

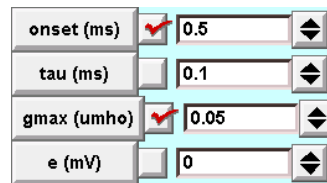
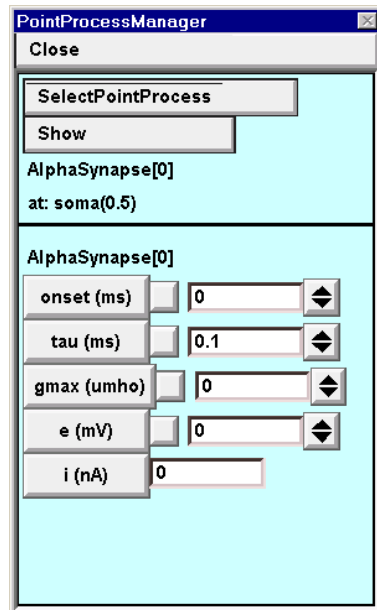
Figure 1.27. Configuring a new PointProcessManager to emulate a synapse.



A. Note the labels in the top panel. None means that a signal source has not yet been created. The bottom panel shows a stick figure of our model cell.



B. SelectPointProcess / AlphaSynapse creates a point process that emulates a synapse with a conductance change governed by Eq. 1.1, and shows us a panel for adjusting its parameters.



C. The top panel of the PointProcessManager indicates what kind of point process has been specified, and where it is located (in this case, at the midpoint of the soma). The bottom panel shows the parameters of an AlphaSynapse: its start time onset and time constant τ (t_{act} and τ_s in Eq. 1.1), peak conductance g_{max} (g_{max} in Eq. 1.1), and reversal potential e (E_s in Table 1.2). The button marked i (nA) is just a label for the adjacent numeric field, which displays the instantaneous synaptic current.

D. For this example change onset to 0.5 ms and g_{max} to 0.05 μS ; leave τ and e unchanged.

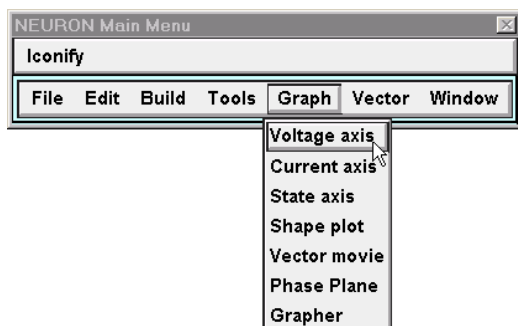
Signal monitors

Since one motivation for the model is to examine how synaptic responses observed at the soma vary with synaptic location, we want a graph that shows the time course of somatic membrane potential. In the laboratory this would ordinarily require attaching an electrode to the soma, so in a NEURON simulation it might seem to require a point process. However, the computer automatically evaluates somatic V_m in the course of a simulation. In other words, graphing V_m doesn't really change the system, unlike attaching a signal source, which adds new equations to the system. This means that a

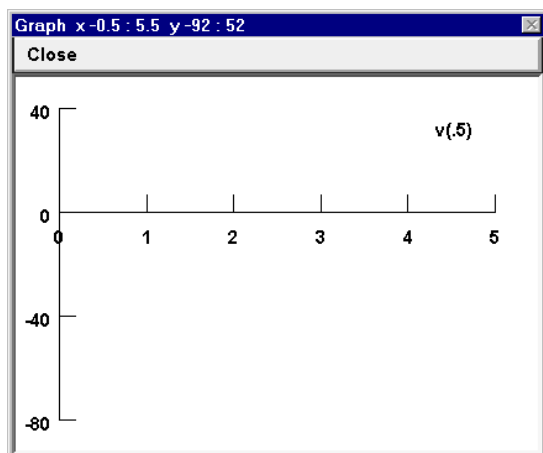
point process is not needed; instead, we just bring up a graph that includes somatic V_m in the list of variables that it plots (see Fig. 1.28).

We could monitor V_m at other locations by adding more variables to this graph, and bring up additional graphs if this one became too crowded. However, it can be more informative and convenient to create a "space plot" (Fig. 1.29), which shows V_m as a function of position along one or more branches of a cell. This graph will change throughout the simulation run, displaying the evolution of V_m as a function of space and time.

Figure 1.28. Creating a graph to display somatic membrane potential as a function of time.

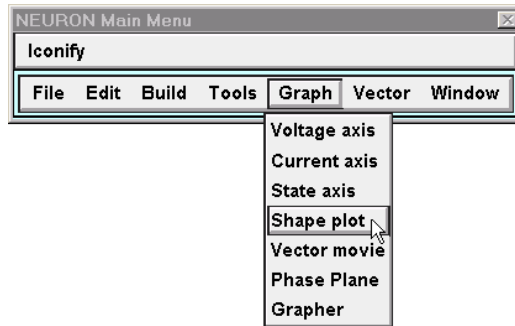


A. Click on Graph / Voltage axis in the NEURON Main Menu.

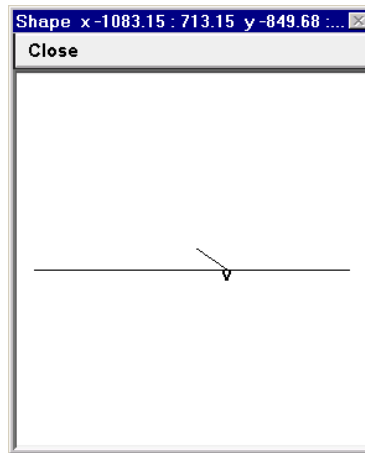


B. In the graph that appears, the horizontal axis is in milliseconds and the vertical axis is in millivolts. The label $v(5)$ signifies that this graph will show V_m at the middle of the default section. With the CellBuilder, this is always the root section, which in this example is the soma (the concepts of "root section" and "default section" are discussed in **Chapter 5**).

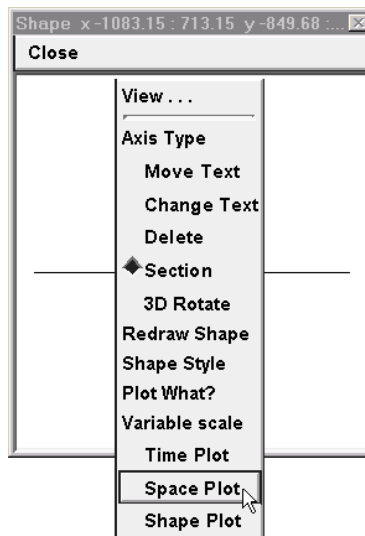
Figure 1.29. Setting up a space plot.



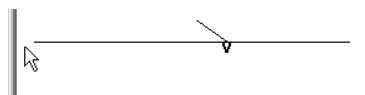
A. The first step is to create a Shape plot by clicking on Graph / Shape plot in the NEURON Main Menu.



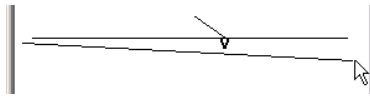
B. This brings up a Shape plot window, which is used to create the space plot.



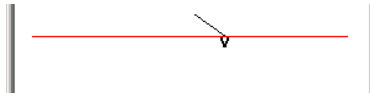
C. Right click in the Shape plot window to bring up its primary menu. While still pressing the mouse button, scroll down the menu to the Space Plot item, then release the button.



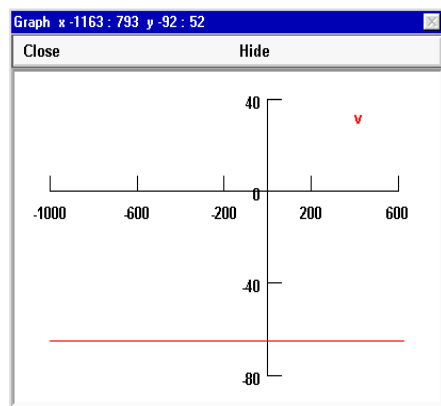
D. Place the cursor just to the left of the distal end of the axon and press the left mouse button.



E. While still holding the button down, drag the cursor across the window to the right, finally releasing the button when the cursor has passed the distal end of the apical dendrite.



F. The branches along the selected path (axon, soma, and apical dendrite) are now shown in red, and a new graph window appears (see G). If you like, you may now click on the Close button at the upper left corner of the shape plot window to conserve screen space



G. The x axis of the Space Plot window shows the distance from the 0 end of the default section, which in this example is the left end of the soma.

5. Set up controls for running the simulation

At this point we have a model cell with a synapse attached to the soma, and a graphical display of somatic V_m . All that is missing is a means to start and control the subsequent course of a simulation run. This is provided by the RunControl window (Fig. 1.30), which allows us to specify many more options than we will use in this example.

6. Save model with instrumentation and run control

After rearranging the RunControl, PointProcessManager, and graph window, our customized user interface for running simulations and observing simulation results should look something like Fig. 1.31. For the sake of safety and possible future convenience, it is a good idea to use NEURON Main Menu / File / Save Session to save this custom GUI to a session file.

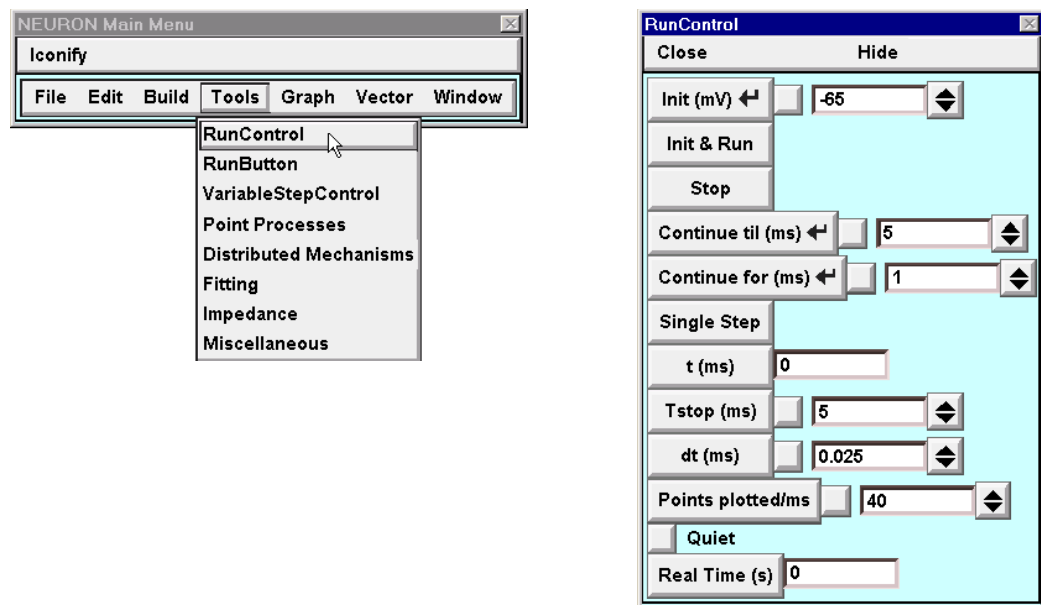


Figure 1.30. Left: To bring up a window with controls for running simulations, click on the RunControl button in NEURON Main Menu / Tools. Right: The RunControl window provides many options for controlling the overall time course of a simulation run. For this example, only three of these controls are relevant.

1. Init (mV) sets time t to 0, assigns the displayed starting value (-65 mV) to V_m throughout the model cell, and sets the ionic conductances to their steady state values at this potential.
2. Init & Run performs the same initialization as Init (mV), and then starts a simulation run.
3. Points plotted/ms determines how often the graphical displays are updated during a simulation.

Three other items in this panel are of obvious interest, although we will not do anything with them in this example. The first is dt , which sets the size of the time intervals at which the equations that describe the model are solved. The second is $Tstop$, which specifies the duration of a simulation run. Finally, the button marked t doesn't actually do anything but is just a label for the adjacent numeric field, which displays the elapsed simulation time. Additional features of the RunControl window are discussed in **Chapter 7: How to control simulations**.

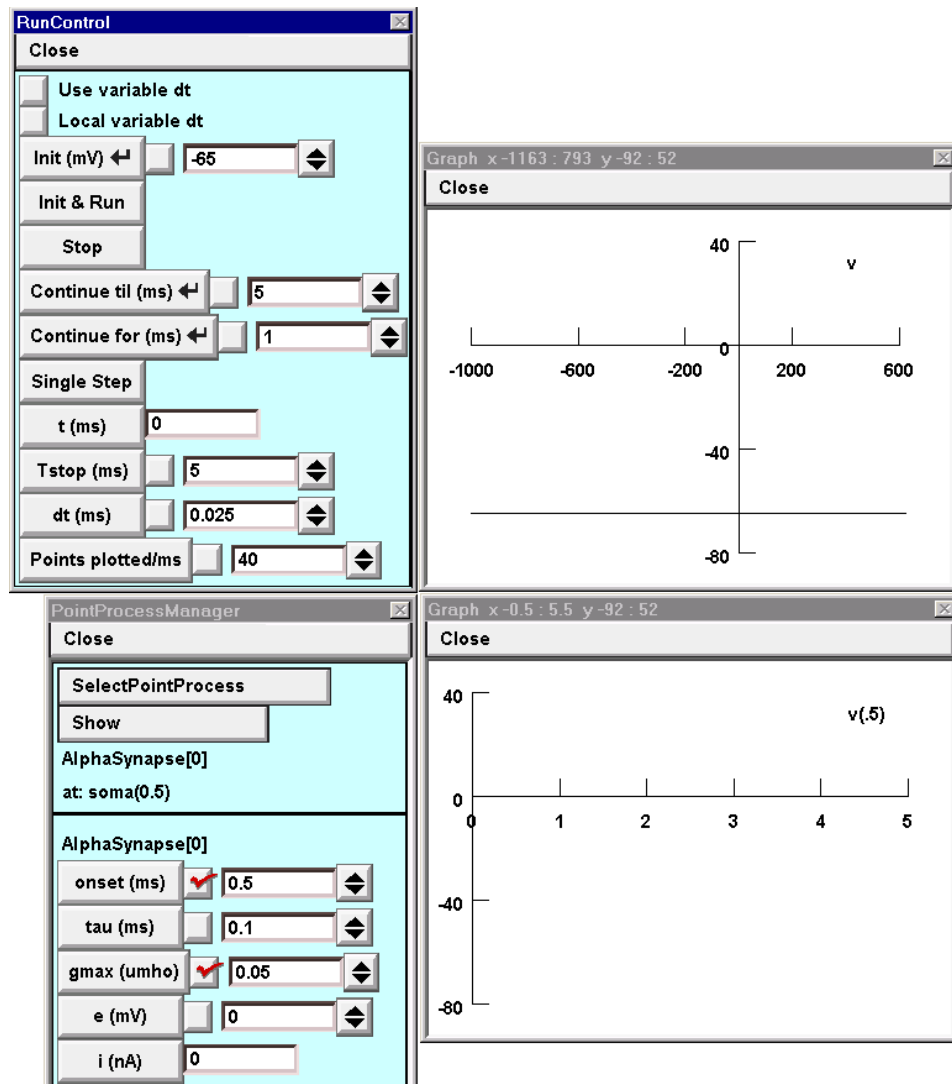


Figure 1.31. The windows we will use to run simulations and observe simulation results. Other windows that are present on the screen but not shown in this figure are the NEURON Main Menu and the CellBuilder.

7. Run the simulation experiment

We are now ready to use our "virtual experimental rig" to exercise the model. When we run a simulation with the synapse located at the soma (Fig. 1.32 and 33), a spike is

triggered. However, if we move the synapse even a small distance away from the soma along the apical dendrite (Fig. 1.34) and run a new simulation, the epsp is too small to evoke a spike (Fig. 1.35).

The utility of the space plot as a tool for understanding the temporal evolution of V_m throughout the cell can be enhanced by using it like a storage oscilloscope, as shown in Fig. 1.36. This allows us to compare the distribution of V_m at successive intervals during a run. It might be helpful to do something similar with the plot of somatic V_m vs. t if we wanted to compare responses to synaptic inputs with different parameters or locations.

Figure 1.32. Running a simulation.

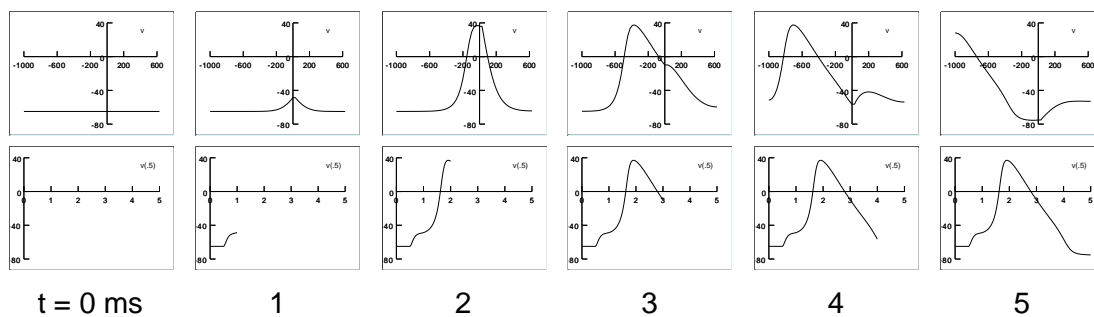
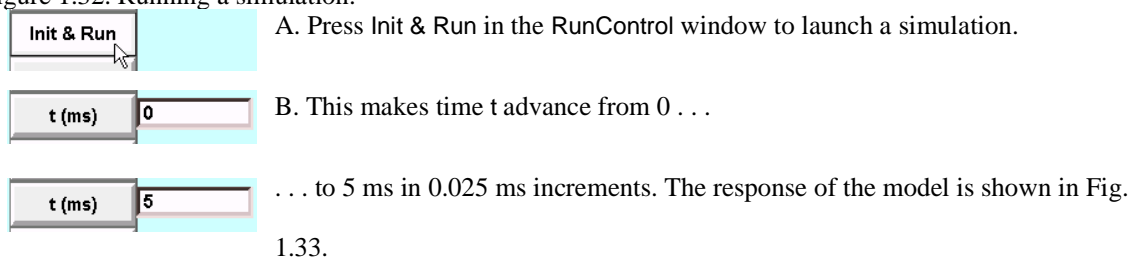
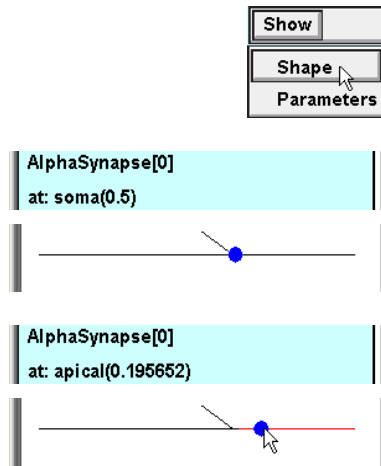


Figure 1.33. Snapshots of the space plot (top) and the graph of V_m vs. t at the soma (bottom) taken at 1 ms intervals. Synaptic input at the soma triggers a spike that propagates actively along the axon and spreads with passive decrement into the apical dendrite.

Figure 1.34. Changing synaptic location.



A. In the top panel of the PointProcessManager, click on Show and scroll down to Shape.

B. The top panel remains unchanged, but the bottom panel of the PointProcessManager now displays a shape plot of the cell, with a blue dot that indicates the location of the synapse.

C. Clicking on a different point in the shape plot moves the synapse to a new location. This change is reflected in the top and bottom panels of the PointProcessManager.

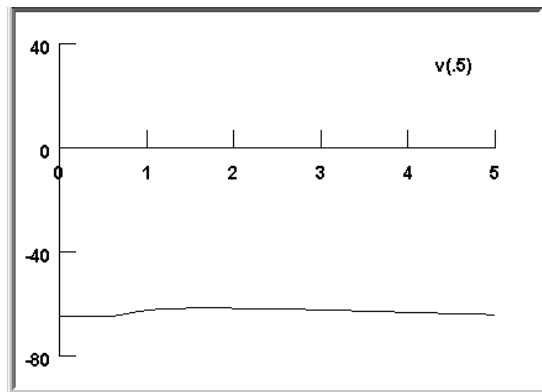


Figure 1.35. Pressing Init & Run starts a new simulation. Even though the synapse is still quite close to the soma, the somatic depolarization is now too small to trigger a spike (space plot not shown).

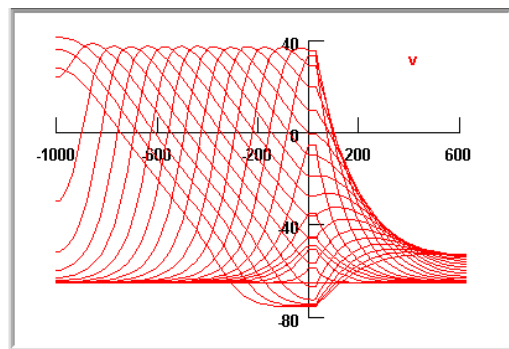
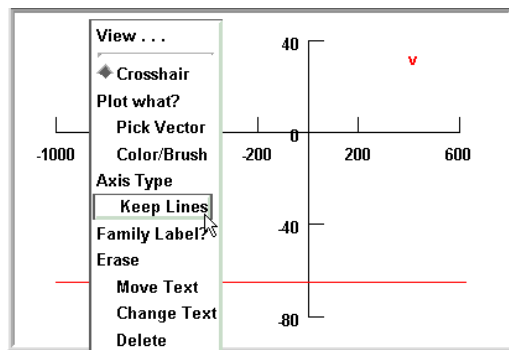


Figure 1.36. A. Activating "keep lines" can help visualize the evolution of V_m more clearly. Right click in the space plot window to bring up its primary menu, then scroll down to Keep Lines and release the mouse button. The next time the primary graph menu is examined, a red check mark will appear next to this item as an indication that keep lines has been toggled *on* (Fig. 1.37 A).

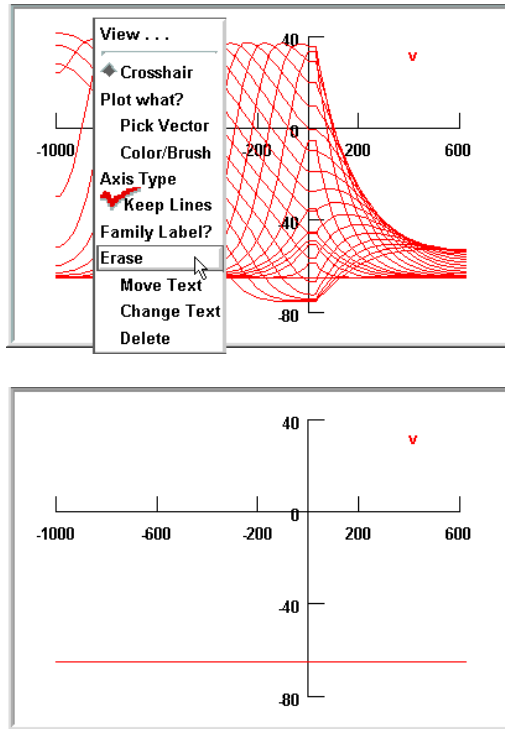
B. To keep the graph from filling up with an opaque tangle of lines, we should make sure the stored traces will be sufficiently different from each other.

Plotting only 5 traces per millisecond will do the trick for this example (leave $dt = 0.025$ ms).

C. Now pressing Init & Run generates a set of traces that facilitate a close examination of the process of excitation and impulse conduction over the model.

For this example the synapse was at the middle of the soma (soma(0.5)). Before running another simulation with a different synaptic location, it would be a good idea to erase these traces (see Fig. 1.37).

Figure 1.37. How to erase traces.



A. Bring up the primary graph menu and scroll down to Erase.

B. The traces will disappear when the mouse button is released. Since keep lines is active, running another simulation will generate a new set of traces.

8. Analyze results

In this section we turn from our specific example to a consideration of the analysis of results. Models are generally constructed either for didactic purposes or as a means for testing a hypothesis. The design and analysis of any model are both strongly dependent on this original motivation, which determines what features are included in the model, what variables are regarded as important enough to measure, and how these measurements are to be interpreted.

While computational models are arguably simpler than any (interesting) experimental preparation, analysis of simulation results presents its own special problems. In the first

place, attempting to use a digital computer to mimic the behavior of a biological system introduces many potential complexities and artifacts. Some arise from the fact that neurons are continuous in space and time, but a digital computer can only generate approximate solutions for a finite number of discrete locations at particular instants. Even so, under the right conditions the approximation can be very good indeed. Furthermore, a well-designed simulation environment can reduce the difficulty of achieving good results.

Other difficulties can arise if there is a mismatch between the expectations of the user and the level of detail that has been included in a model. For example, the most widely used computational model of a conductance change synapse is designed to do the same thing each and every time it is "activated," yet most real synapses display many kinds of use-dependent plasticity, and many also have a high degree of stochastic variability. And even the venerable Hodgkin-Huxley model (Hodgkin and Huxley 1952), which is probably *the* classical success story of computational neuroscience, does not replicate all features of the action potential in the squid giant axon, because it does not completely capture the dynamics of the currents that generate the spike (Moore and Cox 1976; Fohlmeister et al. 1980; Clay and Shlesinger 1982). Such discrepancies are potentially a problem only if a user who is unaware of their existence attempts to apply a model outside of its original context.

The first analysis that is required of all computational modeling is actually the verification that what has been implemented in the computer is a faithful representation of the conceptual model. At the least, this involves checking to be sure that the intended anatomical and biophysical features have been included, that parameters have been assigned the desired values, and that appropriate initialization and integration methods

have been chosen. It may also be necessary to test the model's biophysical mechanisms to ensure that they show the correct dependence on time, membrane potential, ionic concentrations, and modulators. This means understanding the internals of the computational model, which in turn demands a nontrivial grasp of the programming language in which it is expressed. A custom graphical interface that includes well-designed menus and "variable browsers" can make it easier to answer the frequently occurring question "what are the names of things?" Even so, every simulation environment is predicated on a set of underlying concepts and assumptions, and questions inevitably arise that can only be answered on the basis of knowledge of these core concepts and assumptions.

Verification should also involve the qualitative, if not quantitative, comparison of simulation results with basic predictions obtained from experimental observations on biological preparations or generated with prior models. Discrepancies between prediction and simulation are usually caused by trivial errors in model implementation, but sometimes the fault lies in the prediction. Detecting these more interesting outcomes requires practical facility with the simulation environment, so that the level of effort does not obscure one's thinking about the problem.

Agreement between prediction and simulation is reassuring and suggests that the model itself may be useful for generating experimentally-testable predictions. Thus the effort shifts from verifying the model to characterizing its behavior in ways that extend beyond the initial test runs. Both verification and characterization of neural models may entail determining not only membrane potential but also rate functions, levels of modulators, and ionic conductances, currents, and concentrations at one or more locations

in one or more cells. Thus it is necessary to be able to gather and manage measurements, both within a single simulation run and across a family of runs in which one or more independent variables are assigned different values.

Similar concerns arise in connection with optimization, in which one or more parameters are adjusted until the behavior of the model satisfies certain criteria. Optimization also opens a host of new questions whose answers depend in part on the user's judgment, and in part on the resources provided by the simulation environment. Which parameters should remain fixed and which should be adjustable? What constitutes a "run" of the model? What are the criteria for goodness of fit? What constraints, if any, should be imposed on adjustable parameters, and what rules should govern how they are adjusted?

In summary, analysis of results can be the most difficult aspect of any experiment, whether it was performed on living neurons or on a computer model, yet it can also be the most rewarding. The issues raised here are critical to the informed use of any simulation environment, and in the following chapters we will reexamine them in the course of learning how to develop and exercise models with NEURON.

References

Bliss, T.V.P. and Lømo, T. Long-lasting potentiation of synaptic transmission in the dentate area of the anaesthetised rabbit following stimulation of the perforant path. *J. Physiol.* 232:331-356, 1973.

- Castro-Alamancos, M.A. and Connors, B.W. Distinct forms of short-term plasticity at excitatory synapses of hippocampus and neocortex. *Proc. Nat. Acad. Sci.* 94:4161-4166, 1997.
- Clay, J.R. and Shlesinger, M.F. Delayed kinetics of squid axon potassium channels do not always superpose after time translation. *Biophys. J.* 37:677-680, 1982.
- Fohlmeister, J.F., Adelman, W.J.J., and Poppele, R.E. Excitation properties of the squid axon membrane and model systems with current stimulation. Statistical evaluation and comparison. *Biophys. J.* 30:79-97, 1980.
- Hodgkin, A.L. and Huxley, A.F. A quantitative description of membrane current and its application to conduction and excitation in nerve. *J. Physiol.* 117:500-544, 1952.
- Ito, M. Long-term depression. *Ann. Rev. Neurosci.* 12:85-102, 1989.
- Jack, J.J.B., Noble, D., and Tsien, R.W. *Electric Current Flow in Excitable Cells*. London: Oxford University Press, 1983.
- Moore, J.W. and Cox, E.B. A kinetic model for the sodium conductance system in squid axon. *Biophys. J.* 16:171-192, 1976.
- Rall, W. Core conductor theory and cable properties of neurons. In: *Handbook of Physiology, vol. 1, part 1: The Nervous System*, edited by E.R. Kandel. Bethesda, MD: American Physiological Society, 1977, p. 39-98.
- Thomson, A.M. and Deuchars, J. Synaptic interactions in neocortical local circuits: dual intracellular recordings *in vitro*. *Cerebral Cortex* 7:510-522, 1997.

Chapter 1 Index

A

AlphaSynapse 26

parameters 27

analysis 37

approximation 38

assumptions 1

C

CellBuilder 7

bringing up 10

root section 12, 28

CellBuilder GUI

Biophysics page 20

assigning values 22

specifying strategy 21

Continuous Create 24

Geometry page 16

assigning values 18

d_lambda 18

specifying strategy 18

Management page 24

Management page

Export 24

Subsets page 11, 15

all subset 15

making a new subset 16

Topology page 11

base name 14

Basename 14

changing the name of a section 14

making a new section 13

cm 20-22

compartmentalization 11, 19

cytoplasmic resistivity 21

D

d_lambda rule 11

detail

how much 1, 38

diam 16

diameter 4

directory browser 23, 24

discrepancy

 between physical system and conceptual model 38

 between prediction and simulation 38, 39

discretization 19, 38

distributed mechanism 20, 25

dt 32

E

elapsed simulation time 32

F

focus

 cursor 14

G

good programming style

 divide and conquer 7

 modular programming 7

Graph

creating

Shape plot 29

Space Plot 29

Voltage axis 28

Graph GUI

primary menu

Erase 37

Keep Lines 36

H

hh mechanism 22

hoc 3, 24

hypothesis 1, 37

I

initialization 32, 38

initialization

membrane potential 32

instrumentation 7, 25

ion channel 20

ionic conductance 25

J

judgment 1, 40

L

L 16

length 4

length constant 16, 19

M

membrane capacitance 21

membrane potential 27, 28

membrane resistance 22

model

computational 1, 7

analysis 37

model specification 24

conceptual 1, 4, 38

model specification 7, 24

N

NEURON

starting and exiting 8

NEURON Main Menu 8

NEURON Main Menu GUI

Build

CellBuilder 10

File

load session 24

save session 23

Save Session 31

Graph

Shape plot 29

Voltage axis 28

Tools

Point Processes 26

RunControl 31

NEURON program group 8

nrngui 8

numeric integration 38

O

optimization 40

oscilloscope 34

P

parameters

biophysical 21, 22

geometric 18

pas mechanism 21

e_pas 22

g_pas 22

PFWM 22

point process 25

PointProcessManager 25

configuring as

AlphaSynapse 26

creating 26

location 27, 35

location

changing 35

parameters 27

PointProcessManager GUI

SelectPointProcess 26

Show

Shape 35

R

Ra 20, 21

default value 22

RunControl 30

creating 31

RunControl GUI

dt 32

Init 32

Init & Run 32

Points plotted/ms 32

t 32

Tstop 32

running a simulation 34

S

section 11

currently accessed

- default section 28, 30
 - root section 12
- session file 22
- loading
 - from NEURON Main Menu 24
 - from PFWM 22
- saving
 - from NEURON Main Menu 23
 - from PFWM 22
- shape plot 12, 14
- Shape plot
 - creating 29
- Shape plot GUI
 - primary menu
 - Space Plot 29
- signal monitors 27
 - vs. signal sources 27
- signal sources 25
- simulation

running	34
starting	34
time	32
simulation control	7, 30
space	28, 29, 38
space plot	28
Space Plot	
creating	29
squid axon	38
storage oscilloscope	34
synapse	
AlphaSynapse	26
conductance change	26
system equations	
effect of signal sources	27
T	
t	32, 34
time	28, 32, 34, 38
topology	10

U

user interface

as virtual experimental rig 33

custom GUI 31, 39

V

v 28

variable browser 39

variables

independent 40

verification 38

V_m 27

W

what are the names of things? 39