Contents

Preface

I promise nothing complete; because any human thing supposed to be complete, must for that very reason infallibly be faulty.

Who should read this book

This book is about how to use the NEURON simulation environment to build and use empirically-based models of neurons and neural networks. It is written primarily for neuroscience investigators, teachers, and students, but readers with a background in the physical sciences or mathematics who have some knowledge about brain cells and circuits and are interested in computational modeling will also find it helpful. The emphasis is on the most productive use of NEURON as a means for testing hypotheses that are founded on experimental observations, and for exploring ideas that may lead to the design of new experiments. Therefore the book uses a problem-solving approach, with many working examples that readers can try for themselves.

What this book is, and is not, about

Formulating a *conceptual model* is an attempt to capture the essential features that underlie some particular function. This necessarily involves simplification and abstraction of real-world complexities. Even so, one may not necessarily understand all implications of the conceptual model. To evaluate a conceptual model it is often necessary to devise a hypothesis or test in which the behavior of the model is compared against a prediction. *Computational models* are useful for performing such tests. The conceptual model and the hypothesis should determine what is included in a computational model and what is left out. This book is not about how to come up with conceptual models or hypotheses, but instead focuses on how to use NEURON to create and use computational models as a means for evaluating conceptual models.

What to read, and why

The first chapter conveys a basic idea of NEURON's primary domain of application by guiding the reader through the construction and use of a model neuron. This exercise is based entirely on NEURON's GUI, and requires no programming ability or prior experience with NEURON whatsoever.

The second chapter considers the role of computational modeling in neuroscience research from a general perspective. Chapters 3 and 4 focus on aspects of applied mathematics and numerical methods that are particularly relevant to computational neuroscience. Chapter 5 discusses the concepts and strategies that are used in NEURON to simplify the task of representing neurons, which (at least at the level of synapses and cells) are distributed and continuous in space and time, in a digital computer, where neither time nor numeric values are continuous. Chapter 6 returns to the topic of model construction, emphasizing the use of programming.

Chapters 7 and 8 provide "inside information" about NEURON's standard run and initialization systems, so that readers can make best use of their features and customize them to meet special modeling needs. Chapter 9 shows how to use the NMODL programming language to add new biophysical mechanisms to NEURON. This theme continues in Chapter 10, which starts with mechanisms of communication between cells (gap junctions, graded and spike-triggered synaptic transmission), and moves on to models of artificial spiking neurons (e.g. integrate and fire cells). The first half of Chapter 11 is a tutorial on NEURON's GUI tools for creating simple network models, and the second half shows how to use the combined strength of the GUI and `hoc` programming to create more complex networks.

Chapter 12 discusses the elementary features of the `hoc` programming language itself. Chapter 13 describes the object-oriented extensions that have been added to `hoc`. These extensions have greatly facilitated construction of NEURON's GUI tools, and they can also be very helpful in many other complex programming tasks such as creating and managing network models. Chapter 14 presents an example of how to use object oriented programming to increase the functionality of NEURON.

Appendix 1 presents a mathematical analysis of the `IntFire4` artificial spiking cell mechanism, proving a result that is used to achieve computational efficiency. Appendix 2 summarizes the commands for NEURON's built-in text editor.

Typeface conventions

Program listings, names of sections and density mechanisms, classes, objects, methods, procedures, functions, statements, and URLs are printed in a `monospaced` typeface. Optional code, or items that are generic placeholders that the reader should substitute with his or her own specific entries, are indicated by *`slanted monospace`*. Samples of command-line usage employ **`bold monospace`** to signify user input. Labels and menus that appear in NEURON's graphical interface are presented with a sans serif typeface.

Acknowledgments

First and foremost, we want to thank our mentor and colleague John W. Moore for his vision, support, encouragement, and active participation in the development of NEURON, without which neither it nor this book would exist. Through his research and teaching, he was introducing students to ”computational neuroscience” long before that

glorious term was invented. NEURON had its beginnings in John's laboratory at Duke University almost three decades ago, when he and one of the authors (MLH) started their collaboration to develop simulation software for neuroscience research. Users of NEURON on the Macintosh owe John a particular debt. He continues to participate in the development and dissemination of NEURON, concentrating most recently on educational applications in collaboration with Ann Stuart(Moore and Stuart, 2000, 2007).

The list of others who have added in one way or another to the development of NEURON is far too long for this short preface. Zach Mainen, Alain Destexhe, Bill Lytton, Terry Sejnowski, and Gordon Shepherd deserve special mention for many contributions, both direct and indirect, that range from specific enhancements to the program, to fostering the wider acceptance of computational approaches in general, and NEURON in particular, by the neuroscience community at large. We also thank the countless NEURON users whose questions and suggestions continue to help guide the evolution of this software and its documentation. The development of NEURON and this book has been made possible by support from the National Institutes of Health and the National Science Foundation. We are sure that our readers will recognize the epigrams from Herman Melville's *Moby Dick* that are scattered throughout this book, as well as the (mis)quotation from *The Treasure of the Sierra Madre* (book by B. Traven, screenplay by John Huston). We hope that everyone else will forgive any omission and remind us, gently, in time for the second edition.

Finally, we thank our wives and children for their encouragement and patience while we completed this book.

1
A tour of the NEURON simulation environment

. . . so, entering, the first thing I did was to stumble over an ash-box in the porch. Ha! thought I, ha, as the flying particles almost choked me, are these ashes from that destroyed city, Gomorrah?

1.1 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?

1.2 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, well over 1000 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 tens of thousands of neurons (see **Research reports that have used NEURON** at `http://www.neuron.yale.edu/neuron/static/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 perform each of these tasks:

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 (GUI). Later chapters examine `hoc` and the graphical tools for constructing models and managing simulations in more detail, as

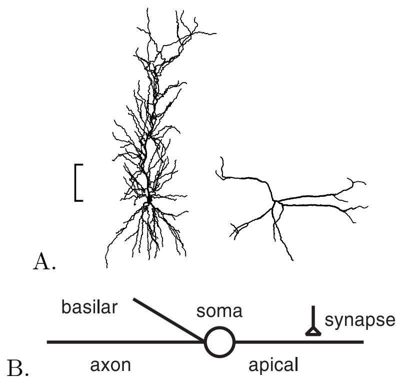


Figure 1.1 A. Two neuronal morphologies obtained from NeuroMorpho.org (`http://neuromorpho.org/`). Ca1 pyramidal cell (left, ri04 from Golding et al. (2005), scale 100 μm) and calretinin-positive interneuron (right, cr20b from Gulyás et al. (1999), scale 50 μm). B. Conceptual model neuron used for the example in this chapter. The synapse can be located anywhere on the cell.

well as many other features and applications of the NEURON simulation environment (e.g. customization of the user interface, complex biophysical mechanisms, neural networks).

1.3 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 many of these from ModelDB (`http://senselab.med.yale.edu/modeldb/`).

1.4 Formulate a conceptual model

Most neurons have many branches with irregularly varying diameters and lengths (Fig. 1.1A), 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. 1.1B. 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. 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 (HH) sodium, potassium, and leak currents, and are capable of generating action potentials (Hodgkin and Huxley, 1952).

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 R_m= 5000 $\Omega\, cm^2$, E_{pas}= -65 mV
Basilar dendrite	200	2	Same as apical dendrite
Axon	1000	1	Same as soma
Cm = $1\mu f/cm^2$, cytoplasmic resistivity = 100 $\Omega\, cm$, temperature = 6.3°C.			

Table 1.2 *Synaptic mechanism parameters*

g_{max}	0.05 μS
τ_s	0.1 ms
E_s	0 mV

A single synaptic activation causes a localized transient conductance increase with a time course described by an alpha function

$$g_s\,(t) = \begin{cases} 0 & t < t_{act} \\ g_{max}\frac{(t-t_{act})}{\tau_s}e^{\frac{(t-t_{act})}{\tau_s}} & t \geq t_{act} \end{cases} \tag{1.1}$$

where t_{act} is the time of synaptic activation, and g_s reaches a peak value of g_{max} at $t = \tau_s$ (Equation 1.1; 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).

1.5 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 **1.6 Instrument the model**).

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

1.5.1 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. 1.2A); 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. 1.2B). If you are using OS X, open the folder where you installed NEURON and double click on the nrngui icon.

You should now see the NEURON Main Menu toolbar (Fig. 1.2C), 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 OS X and MSWindows, the banner and `oc>` prompt will appear in a new terminal window (Fig. 1.2D).

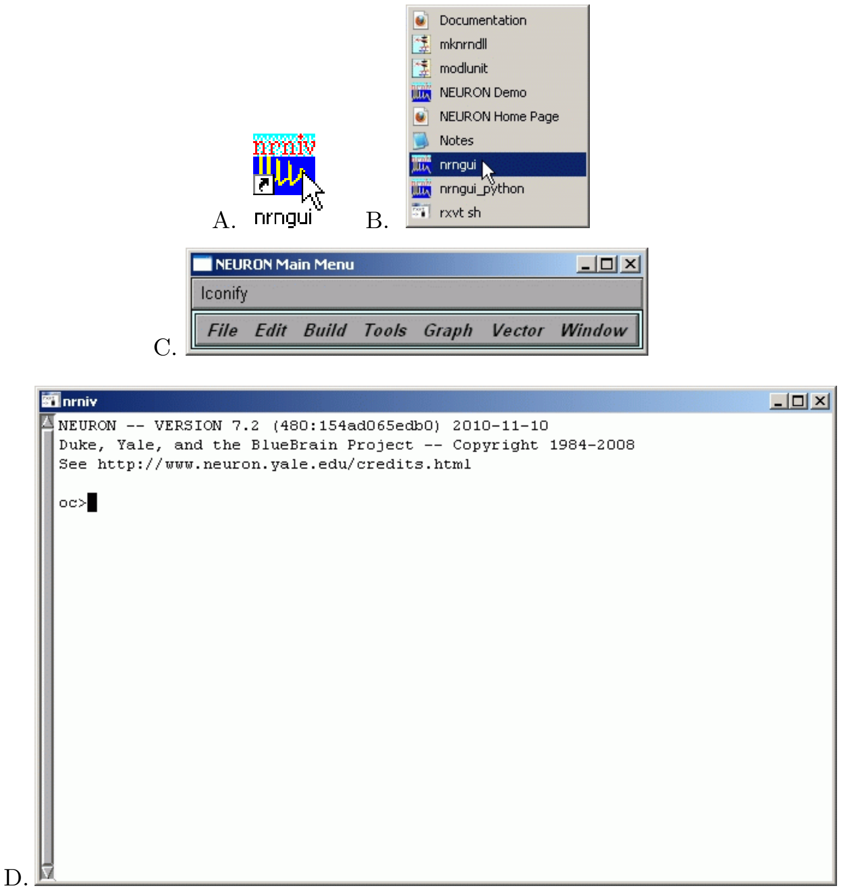
There are three different ways to exit NEURON; use whichever is most convenient:

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. 1.3)

1.5.2 Bringing up a `CellBuilder`

To get a `CellBuilder` just click on Build in the NEURON Main Menu toolbar, scroll down to the CellBuilder item, and release the mouse button (Fig. 1.4A). A `CellBuilder` will appear, displaying its About page which contains some useful hints (Fig. 1.4B).

Across the top of the `CellBuilder` is a row of radio buttons and a checkbox that correspond to the sequence of steps involved in building a model cell. 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 Topology, Subsets, Geometry, and Biophysics pages are used to create



```
NEURON -- VERSION 7.2 (480:154ad065edb0) 2010-11-10
Duke, Yale, and the BlueBrain Project -- Copyright 1984-2008
See http://www.neuron.yale.edu/credits.html

oc>
```

Figure 1.2 A and B. Under MSWindows, it is convenient to start NEURON by clicking on the nrngui icon on the desktop, or by selecting the nrngui item in the NEURON program group. C. Regardless of the operating system, the NEURON Main Menu toolbar looks and works the same. D. NEURON's banner and oc> prompt in an rxvt terminal under MSWindows.

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

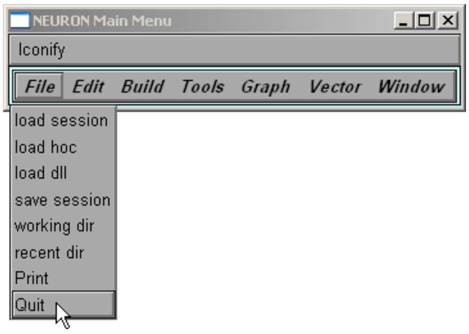


Figure 1.3 One way to exit NEURON is to click on File / Quit in the NEURON Main Menu toolbar.

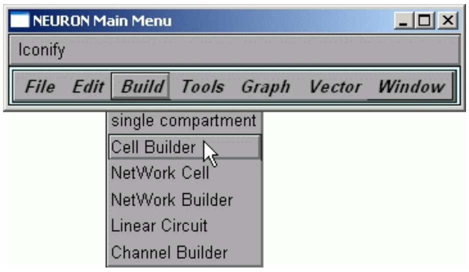
branch, without regard to diameter, length, or biophysical properties. The Subsets page is for grouping sections that share common features. Well-chosen subsets can save a lot of effort later by helping us create very compact specifications of anatomical and biophysical properties. We will deal with 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.

1.5.3 Entering the specifications of the model cell

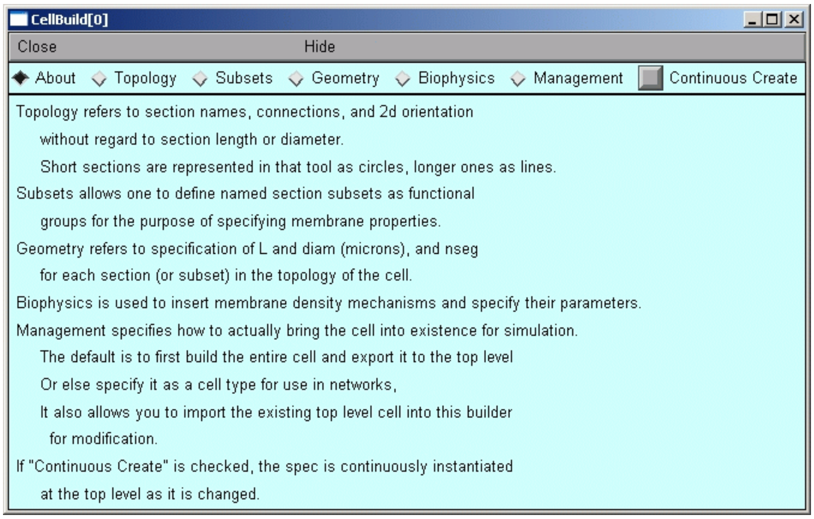
1.5.3.1 Topology

We start by using the Topology page to set up the branched architecture of the model. As Fig. 1.5 shows, when a new `CellBuilder` is created, it already contains a branch (or "section," as it is called in NEURON) that serves 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).

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



A.


CellBuild[0]
Close Hide
About Topology Subsets Geometry Biophysics Management Continuous Create

Topology refers to section names, connections, and 2d orientation
without regard to section length or diameter.
Short sections are represented in that tool as circles, longer ones as lines.
Subsets allows one to define named section subsets as functional
groups for the purpose of specifying membrane properties.
Geometry refers to specification of L and diam (microns), and nseg
for each section (or subset) in the topology of the cell.
Biophysics is used to insert membrane density mechanisms and specify their parameters.
Management specifies how to actually bring the cell into existence for simulation.
The default is to first build the entire cell and export it to the top level
Or else specify it as a cell type for use in networks,
It also allows you to import the existing top level cell into this builder
for modification.
If "Continuous Create" is checked, the spec is continuously instantiated
at the top level as it is changed.

B.

Figure 1.4 A. Using the NEURON Main Menu to bring up a `CellBuilder`. B. The About page of a `CellBuilder` contains some useful hints.

Fig. 1.6. Repeating these actions a couple more times (and resorting to Undo Last, Reposition, and Delete Section as needed to correct mistakes) gives us the basilar dendrite and axon.

Our model cell should now look like Fig. 1.7. At this point some

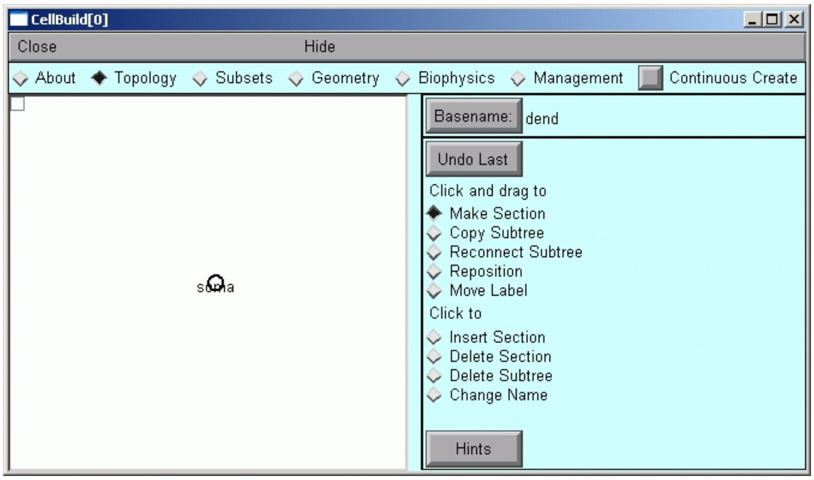
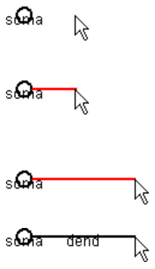


Figure 1.5 The Topology page. The left panel shows a simple diagram of the model (a "shape plot"). The buttons in the right panel are controls for editing the branched architecture of a model cell.



Place the cursor near one end of an existing section.

Click to start a new section. One end of the new section will automatically attach to the nearest end of an existing section; the other end is tethered to the cursor while the mouse button is held down.

Drag to the desired length and orientation.

Release the mouse button.

Figure 1.6 Making a new section. Verify that the Make Section radio button is on, then perform the steps shown above.

minor changes would improve its appearance: moving the labels away from the sections so they are easier to read (Fig. 1.8), and then renaming the apical and basilar dendrites and the axon (Figs. 1.9 and 1.10). The final result should resemble Fig. 1.11.

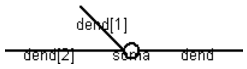
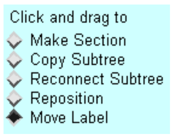
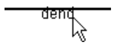


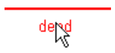
Figure 1.7 The model after all sections have been created.



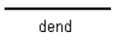
Click on the Move Label radio button,



then click on the label,

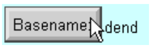


drag it to its new position,

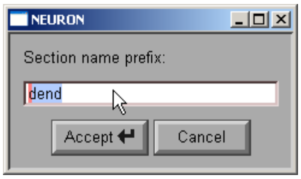


and release the mouse button.

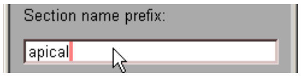
Figure 1.8 How to change the location of a label.



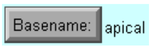
Click the Basename button.



This pops up a Section name prefix window.

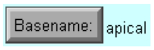


Click inside the text entry field, and type the desired name. It is important to keep the mouse cursor inside the text field while typing; otherwise keyboard entries may not have an effect.

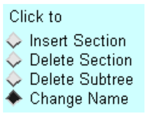


After the new base name is complete, click on the Accept button. This closes the Section name prefix window, and the new base name will appear next to the Basename button.

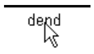
Figure 1.9 Preparing to change the name of a section. Each section we created was automatically given a name based on "dend." To change these names, we must first change the base name as shown here.



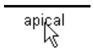
First make sure that the base name is what you want; if not, change the base name (see Fig. 1.9).



Click the Change Name radio button.



Place the mouse cursor over the section whose name is to be changed.



Click the mouse button to change the name of the section.

Figure 1.10 Changing the name of a section.

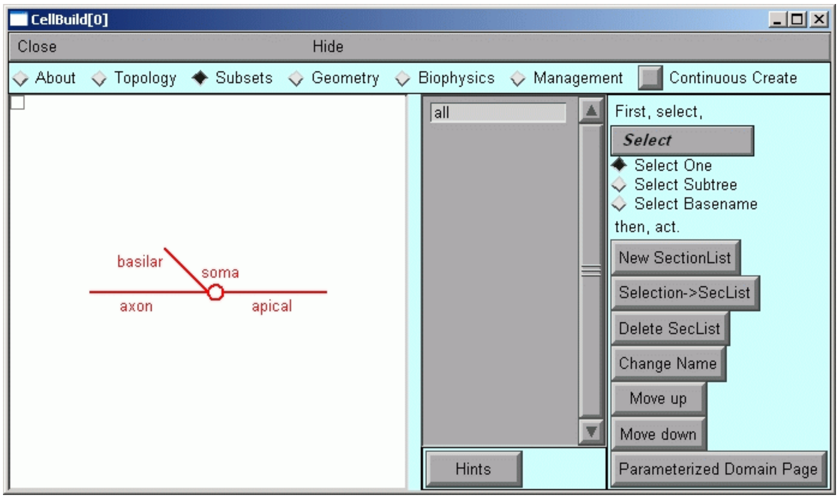


Figure 1.11 The Subsets page. The middle panel lists the names of all existing subsets, and the right panel has controls for managing subsets. In the shape plot, the sections that belong to the currently selected subset are shown in red. When the Subsets page initially appears, it already has an all subset that contains every section in the model.

1.5.3.2 Subsets

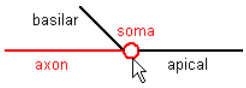
The Subsets page deserves special comment. In almost every model that has more than one branch, two or more branches will have at least some biophysical attributes that are identical, and there are often significant anatomical similarities as well. Furthermore, we can almost always apply the d_lambda rule for compartmentalization throughout the entire cell (see below). Subsets allow us to take advantages of such regularities by assigning shared properties to several branches at once. The Subsets page (Fig. 1.11) 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.

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 (HH) spike currents. To make a subset called has_HH that contains the sections with HH currents, follow the steps in Fig. 1.12. Then make another subset called no_HH that contains the basilar and apical dendrites.

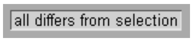
1.5.3.3 Geometry

In order to use the Geometry page (Fig. 1.13) 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.

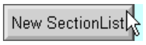
The geometry strategy for our model is simple. Each section has different length `L` and diameter `diam`, so the properties of each section must be entered individually. As far as the spatial resolution of the model is concerned, for this model (and most others) the best choice is to let NEURON automatically determine the spatial discretization of each section based on a fraction of the length constant at 100 Hz (spatial accuracy and NEURON's tools for adjusting the spatial grid



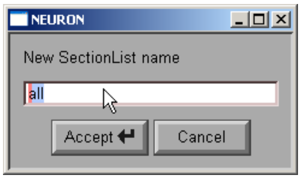
With the Select One radio button on (Fig. 1.11), click on the axon, then hold down the shift key and click on the soma. The selected sections will be indicated in red . . .



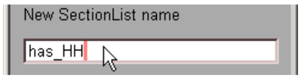
. . . and the list of subsets will change to show that all is not the same as the set {axon, soma}.



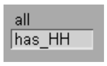
Next, click on the New SectionList button (a subset is a list of sections).



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`.

Figure 1.12 Making a new subset.

are discussed in **Chapter 5**). Figure 1.14 shows how to set up this strategy.

Having set up the strategy, we are ready to assign the geometry parameters (Fig. 1.15). At the top of the list is `d_lambda`. By specifying that spatial discretization will follow the d_lambda rule, we have

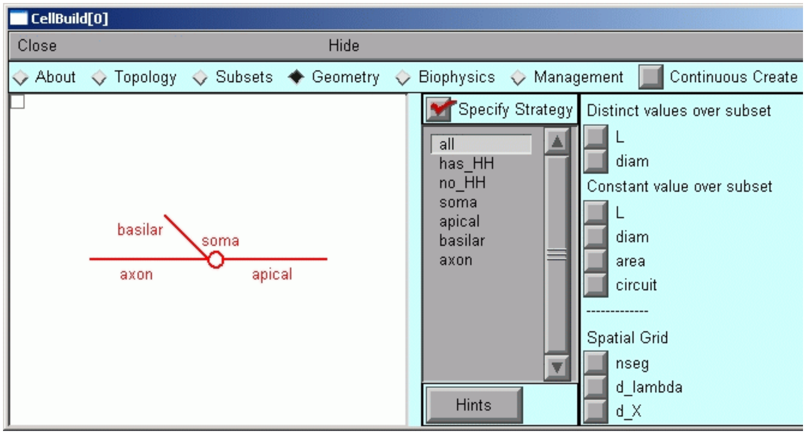
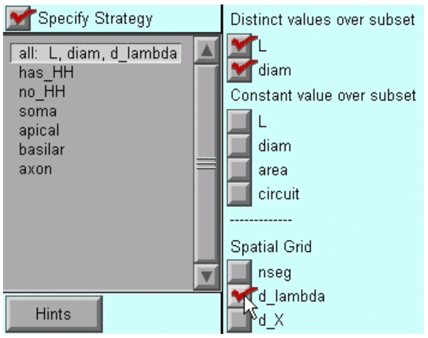


Figure 1.13 When the Geometry page in a new CellBuilder is first viewed, a red check mark should appear in the Specify Strategy checkbox. If not, click on the checkbox to toggle Specify Strategy on.

stipulated that NEURON will automatically discretize the model, breaking each section into just enough compartments so that none is longer than d_lambda times the AC length constant 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.

Scrolling through the other geometry parameters reveals that the default values of many lengths and diameters differ from our desired specification (cf. Table 1.1). Figure 1.16 shows how to change soma length to 30 μm. After all revisions have been made, the geometry parameters should look like Fig. 1.17. Note the x in the middle panel, which signifies that one or more parameters associated with the adjacent section or subset name have been altered. In the right panel, note the red marks in checkboxes; these identify the parameters that have been changed from their default values.


✓ Specify Strategy

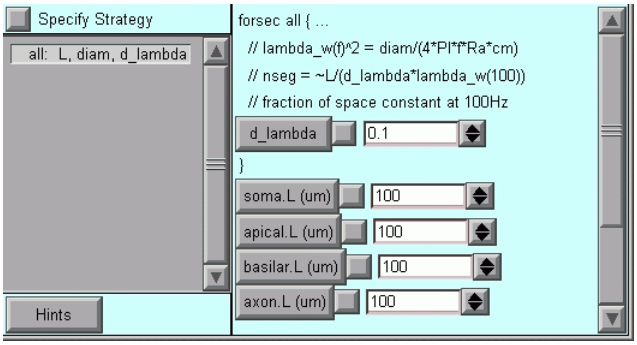
all: L, diam, d_lambda
has_HH
no_HH
soma
apical
basilar
axon

Distinct values over subset
✓ L
✓ diam

Constant value over subset
L
diam
area
circuit

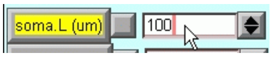
Spatial Grid
nseg
✓ d_lambda
d_X

Figure 1.14 Specifying strategy for assignment of geometry parameters. First make sure that Specify Strategy contains a red check (see Fig. 1.13). For models in which the geometry of each section is unique, like this one, click on the L and diam boxes under "Distinct values over subset." To allow NEURON to take care of spatial discretization automatically, click on the d_lambda box under "Spatial Grid."

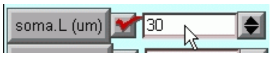


```
forsec all { ...
  // lambda_w(f)^2 = diam/(4*PI*f*Ra*cm)
  // nseg = ~L/(d_lambda*lambda_w(100))
  // fraction of space constant at 100Hz
  d_lambda 0.1
}
soma.L (um) 100
apical.L (um) 100
basilar.L (um) 100
axon.L (um) 100
```

Figure 1.15 Assigning values to the geometry parameters. Toggling Specify Strategy off makes the middle panel show the subsets and sections that we selected when setting up our strategy. Our strategy is based entirely on the all subset, so all is the only item that appears in the middle panel. The buttons in the right panel display the names of the sections and parameters that are associated with the all subset, and offer us the means to change parameters as necessary. The vertical scroll bar along the right edge indicates that some buttons fall below the bottom edge of the `CellBuilder`'s window; these can be revealed by dragging the scroll bar up or down.

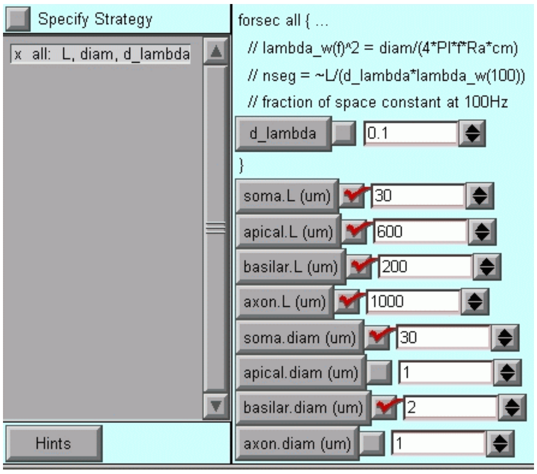


To set the length of the soma to 30 μm, first click inside the numeric field for soma.L so that a red editing cursor appears.



Then use the backspace key to delete the old value, type in the new value, and press "Enter" on your keyboard. The red mark in the checkbox indicates that this parameter has been changed from its default value.

Figure 1.16 Assigning values to the geometry parameters *continued.*



```
x  all:  L, diam, d_lambda
```

```
forsec all { ...
  // lambda_w(f)^2 = diam/(4*PI*f*Ra*cm)
  // nseg = ~L/(d_lambda*lambda_w(100))
  // fraction of space constant at 100Hz
  d_lambda  0.1
}
soma.L (um)      ✓ 30
apical.L (um)    ✓ 600
basilar.L (um)   ✓ 200
axon.L (um)      ✓ 1000
soma.diam (um)   ✓ 30
apical.diam (um)   1
basilar.diam (um)✓ 2
axon.diam (um)     1
```

Figure 1.17 The revised geometry parameters. The x in the middle panel and the red marks in the right panel signal changes from default values.

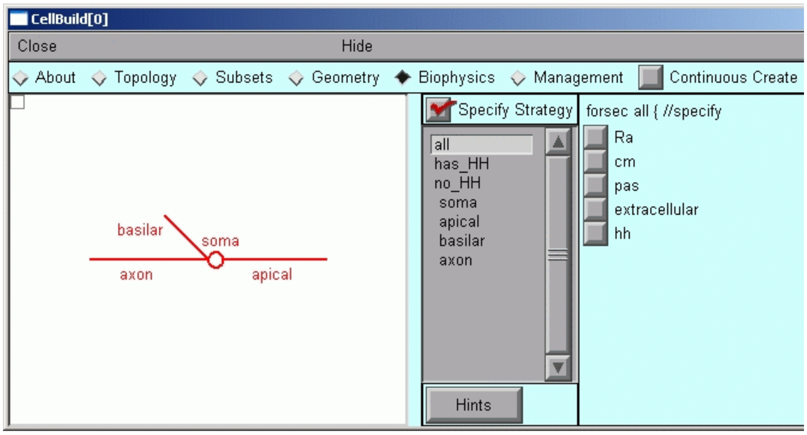
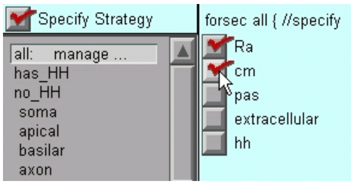
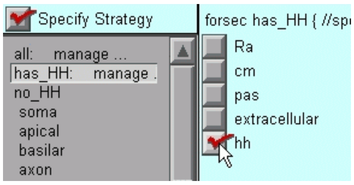


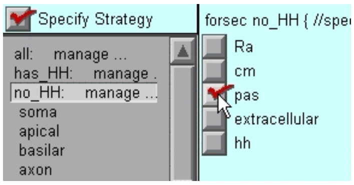
Figure 1.18 The Biophysics page, ready for specification of strategy. The right panel shows the mechanisms that are available to be inserted into our model. For this simple example, the number of mechanisms is deliberately small; adding new mechanisms is covered in **Chapter 9**.



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.

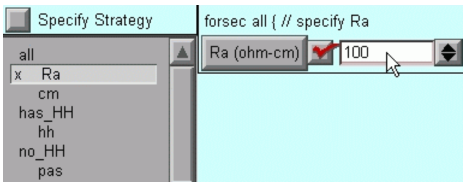


Finally select the no_HH subset and toggle pas on.

Figure 1.19 Specifying strategy for assignment of biophysical parameters. First make sure that Specify Strategy contains a red check, then proceed with the steps described above.

1.5.3.4 Biophysics

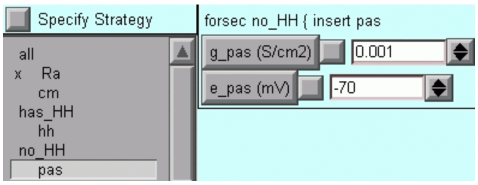
The Biophysics page (Fig. 1.18) is used to specify the biophysical properties of membrane and cytoplasm (e.g. Ra, Cm, ion channels, buffers, pumps) for subsets and individual sections. As with the Geometry page, first we set up a strategy, then we review and adjust parameter values. After we have applied the steps outlined in Figs. 1.19 and 1.20, the `CellBuilder` will contain a complete specification of our model.



forsec all { // specify Ra

Ra (ohm-cm) 100

For the all subset, change Ra from its default 35.4 $\Omega\, cm$ to the desired 100 $\Omega\, cm$.

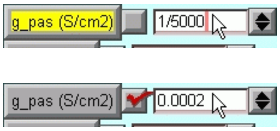


forsec no_HH { insert pas

g_pas (S/cm2) 0.001

e_pas (mV) -70

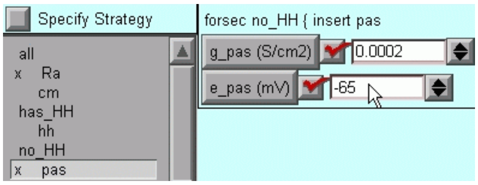
The sections in the no_HH subset have a passive current whose parameters must be changed from their defaults (shown here).



g_pas (S/cm2) 1/5000

g_pas (S/cm2) 0.0002

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



forsec no_HH { insert pas

g_pas (S/cm2) 0.0002

e_pas (mV) -65

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.

Figure 1.20 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 name brings up controls in the right panel for displaying and adjusting its parameters.

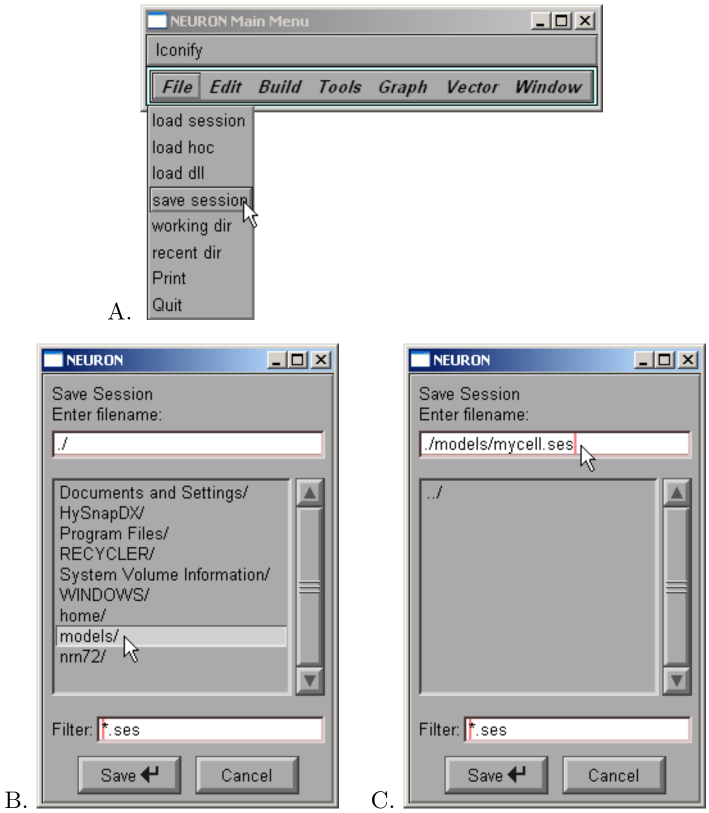
1.5.4 Saving the model cell

Having invested time and effort to set up our model, we would be wise to take a moment to save it. The `CellBuilder`, like NEURON's other graphical tools, can be saved to disk as a "session file" for future re-use, as shown in Figs. 1.21 and 1.22. 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/static/docs/saveses/saveses.html`

1.5.5 Executing 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. 1.23). 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.

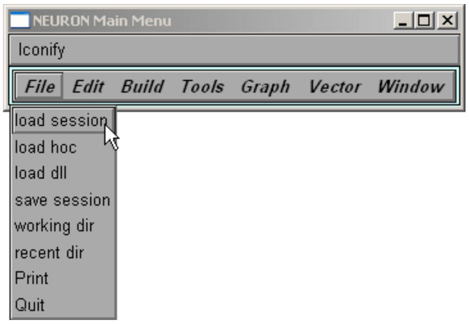
If we make any changes to the model while Continuous Create is on, the `CellBuilder` will automatically send new code to the interpreter. This can be very convenient during model development, since it allows us to quickly examine the effects of any change. Automatic updates might bog things down if we 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.



Save Session
Enter filename: ./
Filter: *.ses

Save Session
Enter filename: ./models/mycell.ses
Filter: *.ses

Figure 1.21 A. To save all of NEURON's graphical windows to a session file, first click on File in the NEURON Main Menu and scroll down to save session. B. This brings up a directory browser that can be used to navigate to the directory where the session file will be saved. C. 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.

A.


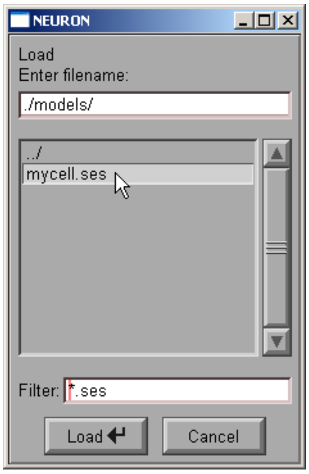
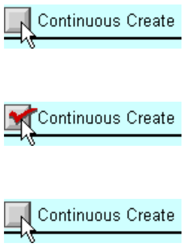
B.


Figure 1.22 A. 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. B. 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.



Continuous Create is initially off,

but clicking on the checkbox toggles it on

and off.

Figure 1.23 Using Continuous Create.

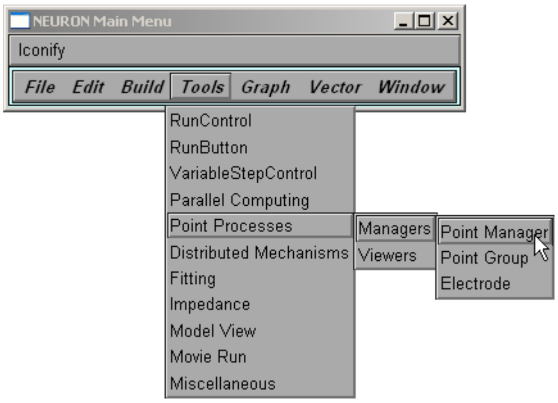
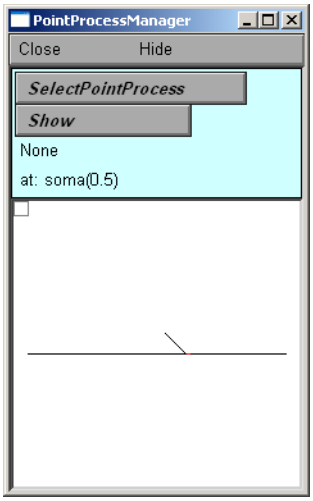


Figure 1.24 Bringing up a `PointProcessManager` in order to attach a synapse to our model cell. In the NEURON Main Menu, click on Tools / Point Processes / Managers / Point Manager, then proceed as shown in Fig. 1.25

A.


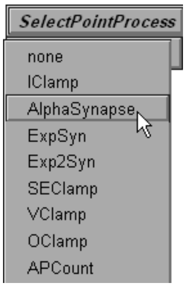
B.


Figure 1.25 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.

1.6 Instrument the model

1.6.1 Signal sources

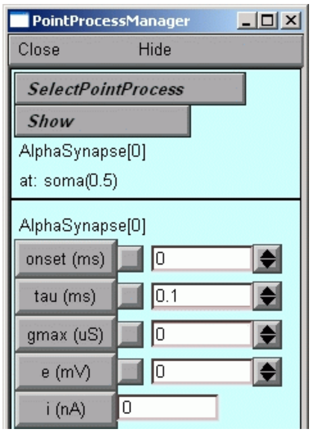
In the NEURON simulation environment, a synapse or electrode for passing current (current clamp or voltage clamp) is represented by a point source of current which is associated with a localized conductance. Such localized signal sources are called ”point processes” to distinguish them from properties 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. 1.24). Using a `PointProcessManager` we can specify the type and parameters of the point process (Fig. 1.25) and where it is attached to the cell.

1.6.2 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. 1.27).

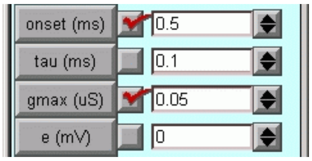
We could monitor V_m at other locations by adding more variables to this graph, and bring up additional graphs if this one became too crowded. However, it can be more informative and convenient to



AlphaSynapse[0]
at: soma(0.5)

AlphaSynapse[0]
onset (ms) 0
tau (ms) 0.1
gmax (uS) 0
e (mV) 0
i (nA) 0

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 `tau` (t_{act} and τ_s in Eq. 1.1), peak conductance `gmax` (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.

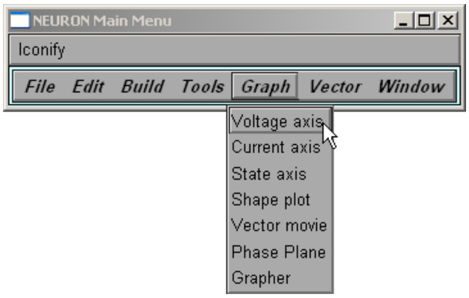


onset (ms) ✓ 0.5
tau (ms) 0.1
gmax (uS) ✓ 0.05
e (mV) 0

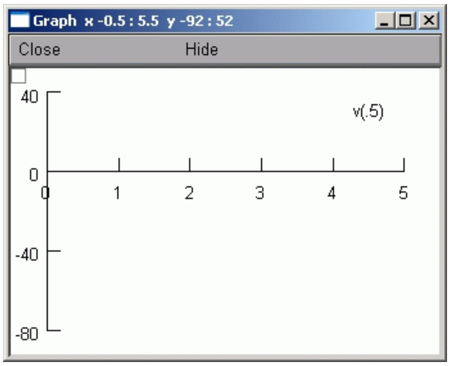
For this example change `onset` to 0.5 ms and `gmax` to 0.05 μS; leave `tau` and `e` unchanged.

Figure 1.26 Specifying the properties of an `AlphaSynapse`.

create a ”space plot”. A space plot is a `Graph` in which a variable is plotted as a function of distance along one or more branches of a cell. Figures 1.28-1.30 show how to set up a space plot of membrane potential that can change throughout a simulation, displaying the evolution of V_m as a function of space and time.



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



The horizontal axis of a "voltage axis graph" 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, (for this model, the soma). The concepts of root section and default section are discussed in **Chapter 5**.

Figure 1.27 Creating a graph to display somatic membrane potential as a function of time.

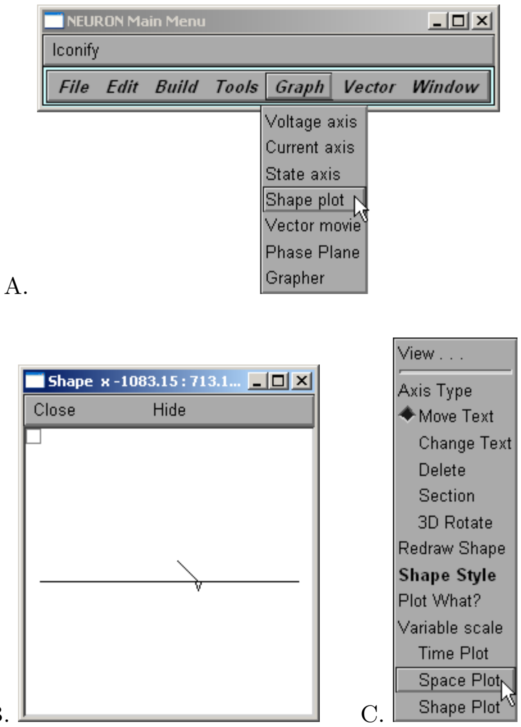
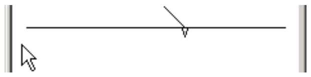
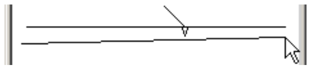


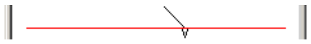
Figure 1.28 The first step in setting up a space plot is to create a `Shape` object, which is used to specify the space plot's path. A. To create a `Shape`, click on Graph / Shape plot in the NEURON Main Menu. B. This brings up a `Shape` window, which can be used to set up graphs of a range variable–membrane potential (v) in this case–vs. time or distance. C. Click on the menu box in the `Shape` window to bring up its primary menu. While still depressing the mouse button, scroll down the menu to the Space Plot item, then release the button. The `Shape` window is now ready to be used to specify the space plot's paths.



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



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.



The branches along the selected path (axon, soma, and apical dendrite) are now shown in red, and a new window pops up that shows a space plot of v along this path (see Fig. 1.30). At this point, you may click on the Close button at the upper left corner of the `Shape` window to conserve screen space.

Figure 1.29 Specifying a space plot's paths.

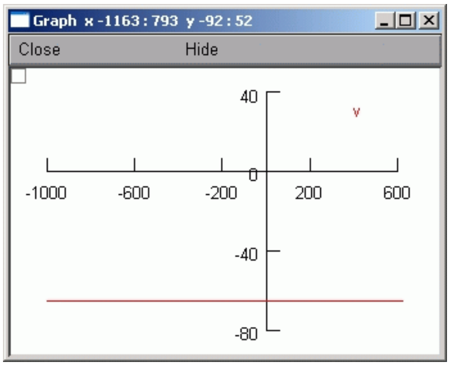
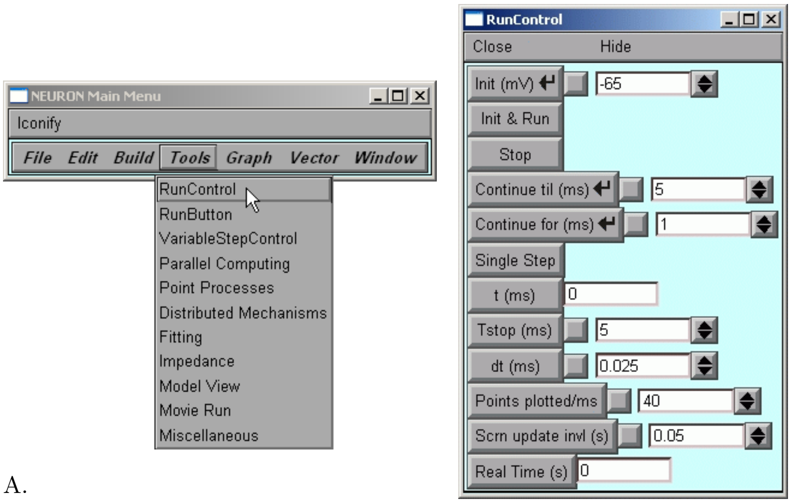


Figure 1.30 The space plot of membrane potential created by the steps shown in Figs. 1.28 and 1.29. The x axis shows the distance from the 0 end of the default section, which in this example is the left end of the soma.



Init (mV) -65
Continue til (ms) 5
Continue for (ms) 1
t (ms) 0
Tstop (ms) 5
dt (ms) 0.025
Points plotted/ms 40
Scrn update invl (s) 0.05
Real Time (s) 0

A.

B.

Figure 1.31 A. To bring up a window with controls for running simulations, click on the RunControl button in NEURON Main Menu / Tools. B. The `RunControl` provides many options for controlling the overall time course of a simulation run.

1.7 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 a `RunControl` panel (Fig. 1.31), which offers a great deal of control over simulations.

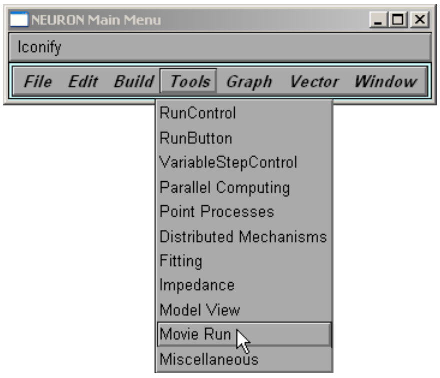
Of the many options that this tool allows us to specify, these three are most relevant to this example:

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 their steady state values at this potential.

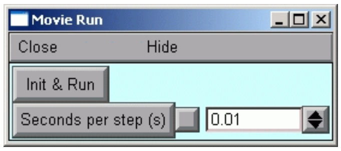
2. Init & Run performs the same initialization as Init (mV), and then starts a simulation run.
3. Points plotted/ms determines how often the graphical displays are updated during a simulation.

Three other items in this panel are of obvious interest, although we will not do anything with them for now. 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` panel are discussed in **Chapter 7**.

The last item to add to our user interface is a Movie Run tool, as shown in Fig. 1.32. We will use this tool to launch simulations in which the space plot of membrane potential evolves smoothly.



A.



B.

Figure 1.32 A. To bring up a tool for running simulations that update space plots smoothly, click on the Movie Run button in NEURON Main Menu / Tools. B. Clicking on the Init & Run button in the Movie Run tool starts a simulation in which space plots are refreshed at intervals specified by the value shown in the the Seconds per step field.

1.8 Save model with instrumentation and run control

After some rearrangement, our customized user interface for running simulations and observing simulation results should look something like Fig. 1.33. 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.

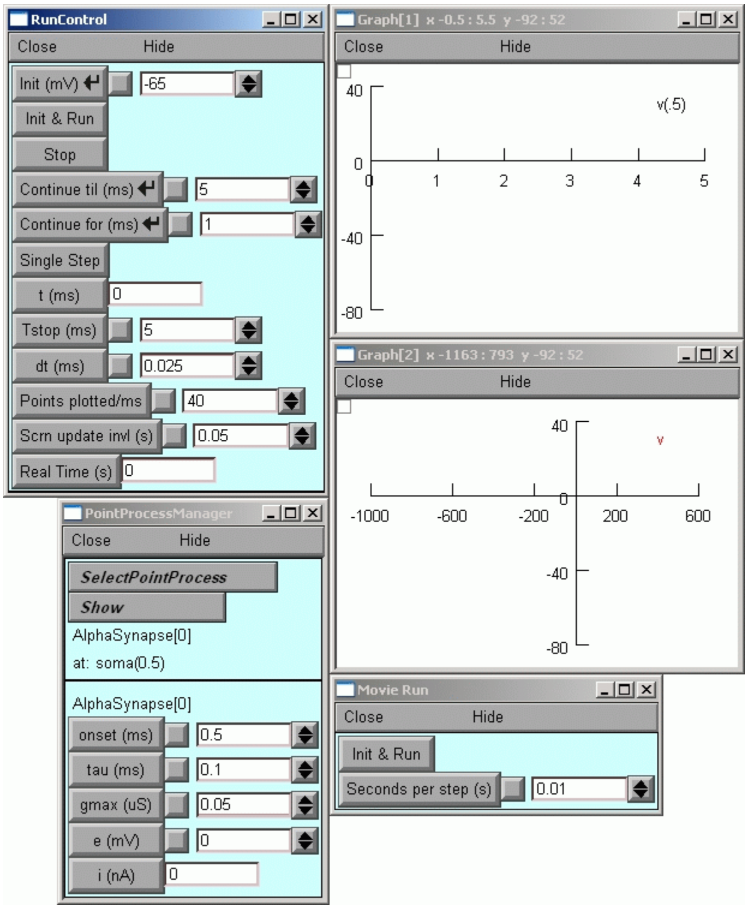
1.9 Run the simulation experiment

We are now ready to use our ”virtual experimental rig” to exercise the model. Clicking on the Init & Run button in the `RunControl` panel (Fig. 1.34) launches a simulation, and the graph of somatic membrane potential vs. time shows that the synaptic input triggers a spike at that location (Fig. 1.35).

To examine how V_m evolves throughout the cell, let us now turn to the space plot. If a simulation is launched with `RunControl`'s Init & Run button, NEURON takes the time-saving shortcut of deferring shape plot updates until the end of the run. Consequently our shape plot only shows the distribution of V_m at the end of the simulation. In order to see the shape plot evolve over the course of the simulation, it is necessary to click on the Movie Run tool's Init & Run button. This makes NEURON update the shape plot at each new time step, and reveals how the spike starts at the soma and spreads out to the axon and apical dendrite (Fig. 1.36).

The utility of the space plot as a tool for understanding the spatiotemporal evolution of a variable can be enhanced by using it like a storage oscilloscope (Figs. 1.37 and 1.38; also see Fig. 1.39 for how to erase “stored” traces). This can make it easier to evaluate and compare the spatial distribution of variables at successive intervals during a run.

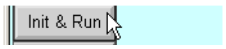
NEURON's GUI greatly simplifies the task of constructing and using models. In particular, the GUI makes it easy to perform experi-



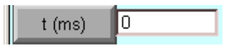
```
Init (mV)            -65
Continue til (ms)    5
Continue for (ms)    1
t (ms)               0
Tstop (ms)           5
dt (ms)              0.025
Points plotted/ms    40
Scrn update invl (s) 0.05
Real Time (s)        0

AlphaSynapse[0]
at: soma(0.5)

AlphaSynapse[0]
onset (ms)   0.5
tau (ms)     0.1
gmax (uS)    0.05
e (mV)       0
i (nA)       0

Seconds per step (s) 0.01
```

Figure 1.33 Our user interface for running simulations and observing results. Other windows that are present on the screen but not shown in this figure are the NEURON Main Menu and the `CellBuilder`.

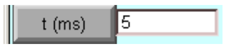
mental manipulations of a model and see what happens. For example, we can explore the effect of changing the location of the synaptic input. If we move the synapse even a small distance away from the



Press Init & Run in the `RunControl` panel to launch a simulation.



This makes time `t` advance from 0 . . .



. . . to 5 ms in 0.025 ms increments.

Figure 1.34 Running a simulation. The response to an excitatory synaptic input at the soma is shown in Figs. 1.35, 1.36, and 1.38.

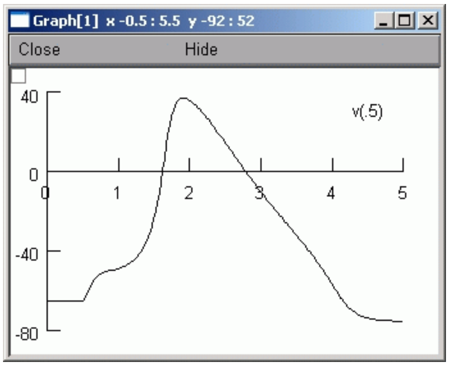


Figure 1.35 The excitatory synapse at the soma elicits a somatic spike.

soma along the apical dendrite (Fig. 1.40) and run a new simulation, the epsp at the soma is too small to evoke a spike (Fig. 1.41).

1.10 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

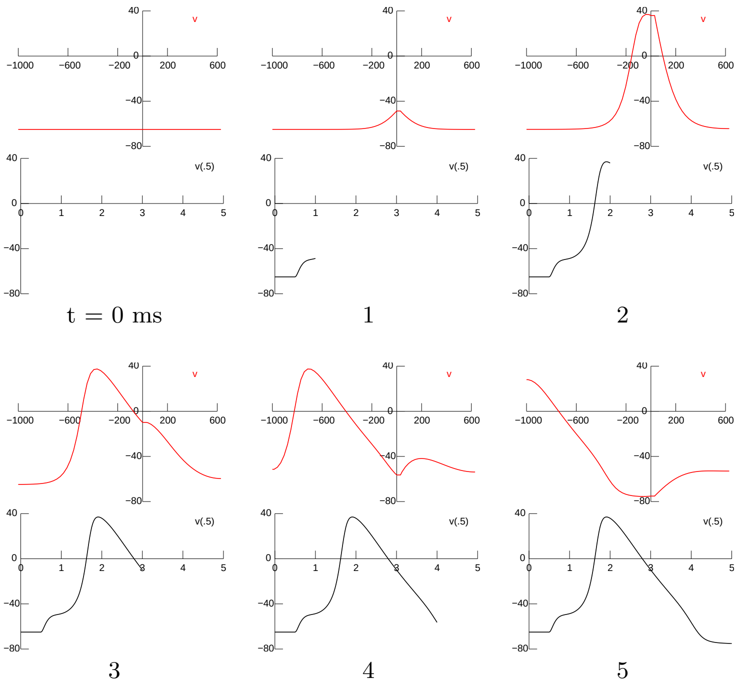
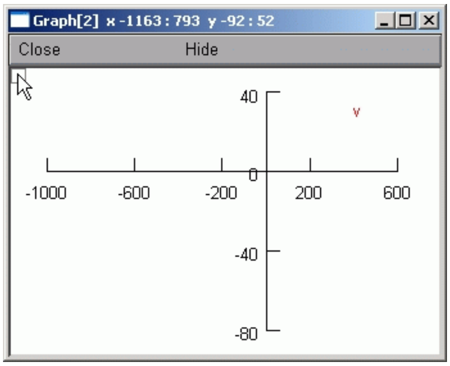


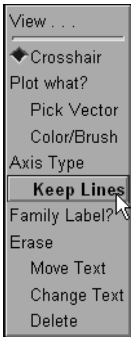
Figure 1.36 Snapshots of simulation results taken at 1 ms intervals. Each pair of graphs shows V_m vs. distance and V_m at the soma (`v(.5)`) vs. `t`. Synaptic input at the soma triggers a spike that propagates actively along the axon and spreads with passive decrement into the apical dendrite.

didactic purposes or as a means for testing a hypothesis. Both the design and analysis of any model are 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.

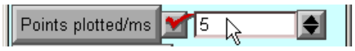
While computational models are arguably simpler than any (interesting) experimental preparation, analysis of simulation results presents its own special problems. In the first place, attempting to use



Bring up the space plot's primary menu by placing the mouse cursor in the menu box (square in upper left corner) and pressing on the mouse button.



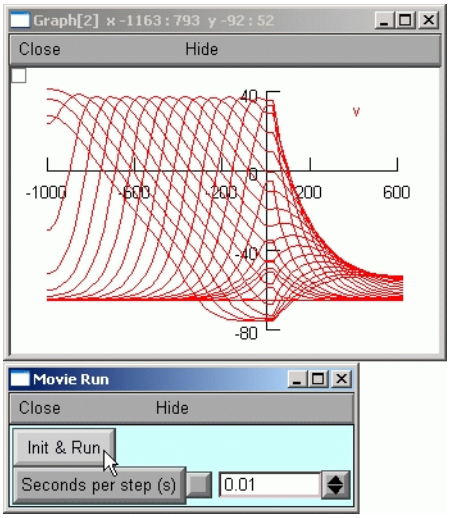
While still depressing the button, scroll the cursor down to Keep Lines, then release the mouse button. The next time the primary graph menu is examined, a red check mark will appear next to this item, indicating that keep lines has been toggled on (e.g. see Fig. 1.39).



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 is sufficient for this example (leave dt = 0.025 ms).

Figure 1.37 Preparing to capture "multiple exposures" of the spatial distribution of V_m.

a digital computer to mimic the behavior of a biological system introduces many potential complexities and artifacts. Some arise from



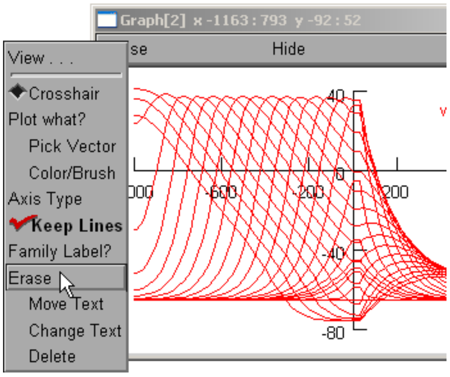
Now clicking on Init & Run in the Movie Run tool generates a set of traces that facilitate examination of impulse initiation and propagation through the model.

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. 1.39).

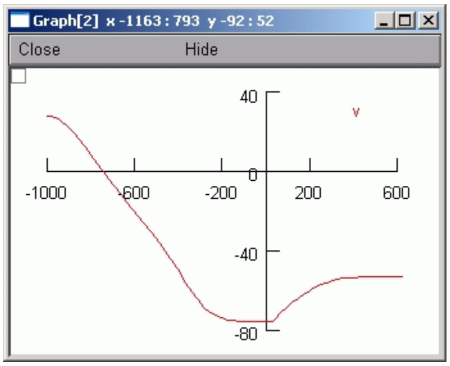
Figure 1.38 Capturing "multiple exposures" of the spatial distribution of V_m.

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 (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



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

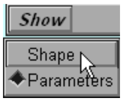


When the mouse button is released, all traces vanish but one: the trace that that shows the current values of V_m along the path.

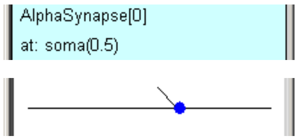
Figure 1.39 How to erase traces.

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.

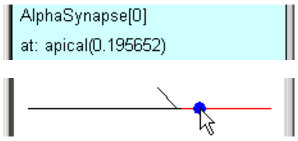
The first analysis that is required of all computational modeling is actually the verification that what has been implemented in the computer is a faithful representation of the conceptual model. At the least, this involves checking to be sure that the intended anatom-



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



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.



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 1.40 Changing synaptic location.

ical 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

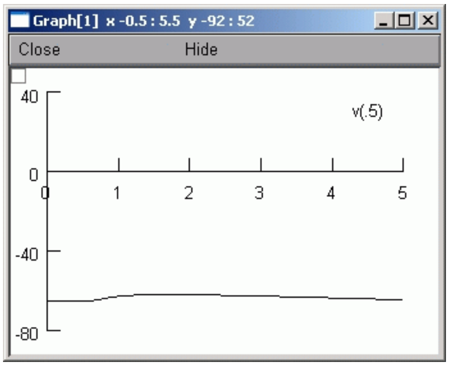


Figure 1.41 Pressing Init & Run starts a new simulation. Even though the synapse is still quite close to the soma, the somatic depolarization is now too small to trigger a spike (space plot not shown).

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.

2
The modeling perspective

. . . can you not tell water from air? My dear sir, in this world it is not so easy to settle these plain things. I have ever found your plain things the knottiest of all.

This chapter and the next deal with concepts that are not NEURON-specific but instead pertain equally well to *any* tools used for neural modeling.

2.1 Why model?

In order to achieve the ultimate goal of understanding how nervous systems work, it will be necessary to know many different kinds of information:

- the anatomy of individual neurons and classes of cells, pathways, nuclei, and higher levels of organization
- the pharmacology of ion channels, transmitters, modulators, and receptors
- the biochemistry and molecular biology of enzymes, growth factors, and genes that participate in brain development and maintenance, perception and behavior, learning and forgetting, health and disease

But while this knowledge will be necessary for an understanding of brain function, it isn't sufficient. This is because the moment-to-moment processing of information in the brain is carried out by the spread and interaction of electrical and chemical signals that are distributed in space and time. These signals are generated and regulated by mechanisms that are kinetically complex, highly nonlinear, and arranged in intricate anatomical structures. Hypotheses about these signals and mechanisms, and how nervous system function emerges from their operation, cannot be evaluated by intuition alone, but require empirically-based modeling. From this perspective, modeling is fundamentally a means for enhancing insight, and a simulation environment is useful to the extent that it maximizes the ratio of insight obtained to effort invested.

2.2 From physical system to computational model

Just what is involved in creating a computational model of a physical system?

2.2.1 Conceptual model: a simplified representation of a physical system

The first step is to formulate a *conceptual model* that attempts to capture just the essential features that underlie a particular function or property of the physical system. If the aim of modeling is to provide insight, then formulating the conceptual model necessarily involves simplification and abstraction (Fig. 2.1 left). When a physical system is already simple enough to understand, there is no point in further simplification because we won't learn anything new. If instead the system is complex, a conceptual model that omits excess detail can foster understanding.

But some models contain essential irreducible complexities, and even conceptual models that are superficially simple can resist intuition. To evaluate such a model it is often necessary to devise a hypothesis or test in which the behavior of the model is compared against a prediction. *Computational models* are useful for performing such tests. The conceptual model, and the hypothesis behind it, determine what is included in the computational model and what is left out.

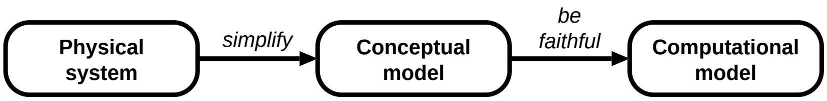


Figure 2.1 Creating a computational model of a physical system involves two steps. The first step deliberately omits real-world complexities to produce a conceptual model. In the second step, this conceptual model must be faithfully translated into a computational model, without any further subtractions or additions.

When we formalize our description of a biological system, the first language we use is mathematics. The conceptual model is usually expressed in mathematical form, although there are occasions when it is more convenient to express the concept in the form of a computer algorithm. **Chapter 3** is concerned with mathematical representations of chemical and electrical phenomena relevant to signaling in neurons.

2.2.2 Computational model: an accurate representation of a conceptual model

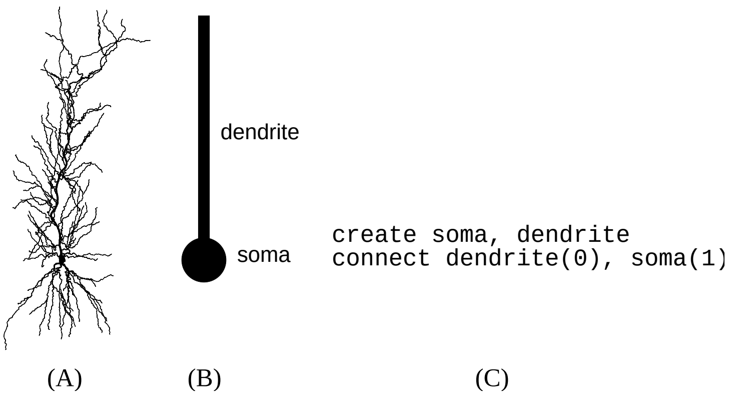
A computational model is a working embodiment of a conceptual model through the medium of computer simulation. It can assist hypothesis testing by serving as a virtual laboratory preparation in which the functional consequences of the hypothesis can be examined. Such tests can be valid only if the computational model is as faithful to the conceptual model as possible. This means that the computational model must be implemented in a way that does not impose additional simplifications or introduce new properties that were not consciously chosen by the user; otherwise how can the user tell whether simulation results truly reflect the properties of the conceptual model, and are not a byproduct of distortions produced by trying to implement the model with a computer? This ideal is impossible to achieve, and the proper use of any simulator requires judgment by the user as to whether discrepancies between concept and concrete representation are benign or vicious.

A useful simulation environment enables experimental tests of hypotheses by facilitating the construction, use, and revision of computational models that are faithful to the original idea and its subsequent evolution. NEURON is designed to meet this goal, and one of the aims of this book is to show you how to tell whether the model you have in mind is matched by the NEURON simulation you create.

2.2.3 An example

Figure 2.2A shows the side view of a Ca1 pyramidal neuron. Suppose we are interested in how this cell responds to current injected at the soma. We could imagine an enormously complicated conceptual model that attempts to mimic all of the detail of the physical system. But if we are trying to gain insight into the charging properties of the cell as observed at the soma, we might start with a much simpler conceptual model, like the ball and stick shown in Fig. 2.2B. Most of the anatomical complexity of the physical system lies in the dendritic tree, but our conceptual model approximates the entire dendritic tree by a very simple abstraction: a cylindrical cable.

So going from the physical system to the model involved simplification and abstraction. What about going from the conceptual model to a computational model? The statements in Fig. 2.2C specify the topology of the computational model `hoc`, NEURON's built-in programming language. Note that everything in the conceptual model has a direct counterpart in this computational implementation, and vice versa: the transition between concept and computational model involves neither simplification nor additional complexity. All that remains is to assign physical dimensions and biophysical properties, and the computational model can be used to generate simulations that reflect the behavior of the conceptual model.



```
create soma, dendrite
connect dendrite(0), soma(1)
```

Figure 2.2 A. Side view of Ca1 pyramidal neuron (ri04 from Golding et al. (2005), data available at `http://neuromorpho.org/`). B. "Ball and stick" conceptual model. C. Computational implementation of the conceptual model.

References

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

Brown, P.N., Hindmarsh, A.C., and Petzold, L.R. 1994. Using Krylov methods in the solution of large-scale differential-algebraic systems. *SIAM Journal of Scientific Computing*, **15**, 1467–1488.

Carslaw, H.S., and Jaeger, J.C. 1980. *Conduction of Heat in Solids*. 2 edn. Oxford: Oxford University Press.

Castro-Alamancos, M.A., and Connors, B.W. 1997. Distinct forms of short-term plasticity at excitatory synapses of hippocampus and neocortex. *Proceedings of the National Academy of Sciences of the United States of America*, **94**, 4161–4166.

Clay, J.R., and Shlesinger, M.F. 1982. Delayed kinetics of squid axon potassium channels do not always superpose after time translation. *Biophysical Journal*, **37**, 677–680.

Cohen, S.D., and Hindmarsh, A.C. 1994. *CVODE User Guide*. Tech. rept. URCL-MA-118618. Lawrence Livermore National Laboratory.

Cohen, S.D., and Hindmarsh, A.C. 1996. CVODE, a stiff/nonstiff ODE solver in C. *Computers in Physics*, **10**, 138–143.

Crank, J. 1979. *The Mathematics of Diffusion*. 2 edn. London: Oxford University Press.

Crank, J., and Nicholson, P. 1947. A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type. *Proceedings of the Cambridge Philosophical Society*, **43**, 50–67.

Dahlquist, G., and Bjorck, A. 1974. *Numerical Methods*. Englewood Cliffs, New Jersey: Prentice-Hall.

Fohlmeister, J.F., Adelman, W.J.Jr., and Poppele, R.E. 1980. Excitation properties of the squid axon membrane and model systems with current stimulation. *Biophysical Journal*, **30**, 79–97.

Golding, N.L., Mickus, T.J., Katz, Y., Kath, W.L., and Spruston, N. 2005. Factors mediating powerful voltage attenuation along CA1 pyramidal neuron dendrites. *Journal of Physiology-London*, **568**(1), 69–82.

Gulyás, A.I., Megías, M., Emri, Z., and Freund, T.F. 1999. Total number and ratio of excitatory and inhibitory synapses converging onto single interneurons of different types in the CA1 area of the rat hippocampus. *Journal of Neuroscience*, **19**, 10082–97.

Hamming, R.W. 1987. *Numerical Methods for Scientists and Engineers.* 2 edn. Dover Publications.

Hindmarsh, A.C., and Serban, R. 2002. *User documentation for CVODES, an ODE solver with sensitivity analysis capabilities.* Tech. rept. UCRL-MA-148813. Lawrence Livermore National Laboratory.

Hindmarsh, A.C., and Taylor, A.G. 1999. *User documentation for IDA, a differential-algebraic equation solver for sequential and parallel computers.* Tech. rept. UCRL-MA-136910. Lawrence Livermore National Laboratory.

Hines, M. 1984. Efficient computation of branched nerve equations. *International Journal of Bio-Medical Computation*, **15**, 69–76.

Hines, M.L., and Carnevale, N.T. 1997. The NEURON simulation environment. *Neural Computation*, **9**, 1179–1209.

Hines, M.L., and Carnevale, N.T. 2001. NEURON: a tool for neuroscientists. *The Neuroscientist*, **7**, 123–135.

Hodgkin, A.L., and Huxley, A.F. 1952. A quantitative description of membrane current and its application to conduction and excitation in nerve. *Journal of Physiology*, **117**, 500–544.

Ito, M. 1989. Long-term depression. *Annual Review of Neuroscience*, **12**, 85–102.

Jack, J.J.B., Noble, D., and Tsien, R.W. 1983. *Electric Current Flow in Excitable Cells.* London: Oxford University Press.

Kundert, K. 1986. Sparse matrix techniques. In: Ruehli, Albert (ed), *Circuit Analysis, Simulation and Design.* North-Holland.

Mainen, Z.F., and Sejnowski, T.J. 1996. Influence of dendritic structure on firing pattern in model neocortical neurons. *Nature*, **382**, 363–366.

Moore, J.W., and Cox, E.B. 1976. A kinetic model for the sodium conductance system in squid axon. *Biophysical Journal*, **16**, 171–192.

Moore, J.W., and Stuart, A.E. 2000. *Neurons in Action: Computer Simulations with NeuroLab.* Sunderland, MA: Sinauer Associates.

Moore, J.W., and Stuart, A.E. 2007. *Neurons in Action 2: Tutorials and Simulations using NEURON.* Sunderland, MA: Sinauer Associates.

Nilsson, J.W., and Riedel, S.A. 1996. *Electric Circuits.* 5 edn. Reading, MA: Addison-Wesley.

Press, W.H., Teukolsky, S.A., Vetterling, W.T., and Flannery, B.P. 1992. *Numerical Recipes in C.* 2 edn. Cambridge: Cambridge University Press.

Rall, W. 1964. Theoretical significance of dendritic tree for input-output relation. Pages 73–97 of: Reiss, R.F. (ed), *Neural Theory and Modeling.* Stanford: Stanford University Press.

Rall, W. 1977. Core conductor theory and cable properties of neurons. Pages 39–98 of: Kandel, E. R. (ed), *Handbook of Physiology, vol. 1, part 1: The Nervous System.* Bethesda, MD: American Physiological Society.

Stewart, D., and Leyk, Z. 1994. *Meschach: Matrix Computations in C.* Vol. 32. Canberra, Australia: School of Mathematical Sciences, Australian National University.

Strang, G. 1986. *Introduction to Applied Mathematics.* Wellesley, MA: Wellesley-Cambridge Press.

Thomson, A.M., and Deuchars, J. 1997. Synaptic interactions in neocortical local circuits: dual intracellular recordings in vitro. *Cerebral Cortex*, **7**, 510–522.

Index