brains, minds media
ISSN: 1861-1680

GENESIS Modeling Tutorial

David Beeman

Electrical and Computer Engineering Department, University of Colorado, Boulder, USA.

Phone: (303) 492-2852, fax: (303) 492-2758, email: dbeeman@dogstar.colorado.edu

urn:nbn:de:0009-3-2206

Abstract. This tutorial is intended to be a "quick start" to creating simulations with GENESIS. It should give you the tools and enough information to let you quickly begin creating cells and networks with GENESIS, making use of the provided example simulations. Advanced topics are covered by appropriate links to the Advanced Tutorials on Realistic Neural Modeling.

Keywords: computational neuroscience, realistic neural modeling, simulations, GENESIS, tutorial

Citation: Beeman D (2005). GENESIS Modeling Tutorial. Brains, Minds and Media, Vol. 1, bmm220 (urn:nbn:de:0009-3-2206)

Supplementary Material

GENESIS Resources

Installation Guidelines

Datasheet

Introduction

This is an updated version of the multi-part GENESIS Modeling Tutorial for the 2005 WAM-BAMM (http://www.wam-bamm.org) meeting on realistic biological modeling in San Antonio, TX. It was offered as a "take-home" hands-on tutorial to be used for self-study, following the Introduction to Realistic Neural Modeling tutorial (Beeman 2005, this Volume). It is intended to be a "quick start" to creating simulations with GENESIS, and will give you enough information to let you quickly begin creating cells and networks with GENESIS.

One way to use the tutorial is to go through it at a leisurely pace, exploring all the links to detailed information and documentation as you create models and perform the exercises. Another way is to take the most direct path, creating models without always understanding the details of what you have done. If you opt for this path, be sure to return to pick up the details when your lack of understanding makes progress difficult. Take your pick!

The tutorial makes frequent references and links to sections of the GENESIS Reference Manual[1]. This is primarily a summary of the syntax used by the GENESIS Script Language Interpreter, the commands which it recognizes, and of the GENESIS "objects" that are available for constructing simulations. It also refers to chapters in *The Book of GENESIS*, by James M. Bower and David Beeman (1998). The second edition of this book, familiarly called "The BoG", was originally published by Springer-Verlag. As the publisher does not plan to reprint the book, the copyright has been reclaimed by the authors, and we are now able to offer it as a free "Internet Edition"[2], with links to it in this package of Tutorials.

The BoG gives a detailed coverage of realistic neural modeling from the subcellular to the network level, and provides the detailed guide to the construction of GENESIS simulations that is missing from the Reference Manual. This tutorial is constructed to minimize your need for the BoG. Nevertheless, this short tutorial can't possibly go into all the detail of a 482 page book, so you may wish study it in some detail, if you later decide to do some serious work with GENESIS. There are links to chapters in the BoG throughout this tutorial, in order to provide more information on some of the topics that we will cover.

GENESIS Scripting

If you have read the *Introduction to Realistic Neural Modeling* (Beeman 2005, this volume), you will be familiar with the rich variety of graphical user interfaces (GUIs), that are possible with GENESIS. In fact, it is possible to create and run GENESIS simulations with little or no programming, by using Neurokit (for single cells) or Kinetikit (for biochemical reactions). But, sooner or later, you will want the flexibility of creating your own GENESIS scripts.

The bad news is that to get the maximum benefit from GENESIS and from this tutorial, you will have to do some programming in the GENESIS scripting language. The good news is that the modular object-oriented nature of GENESIS makes it easy to modify existing scripts. Once you have learned a few basics of GENESIS scripting and have some good example simulation scripts to get you started, you can create most of your simulations by simple "hacking" of existing scripts. That is the approach taken in this tutorial, and by most GENESIS modelers.

Some preliminaries

Before you start, you will need a little background.

[1] The GENESIS Reference Manual http://www.genesis-sim.org/GENESIS/Hyperdoc/Manual.html; last visit August 01, 2005.
[2] James M. Bower and David Beeman 2003 - Free Internet Edition (pdf) of The Book of Genesis, 2nd Edition. http://www.genesis-sim.org/GENESIS/bog/bog.html; last visit: August 01, 2005

- You should have some basic knowledge of compartmental modeling, which is covered in detail in the BoG. You can get a brief overview in parts of the Lectures on Computational Neuroscience[3]. The book *Methods in Neuronal Modeling* (Koch and Segev 1998) is another good reference. The *Introduction to Realistic Neural Modeling* (Beeman 2005, this volume) will also be a good background for understanding what follows.

- You should have GENESIS installed, and know how to start it up. You will need to know *where* GENESIS is installed. There will be a directory *genesis* that contains the executable program *"genesis"* and other directories, such as *Scripts*, which contains the example and tutorial simulation scripts that are distributed with GENESIS. So, when we refer to *genesis/Scripts*, you will need to know the full path, e.g. */usr/local/genesis/Scripts*, in order to locate these files. You will also need to know where these tutorials are installed, as they may refer to the "GENESIS Tutorials directory", or to a file within this directory such as *cells/simplecell/simplecell.g*.

- The "GENESIS Tutorials directory" is an optional package that would normally be installed as *genesis/Tutorials*. It contains this tutorial (in *Tutorials/genprog*), other GENESIS tutorials, the *cells* directory (in *Tutorials/cells*), and other directories used by the GENESIS tutorials. The procedure for obtaining, installing and running GENESIS and the Tutorials package is given in the README file in the supplementary material or in the original Tutorials directory, respectively.

- You will need to know how to get around in a UNIX command-line environment, and how to use a text editor. Fortunately, you only need to know a little. You can learn most of what you need to know about UNIX in order to use GENESIS in this Introduction to UNIX or Linux and the graphical desktop.

The two most common text editors for UNIX are 'vi' and 'emacs'. If you are not familiar with either editor, you may find it easier to learn emacs. For an even simpler text editor with built-in help, try 'pico' if it is installed. If you are using Linux with the GNOME or KDE desktop, try 'gedit' or 'kedit".

Getting started with GENESIS programming

The building blocks used to create simulations under GENESIS are referred to as "elements". Elements are created from templates called "objects". This terminology can be somewhat confusing, because a GENESIS object is similar to what object-oriented languages such as Java or C++ call a "class", and a GENESIS element corresponds to an "object" in these languages. In GENESIS, elements are created as instantiations of a particular object.

Simulations are constructed from these modules that receive inputs, perform calculations on them, and then generate outputs. Model neurons are constructed from these basic components, such as compartments. and variable conductance ion channels.

[3] Dave Beeman – Lectures on Computational Neuroscience, http:/www.genesis-sim.org/GENESIS/cnsweb/cnslecs.html; last visit: August 1, 2005.

The various elements in a GENESIS simulation are organized in a tree like the one shown below, and are referenced with a notation similar to that used in UNIX directory trees. Thus, "/net/cell[1]/dend[2]/GABA" might refer to an inhibitory synaptically activated conductance residing in the dendrite compartment 2 of cell 1 of a network.

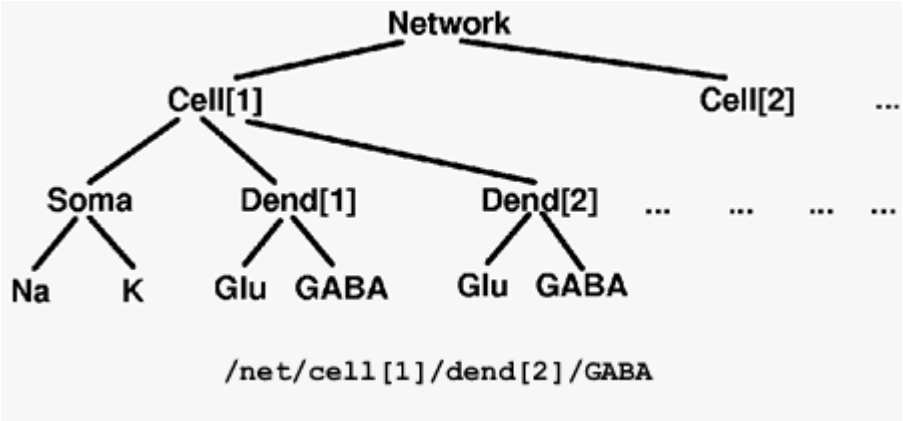

Figure 1: The various elements in a GENESIS simulation are organized in a tree and are referenced with a notation similar to that used in UNIX directory trees.

These objects communicate by passing messages to each other, and each contain the knowledge of their own variables (fields) and the methods (actions) that they use to perform their duties during a simulation. For example, during a simulation step, the PROCESS action will be carried for each type of object in its own way. If it is a voltage activated channel, this means carrying out a step in the numerical solution of the Hodgkin-Huxley equations for the conductance. If it is a graph object, this would mean plotting a point with data from any messages that it receives from other objects.

GENESIS Scripting Example

tutorial1.g

The use of the GENESIS scripting language can best be illustrated with a simple example like the script from genesis/Scripts/tutorials/tutorial1.g.

```
//genesis script for a simple compartment simulation (Tutorial #1)
// create a parent element

create neutral /cell


// create an instance of the compartment object

create compartment /cell/soma


// set some internal fields

setfield /cell/soma Rm 10 Cm 2 Em 25 inject 5
```

```
// create and display a graph inside a form
create xform /data
create xgraph /data/voltage
xshow /data
// set up a message (PLOT Vm) to the graph
addmsg /cell/soma /data/voltage PLOT Vm *volts *red
addmsg /cell/soma /data/voltage PLOT inject *current *blue

// make some buttons to execute simulation commands
create xbutton /data/RESET -script reset
create xbutton /data/RUN -script "step 100"
create xbutton /data/QUIT -script quit

check       // perform a consistency check for each element
reset       // initialize each element before starting the simulation
```

This would produce the display:

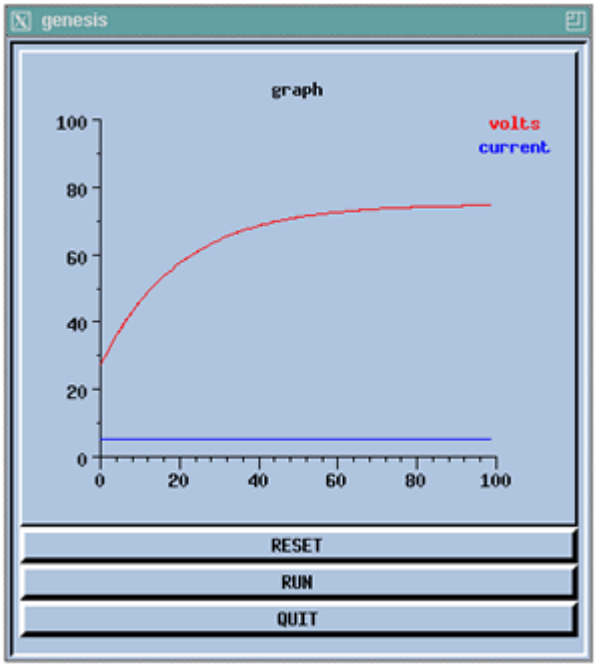

Figure 2: Display for *tutorial1.g*.

Unless you have some previous experience with GENESIS (or are very impatient), you should quickly run through the updated web version of Matt Wilson's original “Basic tutorial on using GENESIS” (skip Basic Tutorial). This gives an introduction to the most common GENESIS commands for creating, interacting with, and debugging GENESIS simulations on the command line, and leads you through the steps of creating the tutorial1.g script. (This tutorial was later expanded to become Chapter 12 of the BoG)

A Basic Tutorial on GENESIS - Constructing a Simple Compartment

Originally written by *Matt Wilson*

This document will guide you through a brief session under GENESIS in which you will use the basic features of the simulator to create and run a simple simulation.

Some Notation

In this guide you will be instructed, at various points, to enter GENESIS commands through the keyboard. This will be indicated by showing the text that you should enter in a monospaced "typewriter" font. For example:

```
type this
```

Getting Started

To run the simulator, first make sure that you are at the UNIX shell command prompt. At the prompt type genesis. If your path is properly configured this should start up the simulator and display the opening credits. If you get a message such as genesis: Command not found, check your path (echo $PATH) to be sure that it contains the genesis directory (often */usr/genesis*).

Interpreter Basics

After the simulator has completed its startup procedure you should see the GENESIS command prompt "genesis #0 >" indicating that you are now in the GENESIS interpreter (SLI). In the interpreter you can execute both UNIX shell and GENESIS commands. Try this by typing

```
ls
```

This should invoke the UNIX ls command displaying files in the current directory. Typing

```
listcommands
```

should produce a list of available GENESIS commands. Note that while there are a large number of available commands, you will typically use a much smaller subset of these. It is also possible to combine GENESIS and UNIX shell commands.

Typing

```
listcommands | more
```

will "pipe" the output of the GENESIS command listcommands through the UNIX command more, thus allowing you to page through the listing.

```
listcommands | more
```

will "pipe" the output to the printer and

```
listcommands > myfile
```

will redirect the output into a file called "myfile".

Basic Objects

The building blocks used to create simulations under GENESIS are referred to as elements. Elements are created from templates called "objects". The simulator comes with a number of basic objects. To list the available objects type

```
listobjects
```

To get more information on a particular objects type

```
showobject name
```

where "name" is replaced by any name from the object list.

The **compartment** object is commonly used in GENESIS simulations to construct parts of neurons. As we will be using this object, try the command showobject compartment at this time. There are a few commonly used objects which are documented more thoroughly with the GENESIS help command. In order to obtain a detailed description of the equivalent circuit for the **compartment** object, type

```
help compartment
```

(HINT: You may pipe these commands into more to prevent the output from scrolling off the top of the screen.) For example.

```
help compartment | more
```

Creating Elements

To create an Element from an Object description you use the create command. Try typing the create command without arguments

```
create
```

This gives a usage statement which gives the proper syntax for using this command. Most commands will produce a usage statement if invoked without arguments, or if followed with the option -usage or -help.

In the case of the create command the usage statement looks like

```
usage: create object name -autoindex [object-specific-options]
```

In this exercise we will create a simple passive compartment. In order to keep track of the many elements that go into a simulation, each element must be given a name. To create a compartment with the name *soma* type

```
create compartment /soma
```

Elements are maintained in a hierarchy much like that used to maintain files in the UNIX operating system. In this case, */soma* is a pathname which indicates that the soma is to be placed at the root or top of the hierarchy.

We will eventually build a fairly realistic neuron called */cell* with a soma, dendrites, channels and an axon. It would be a good idea to organize these components into a hierarchy of elements such as */cell/soma*, */cell/dend*, */cell/dend/Ex_channel*, and so on. If we do this, we need to create the appropriate type of element for */cell*. GENESIS has a **neutral** object for this sort of use. An element of this type is an empty element that performs no actions and is used chiefly as a parent element for a hierarchy of child elements.

To start the construction of our cell, give the commands

```
create neutral /cell

create compartment /cell/soma}
```

As we no longer need our original element /soma, we may delete it with the command delete.

```
delete /soma
```

Examining Elements

The commands for maintaining elements within their hierarchy are very much like those used to maintain files in the UNIX operating system. In that spirit, the commands for moving about within the GENESIS element hierarchy are similar to their UNIX counterparts. For example, to list the elements in the current level of the hierarchy use the le (list elements) command

```
le
```

You should see several items listed including the newly created *cell*.

Each element contains data fields which contain the values of parameters and state variables used by the element. To show the contents of these data fields use the showfield command.

```
showfield /cell *
```

This will display the names and contents of the data fields of the "cell". The "*" indicates that you wish to display all the data fields associated with the element. To display the contents of a particular field, type

```
showfield /cell/soma Rm
```

To display an extended listing of the element contents including a description of the object associated with the element, type

```
showfield /cell/soma **
```

Moving About in the Hierarchy

When working in GENESIS you are always located at a particular element within the hierarchy which is referred to as the "working element". This location is used as a default for many commands which require path specifications. For example, the le command used above normally takes a path argument. When the path argument is omitted the working element is used and thus all elements located under the working element are listed. To move about in the hierarchy use the ce (change element) command. To change the current working element to the newly create soma, type

```
ce /cell/soma
```

Now you can repeat the show command used above omitting the explicit reference to the */cell/soma* pathname.

```
showfield *
```

This should display the contents of the */cell/soma* data fields. You may find the current working element by using the pwe (print working element) command. Try giving the command:

```
pwe
```

Note the analogy between these commands and the UNIX commands ls, cd, and pwd. By analogy with UNIX, GENESIS uses the symbols "." to refer to the working element, and ".." to refer to the element above it in the hierarchy. Try using these with the le, ce, and showfield commands. Likewise, GENESIS has pushe and pope commands to correspond to the UNIX pushd and popd commands. These provide a convenient method of changing to a new working element and returning to the previous one. Try the sequence of commands

```
pushe /cell
pwe
pope
pwe
```

Modifying Elements

The contents of the element data fields can be changed using the setfield command. To set the transmembrane resistance of your cell type

```
setfield /cell/soma Rm 10
```

You can set multiple fields in a single command as in

```
setfield /cell/soma Cm 2 Em 25 inject 5
```

Now if you do a showfield command on the element you should see the new values appearing in the data fields.

```
setfield /cell/soma *
```

State variables are automatically updated by the elements when they are "run" during a simulation. For instance the V_m field is a state variable which, while you can change it, will be updated by the element automatically, replacing your value.

Running a Simulation

Before running a simulation the elements must be placed in a known initial state. This is done using the reset command, which should be performed prior to all simulation runs.

```
reset
```

If you now show the value of the compartment voltage V_m you will see that it has been reset to the value given by the parameter E_m.

```
showfield /cell/soma *
```

To run a simulation use the step command, which causes the simulator to advance a given number of simulation steps.

```
step 10
```

Displaying the V_m field now shows that the simulator actually did something and the value has changed from its initial value due to the current injection.

```
showfield /cell/soma Vm
```

Adding Graphics

Some people find that graphics are more effective than endless columns of numbers in monitoring the course of a simulation. With that in mind we will attempt to add a graph to the simulation which will display the voltage trajectory of your cell. Graphics are implemented using graphical objects from the XODUS library which are manipulated using the same techniques described above. The "form" is the graphical object which is used as a container for all other graphical items. Thus, before making a graph we need to make a form to put it in which we will arbitrarily name */data*.

```
create xform /data
```

You may have noticed that nothing much seemed to happen. By default, forms are hidden when first created. To reveal the newly created form use the command

```
xshow /data
```

An empty box should appear somewhere on your screen. To create a graph in this form with the name voltage use the command

```
create xgraph /data/voltage
```

Note that the graph was created beneath the form in the element hierarchy. This is quite important, as the hierarchy is used to define the nesting of the displayed graphical elements.

Linking Elements

Now you have a cell and a graph but you need some way of passing information from one to the other.

Inter-element communication within GENESIS is achieved through a system of links called *messages*. Messages allow one element to access the data fields of another element. For example to cause the graph to display the voltage of the cell you must first pass a message from the cell to the graph indicating that you would like a particular data field to be plotted. This is done using the command

```
addmsg /cell /data/voltage PLOT Vm *volts *red
```

The first two arguments give it the source and destination elements. The third argument tells it what type of message you are sending. In this case the message is a request to plot the contents of the fourth argument which is the name of the data field in the cell which you wish to be plotted. The last two arguments give the label and color to be used in plotting this field. You can now run the simulation and view the results in the graph.

```
reset
step 100
```

Note that to plot another field in the same graph, just send another message

```
addmsg /cell /data/voltage PLOT inject *current *blue
reset
step 100
```

and you are displaying current and voltage.

Adding Buttons to a Form

The **xbutton** graphical element is often used to invoke a function when a mouse button is clicked. Give the command

```
create xbutton /data/RESET -script reset
```

This should cause a bar labeled RESET to appear within the "data" form below the "voltage" graph. When the mouse is moved so that the cursor is within the bar and the left mouse button is clicked, the function following the argument -script is invoked.

Now add another button to the form with the command

```
create xbutton /data/RUN -script "step 100"
```

In this case, the function to be executed has a parameter (the number of steps), so "step 100" must be enclosed in quotes so that the argument of -script will be treated as a single string.

At this stage, you have a complete GENESIS simulation which may be run by clicking the left mouse button on the bar labeled RESET and then on the one labeled RUN. To terminate the simulation and leave GENESIS, type either quit or exit. If you like, you may implement one of these commands with a button also.

At this time, you should use an editor to create a script containing the GENESIS commands which were used to construct this simulation. The script should begin with

```
//genesis
```

and the filename should have the extension "*.g*". For example, if the script were named *tutorial1.g*, you could create the objects and set up the messages with the GENESIS command

```
tutorial1
```

If you have exited GENESIS and are back at the unix prompt, you may run GENESIS and bring up the simulation with the single command

```
genesis tutorial1
```

Making realistic neural compartments

The soma compartment that was simulated in the *tutorial1.g* script corresponds to the "generic neural compartment" diagram (Fig. 3) but without the variable ionic conductances G_k that we will add later. As it is a single isolated compartment, we didn't make use of the axial resistance R_a. The diagram reveals that the current I_{inject} flows through R_m to create a potential difference that is in series with E_m. The simulation results show that initially, V_m will equal E_m, and the steady state will be reached after a time given roughly by the time constant for charging the membrane capacitance, R_mC_m. With the values used, the time constant was 20.

A lot of the simplicity of the script stems from the fact that the numbers used in the simulation worked well with the default values of the graph axis scales and the default integration step size used by GENESIS to integrate the equation for the compartment V_m. In order to make a realistic soma compartment that we can then link to dendrite compartments and populate with ion channels, we will need to pick appropriate values for the passive cell parameters R_m, R_a, C_m, and the membrane resting potential E_m.

So far, we haven't said much about the units used to express the quantities R_m, R_a, C_m, V_m, etc. that appear in the neural compartment diagram (Fig. 3) and the differential equation for V_m (Eq.1).

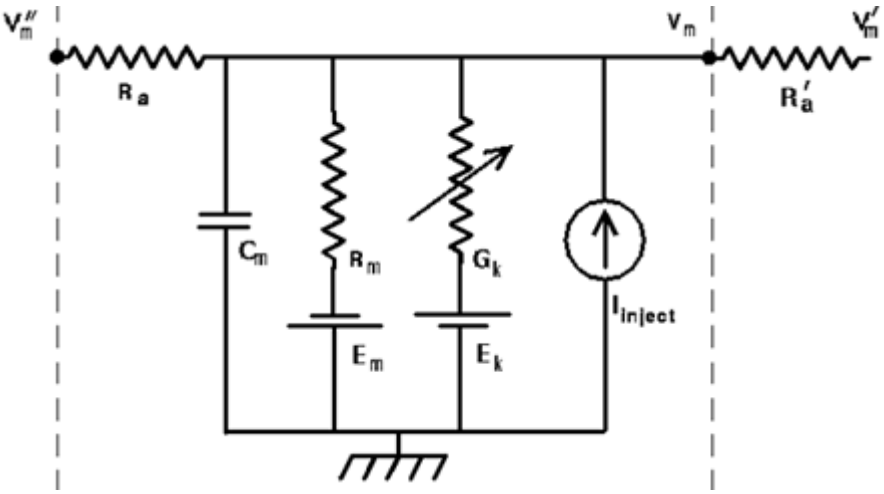

Figure 3: Circuit diagram for a generic compartment.

$$C_m \frac{dV_m}{dt} = \frac{(E_m - V_m)}{R_m} + \sum_k \left[(E_k - V_m) G_k \right] + \frac{(V'_m - V_m)}{R'_a} + \frac{(V''_m - V_m)}{R_a} + I_{inject} \quad (1)$$

Equation 1: Differential equation for V_m

Physicists and engineers like to use SI (MKS) units of ohms, farads, volts, and meters for describing resistance, capacitance, voltage, and length. Neurophysiologists are more likely to prefer kilohms, microfarads, millivolts, and either centimeters or micrometers. One can use any consistent set of units with GENESIS, but it is most common to use SI units.

The problem with using any of these units for resistance and capacitance is that R_m, C*m*, and R_a will depend on the dimensions of the section of dendrite that is represented by the neural compartment. In order to specify parameters that are independent of the cell dimensions, *specific units* are used. For a cylindrical compartment, the membrane resistance is inversely proportional to the area of the cylinder, so we define a *specific membrane resistance* R_M, which has units of *ohms·m²*. The membrane capacitance is proportional to the area, so it is expressed in terms of a *specific membrane capacitance* C_M, with units of *farads/m²*. Compartments are connected to each other through their axial resistances R_a. The axial resistance of a cylindrical compartment is proportional to its length and inversely proportional to its cross-sectional area. Therefore, we define the *specific axial resistance* R_A to have units of *ohms/m*.

For a piece of dendrite or a compartment of length l and diameter d we then have

$$R_m = \frac{R_M}{\pi l d} \ , \quad C_m = \pi l d C_M \ , \quad R_a = \frac{4 l R_A}{\pi d^2} \ . \quad (2)$$

Note the membrane time constant $R_m \cdot C_m$ is also equal to $R_M \cdot C_M$, so that it is independent of the dimensions of the membrane.

WARNING: Many treatments of the passive properties of neural tissue use the symbols R_m, R_a, and C_m for the specific resistances and capacitance, instead of this notation with R_M, R_A, and C_M. Also,

many textbooks and journal papers define the resistance and capacitance in terms of that for a unit length of cable having a specified diameter, where $R_m = r_m/l$, $C_m = c_m l$, $R_a = r_a l$.

Although this notation is convenient and widely used, it obscures the fact that r_m and r_a depend on the dendrite diameter. In your reading, you should be aware of the units that are being used.

You can read more about passive properties of dendrites in the Digression on Cable Theory from the Introduction to Computational Neuroscience lectures.

Our goal is to build a cylindrical soma compartment that has the same physiological properties as those of the squid giant axon studied by Hodgkin and Huxley, and simulated in the GENESIS "squid" tutorial in *genesis/Scripts/squid*. So, we will use these values (in SI units) for the compartment parameters. However, we will make our soma smaller, with both the length and diameter equal to 30 micrometers.

We will also need to choose an appropriate time step for the numerical solution of the equation for V_m. With the values of R_M and C_M that we will use (R_M = 0.33333 and C_M = 0.01), the membrane time constant will be 0.003333 seconds. We would then expect our integration time step to be a small fraction of this. In practice, it turns out that 50 microseconds (0.00005 sec) will be a good value.

You can (and should, at some point) read the section in the GENESIS Reference Manual on Clocks for further suggestions on choosing a time step. The documentation for the commands `setclock` and `useclock` gives the details of setting the time step.

Tutorial2.g

Chapter 13 of the BoG leads the reader through the process of developing the script *tutorial2.g*. If you like, you can run this script from the *genesis/Scripts/tutorials* directory. You should now examine *tutorial2.g:*

```
//genesis  -  tutorial2.g - GENESIS Version 2.0
/*==========================================================================
  A sample script to create a soma-like compartment. SI units are
used.

==========================================================================*/


float PI = 3.14159


// soma parameters - chosen to be the same as in SQUID (but in SI
// units)
float RM = 0.33333       // specific membrane resistance (ohms m^2)
float CM = 0.01          // specific membrane capacitance (farads/m^2)
```

```
float RA = 0.3           // specific axial resistance (ohms m)
float EREST_ACT = -0.07  // resting membrane potential (volts)
float Eleak = EREST_ACT + 0.0106  // membrane leakage potential
(volts)
float ENA   = 0.045      // sodium equilibrium potential
float EK    = -0.082     // potassium equilibrium potential

// cell dimensions (meters)
float soma_l = 30e-6     // cylinder equivalent to 30 micron sphere
float soma_d = 30e-6

float dt = 0.00005       // simulation time step in sec
setclock  0  {dt}        // set the simulation clock

//==============================
//      Function Definitions
//==============================

function makecompartment(path, length, dia, Erest)
    str path
    float length, dia, Erest
    float area = length*PI*dia
    float xarea = PI*dia*dia/4

    create      compartment     {path}
    setfield   {path}              \
                Em      { Erest }   \           // volts
                Rm      { RM/area } \           // Ohms
                Cm      { CM*area } \           // Farads
                Ra      { RA*length/xarea }     // Ohms
end

function make_Vmgraph
    float vmin = -0.100
    float vmax = 0.05
    float tmax = 0.100   // default simulation time = 100 msec
```

```
    create xform /data
    create xgraph /data/voltage
    setfield ^ xmax {tmax} ymin {vmin} ymax {vmax}
    create xbutton /data/RESET  -script reset
    create xbutton /data/RUN  -script "step "{tmax}" -time"
    create xbutton /data/QUIT -script quit
    xshow /data
end

//================================
//         Main Script
//================================

create neutral /cell
// create the soma compartment "/cell/soma"
makecompartment /cell/soma {soma_l} {soma_d} {Eleak}

// provide current injection to the soma
setfield /cell/soma inject  0.3e-9      // 0.3 nA injection current

// make the graph to display soma Vm and pass messages to the graph
make_Vmgraph
addmsg /cell/soma /data/voltage PLOT Vm *volts *red

check
reset
```

There are several features of the GENESIS script language that are introduced here:

- The use of a C-style multiline comment, extending from the second through fourth line.
- Declaration of variables to be used. In this script only floating point variables (float) are used, but "int" and "str" are also allowed. The declaration and use of variables is explained in the GENESIS Reference Manual section on Variables.
- The line "setclock 0 {dt}" sets the global simulation clock (clock 0) to the specified time step, dt. Note the use of curly brackets around the variable name dt. Usually, the value of a variable or an expression must be evaluated by enclosing it in curly brackets in order to distinguish

between the value represented by the character string (0.00005) and the actual string of characters "dt". This is particularly true when the expression is to be evaluated as an argument of a GENESIS command or script language function.

- The use of function declarations for *makecompartment* and *make_Vmgraph*, and their use in the "Main Script" section. GENESIS script language functions are described in the GENESIS Reference Manual section on Functions.

- The use of a backslash to continue a line in the long setfield command in the *makecompartment* function.

- Setting the graph scales, and the use of "^" to denote the element last referenced (in this example, */data/voltage*).

- The use of the "-time" option for the step command with the RUN button, to give the amount of time to run the simulation, rather than the number of steps. (See the documentation for step.)

Building a cell the easy way

Now that we know how to make a realistic neural compartment, the next steps in creating a realistic model of a neuron are to:

1. Set the passive membrane parameters (membrane resistance and capacitance, axial resistance, and membrane resting potential (R_m, C_m, R_a, and E_k) for each of the compartments.

2. Populate the compartments with ionic conductances ("channels"), or other related neural elements.

3. Link compartments for the soma and dendrites together with appropriate messages to make a cell.

The Book of GENESIS gradually builds up to the creation of a cell the "hard way" in Chapter 14 and Chapter 15, describing the element fields that need to be set and the messages that need to be established between the elements. It then describes an easier way in Chapter 16 that uses the GENESIS cell reader to perform these three steps.

There is some value to learning the details of the "hard way" in order to understand what the cell reader is doing, and the messages that it sets up between the elements that make up a cell model. So, here is a link to a short summary of this material:

Detour: Building a cell without the cell reader.

At some point, it would be useful to follow this link to see how these steps would be done using separate GENESIS commands. But, for now, let's plunge ahead and create a simple neuron with the cell reader.

In this part of the tutorial we will use the cell reader to create a simple two-compartment neuron with a dendrite compartment, a soma, and an axon. The dendrite contains synaptically activated excitatory and inhibitory channels and the soma contains voltage-activated Hodgkin-Huxley sodium and potassium channels, plus a **spikegen** element that acts like the initial part of an axon. This may be used to provide synaptic input to another cell.

Locate the "GENESIS Tutorials directory" that contains the GENESIS tutorials and "cd" to it in a terminal window. "ls" will show you that it contains a cells directory, which contains other directories having various GENESIS cell models. To begin, "cd" to the directory *cells/simplecell* and run the simplecell simulation by typing "genesis simplecell". You may vary the injection current (given in Amperes) from the default value of 0.5 nA, by editing the value in the "Injection" dialog box. NOTE: after entering a value in a GENESIS dialog box, you must hit the "Enter" key for the value to be accepted. Experiment with the simulation, and the effect of the RESET and "Overlay" toggle button after changing the injection current. Notice the "scale" button on the graph lets you change the scales for graph axes.

SimpleCell.g

Now, it's time to understand the *simplecell.g* script, which contains only:

```
//genesis  -  simplecell.g

/*=====================================================================
  A sample script to create a neuron containing channels taken from
  hh_tchan.g in the neurokit prototypes library.  SI units are used.

=====================================================================*/


// Create a library of prototype elements to be used by the cell
// reader
include protodefs


float tmax = 0.5                // simulation time in sec
float dt = 0.00005              // simulation time step in sec
setclock  0  {dt}               // set the simulation clock


// include the graphics functions
include graphics



//===============================
//         Main Script
```

```
//================================

readcell cell.p /cell

// make the control panel
make_control

// make the graph to display soma Vm and pass messages to the graph
make_Vmgraph
addmsg /cell/soma /data/voltage PLOT Vm *volts *red

setfield /control/Injection value 0.5e-9
set_inject /control/Injection  // set initial injection from Injection
                                  dialog

check
reset
```

The statement "include protodefs" merges in the contents of the *protodefs.g* file, which is in the current (*cells/simplecell*) directory. This file, which we will examine shortly, creates the prototype compartment and channels, from which the cell reader will construct our cell.

The included file, *graphics.g*, contains the definition of the functions *make_control* and *make_Vmgraph* that are used to make the control panel and the graph for plotting the soma membrane potential V_m.

cell.p

The cell is constructed with the single command

```
readcell cell.p /cell
```

which creates a **neutral** placeholder element */cell* and builds the cell under it, according to the specifications in the cell parameter file *cell.p*:

```
// cell.p - Cell parameter file used in the simplecell tutorial
// Format of file :
// x,y,z,dia are in microns, all other units are SI (Meter Kilogram
// Sec Amp)
// In polar mode 'r' is in microns, theta and phi in degrees
// readcell options start with a '*'
// The format for each compartment parameter line is :
// name  parent r theta  phi  d  ch  dens ...
// in polar mode, and in cartesian mode :
// name  parent x  y  z  d  ch  dens ...
// For channels, "dens" =  maximum conductance per unit area of
// compartment
// For spike elements, "dens" is the spike threshold
//          Coordinate mode
*relative
*cartesian
*asymmetric

// Specifying constants
*set_compt_param   RM    0.33333
*set_compt_param   RA    0.3
*set_compt_param   CM    0.01
*set_compt_param     EREST_ACT -0.07

// For the soma, use the leakage potential (-0.07 + 0.0106) for
// Em *set_compt_param     ELEAK     -0.0594
soma  none   30  0  0  30  Na_hh_tchan 1200   K_hh_tchan 360 spike 0.0
// The dendrite has no H-H channels, so ELEAK = EREST_ACT
// *set_compt_param     ELEAK  -0.07
dend soma  100  0  0   2   Ex_channel 0.795775 Inh_channel 0.397888
```

The comments in the file give a brief description of the format of a cell parameter file, and further details are given in the documentation for `readcell`. The readcell options used here specify that relative cartesian coordinates will be used with asymmetric compartments. (If the option "*symmetric" had been used, then the cell would be constructed using symcompartment elements).

This means that for lines such as

```
soma  none   30  0  0  30
dend  soma  100  0  0   2
```

the soma, which is the start of the cell and has no parent, has its end at the (x, y, z) coordinates of (30, 0, 0) in micrometers. As the cell starts at (0, 0, 0), this means that the soma has a length of 30 micrometers and a diameter of 30 micrometers. The "dend" section will be connected to its parent, the soma, through its own axial resistance Ra, with suitable messages established between the two compartments. As relative coordinates are being used, the end of the dendrite compartment will lie 100 micrometers along the x-axis from the end of of the soma compartment.

With the *simplecell simulation* running, give these commands to the genesis prompt:

```
showfield /cell/soma -all
showfield /cell/dend -all
```

The "-all" option causes the compartment start (x0, y0, z0) and end (x, y, z) coordinates to be displayed, along with the length and diameter and other compartment fields. Verify that these values are what you would expect.

For a more interesting example of 9-compartment cell with branching basal dendrites, see the cell parameter file cells/corticalcells/layer5.p.

In the *tutorial2.g* script, the specific membrane parameters R_M, C_M, and R_A were declared and given values. The *makecompartment* function was used to calculate and set the proper values of the fields R_m, C_m, and R_a using these parameters and the compartment length and diameter. Here, this is all done by the cell reader, using the values of R_M, R_A, and C_M that were specified using the "*set_compt_param" option and the compartment coordinates and diameter. You can verify that this was correctly done, by using the showfield commands above.

tutorial2.g also defined the resting potential of the compartment EREST_ACT, but set the soma compartment E_m field to a different value, "Eleak = EREST_ACT + 0.0106". Normally, we would expect to set this field, which represents the "battery" in series with the membrane resistance R_m, to the value of EREST_ACT. However, Hodgin and Huxley found it necessary to set E_m to a leakage potential E_{leak} that compensates for current flow through other channels (such as chloride channels) which were not explicitly taken into account in their model. E_{leak} is set to a value that results in no net current flow when the cell is at EREST_ACT. The cell reader takes care of this by not only setting EREST_ACT to the value specified in the cell parameter file, but allowing the use of another parameter ELEAK, which if specified, gives an alternate value E_m, but allows V_m to be initialized to EREST_ACT on reset, instead of E_m. (For further details of the initializaton of V_m on reset, see the documentation for compartment.)

Therefore, the cell parameter file above sets ELEAK to EREST_ACT + 0.0106 for the soma, but sets it to EREST_ACT for the dendrite.

Finally, note the list of channels and their conductance densities that follow the compartment coordinates and diameter. For the soma, these are the values used by Hodgkin and Huxley for the squid giant axon, with the sodium channel (*Na_hh_tchan*) given a maximum conductance of 1200

Siemens per square meter of surface area, and the potassium channel (*K_hh_tchan*) given a value of 300 S/m^2. The cell reader scales these by the surface area to set the "Gbar" fields of the channel elements, and also sets up the necessary messages between the channels and the compartment that contains them.

Although the *simplecell* simulation does not make use of them, the dendrite compartment has been given excitatory and inhibitory synaptically activated channels, with appropriate conductance densities. A later section of the tutorial describes how these, and the spike generator element *spike* that was created in the soma, can be used when this cell is connected to another in a circuit or network. (If you want a preview, take a look at the documentation for synchan, spikegen, and the GENESIS Reference Manual section on Synaptic Connections).

To see the messages that have been set up by the cell reader (plus the PLOT message established by *simplecell.g*), give the commands

```
showmsg /cell/soma

showmsg /cell/dend
```

Creating channel prototypes

The remaining thing to be explained is the way that we tell the cell reader about the properties of the elements that the cell parameter file calls Na_hh_tchan, K_hh_tchan, spike, Ex_channel, and Inh_channel. If you are anxious to go on to connect cells together in a network, you can skip ahead to the section on Making synaptic connections and return to this part later. However, at some point, you will need to create these prototype channels.

The cell reader builds the cell by making copies of "prototypes" of the various elements that will be used, replacing the default values of parameter fields with values taken from the cell descriptor file. For example, when constructing a soma with several attached dendrite compartments, it will make multiple copies of a generic compartment prototype and then set the data fields in each compartment to the appropriate values. Likewise, a cell having Hodgkin-Huxley Na channels in several compartments will get these channels from copies of the single Na channel prototype, setting the value of the maximum channel conductance Gbar for each copy, making use of the specified conductance density and dimensions of the compartment that contains the channel.

The cell reader expects to find this library of prototype elements as a set of subelements of the neutral element */library*. Thus, we need to write a script that will create */library* and fill it with a prototype compartment, one copy of each of the different channel types we will use, and a spike generator. Although the statements that are needed to set up the prototype library could go into your main simulation script, it is customary to make a separate script for this, and to then use *include* to bring it into the simulation. This script is often called *protodefs.g*, although you may give it any name that you like in your own simulations.

protodefs.g

At this point, examine the listing for *protodefs.g*.

```
// protodefs.g - Definition of prototype elements for "simplecell"

/* Included files are in genesis/Scripts/neurokit/prototypes */

/* file for standard compartments */
include compartments

// include the definitions for the functions to create H-H tabchannels
include hh_tchan

/* hh_tchan.g assigns values to the global variables EREST_ACT, ENA,
EK, and SOMA_A.  The first three will be superseded by values defined
below. The value of SOMA_A set in hh_tchan.g is not relevant, as the
cell reader calculates the compartment area.
*/
EREST_ACT = -0.07  // resting membrane potential (volts)
ENA   = 0.045      // sodium equilibrium potential
EK    = -0.082     // potassium equilibrium potential

/* file for synaptic channels */
include synchans

/* file which makes a spike generator */
include protospike

// Make a "library element" to hold the prototypes, which will be used
// by the cell reader to add compartments and channels to the cell.
create neutral /library

// We don't want the library to try to calculate anything, so we
// disable it
disable /library

// To ensure that all subsequent elements are made in the library
```

```
pushe /library

/* Make a prototype compartment.  The internal fields will be set by
   the cell reader, so they do not need to be set here.  The
   make_cylind_compartment function is defined in compartments.g.
*/
make_cylind_compartment

/* Functions in hh_tchan.g create prototype H-H tabchannels
   "Na_hh_tchan" and "K_hh_tchan"
*/
make_Na_hh_tchan
make_K_hh_tchan

// Make a prototype excitatory channel, "Ex_channel" - from synchans.g
make_Ex_channel  /* synchan with Ek = 0.045, tau1 = tau2 = 3 msec */

// Make a prototype inhibitory channel, "Inh_channel"
make_Inh_channel /* synchan with Ek = -0.082, tau1 = tau2 = 20 msec */

/* Make a spike generator (spikegen) element "spike" - from
protospike.g */
make_spike

pope // Return to the original place in the element tree
```

Note the use of several statements to include the files compartments.g, hh_tchan.g, synchans.g, and protospike.g.

These files are not in the *cells/simplecell* directory, but are found in the *genesis/Scripts/neurokit/prototypes* directory. The GENESIS initialization file (*.simrc* in your home directory) sets the GENESIS search path (SIMPATH) to include this directory, so that these and many other files with prototype definitions can be accessed from any directory.

These files contain definitions of some global variables for channel reversal potentials and the like, plus function definitions that create the prototype elements. For example, *hh_tchan.g* sets some default values for the cell resting potential EREST_ACT, the sodium reversal potential ENa, and the potassium reversal potential EK. It also declares the functions *make_Na_hh_tchan* and *make_K_hh_tchan* to make the Na and K channel elements. The *protodefs.g* file assigns different

values to these variables after *hh_tchan.g* is included, but before the functions that create the prototype channels are called.

Usually the file containing the functions to create prototype channels will have some means of shifting the voltage scale for activation and time constant curves (minf and tau) by specifying a variable for the voltage offset in equations that depend on the membrane potential. Often the variable EREST_ACT is used for this purpose, as it can also represent the nominal resting potential of the cell for which the active channel models were developed. By using this as a global variable and changing it before calling the channel creation functions that are defined in the channel files, you can shift these curves. This can be very useful when you use a channel developed for one cell model in different cell.

WARNING: Note that EREST_ACT is also used in the cell parameter file to give Em and the starting Vm(initVm). And, if you include more than one file with functions to create channels, these files may set different values for EREST_ACT or the ionic reversal potentials. If this is the case, you should be careful to reset these variables to the desired values, **after** including the file, and **before** invoking the channel creation functions defined in that file.

Detour: Creating your own channel models

Creating the graphics

The file graphics.g can be used somewhat blindly, as long as you are happy with the default control panel and the graph that it creates. After including it, all you need to do is to invoke the *make_control* and *make_Vmgraph* functions and to send appropriate PLOT messages to the */data/voltage* xgraph element.

It is a good idea to keep the commands that involve graphics in a separate file, as we have done here. If you should later want to modify your simulation to run without graphics, as you might want to do when making long simulation runs on another networked computer, then you only need to make small changes to to the main simulation script. This will also allow you to easily add alternate Java-based user interfaces when GENESIS 3 is available.

In future tutorials, we will build upon the *simplecell* model scripts to create more detailed cell models. There is also a much fancier version of this simulation in the cells/simplecell2 directory. This implements the same model neuron, but provides a fancier graphical interface with controls to allow pulsed injection current, synaptic input from spike trains, and random Poisson-distributed background synaptic activation. It also provides user-defined string variables in the main script that you can change to use with different cell models.

If you need to modify this file for your own customized GUI, refer to the GENESIS Reference Manual section on the XODUS Graphical Interface, and the links given there for documentation for the XODUS "widgets" xbutton, xtoggle, xlabel, and xgraph. Chapter 14 of the BoG also explains some of the XODUS commands that were used to create a similar interface for tutorial3.g, and Chapter 22 gives a very detailed treatment of XODUS.

graphics.g

```
/*=============================================================
```

```
  A GENESIS GUI with a simple control panel and graph, with axis
scaling

=========================================================================*/

include xtools.g    // functions to make "scale" buttons, etc.

//================================
//      Function Definitions
//================================

function step_tmax
    step {tmax} -time
end

function overlaytoggle(widget)
    str widget
    setfield /##[TYPE=xgraph] overlay {getfield {widget} state}
end

//================================
//    Graphics Functions
//================================

function make_control
  create xform /control [10,50,250,180]
  create xlabel /control/label -hgeom 25 -bg cyan -label "CONTROL \
      PANEL"
  create xbutton /control/RESET -wgeom 33%       -script reset
  create xbutton /control/RUN  -xgeom 0:RESET -ygeom 0:label -wgeom \
      33% -script step_tmax
  create xbutton /control/QUIT -xgeom 0:RUN -ygeom 0:label -wgeom \
      34% -script quit
  create xdialog /control/Injection -label "Injection (amperes)" \
      -value 0.5e-9 -script "set_inject <widget>"
  create xtoggle /control/overlay   -script "overlaytoggle <widget>"
```

```
  setfield /control/overlay offlabel "Overlay OFF" onlabel "Overlay \
      ON" state 0
  xshow /control
end

function make_Vmgraph
    float vmin = -0.100
    float vmax = 0.05
    create xform /data [265,50,700,350]
    create xlabel /data/label -hgeom 10% -label "Soma contains Na \
        and K channels"
    create xgraph /data/voltage -hgeom 90% -title "Membrane \
        Potential" -bg white
    setfield ^ XUnits sec YUnits Volts
    setfield ^ xmax {tmax} ymin {vmin} ymax {vmax}
    makegraphscale /data/voltage
    xshow /data
end

function set_inject(dialog)
    str dialog
    setfield /cell/soma inject {getfield {dialog} value}
end
```

Some of the GENESIS features used in this file, and explained in the documentation links, are:

- The use of a function *overlaytoggle*, that uses an **xtoggle** widget to set the overlay field of all **xgraph** objects, so that a new graph can be plotted after a reset, without clearing the graph. This is used with the **xtoggle** element that is created in *make_control*, to toggle back and forth between overlay mode and non-overlay mode.

- The use of the wildcard expression "/##[TYPE=xgraph]" to mean any element in the element tree that is created from an **xgraph** object. The wildcard notation is explained in the GENESIS Reference Manual section on Hierarchical Structure.

- The use of the getfield command to return the value of a field of an **xtoggle**, **xdialog**, or any other element.

- The specification of position and dimensions [x, y, width, height] for **xforms**.

- The use of various ways of sizing and positioning of widgets, and setting labels and background colors. You may wish to experiment with these options.

- The use of a **xdialog** widget with a label, default value string, and function to be executed (the "-script" argument). Note the use of the shorthand "<widget>" to refer to the widget itself, which in this case is the xdialog */control/Injection*.

- The use of the XUnits and YUnits fields of an xgraph to add labels to graph X and Y axes.

- The use of the function *makegraphscale* (defined in the included file xtools.g) to create a "scale" button in the upper left corner of a graph, for changing axis scales.

An exercise (highly recommended):

Build some more cell simulations using the *graphics.g* file provided here, your own modified version of *simplecell.g*, and with the channel prototypes and cell parameter files found in some of the subdirectories of the cells directory. The *corticalcells* examples are a good place to start. The *traubcell* subdirectory has most of what you need to construct the 1991 Traub hippocampal CA3 region pyramidal cell model (see also Traub 1991). Each of these subdirectories of *cells* has a README file with further information.

For this, it would be best to create your own directory to which you will copy the files that you will modify for your simulation. For example, assuming that you want to create a subdirectory in your home directory called *newcell* and it doesn't already exist, you might do the following from within the *cells* subdirectory of the "GENESIS Tutorials" directory:

```
mkdir ~/newcell

cp simplecell/* ~/newcell

cp corticalcells/* ~/newcell

cd ~/newcell
```

You can then use your favorite text editor to modify any of these files.

Another exercise

The *genesis/Scripts/neurokit/prototypes* file yamadachan.g contains a function *make_KM_bsg_yka* to generate a Non-inactivating Muscarinic K current. This conductance was used in a model of a bullfrog stomatogastric ganglion cell, by Yamada, Koch, and Adams (1989).

This slow hyperpolarizing current is responsible for spike frequency adaption, i.e., after a current injection causes the cell to begin firing, the spiking rate decreases and reaches a steady slower rate, as in the plot shown below (Fig. 4).

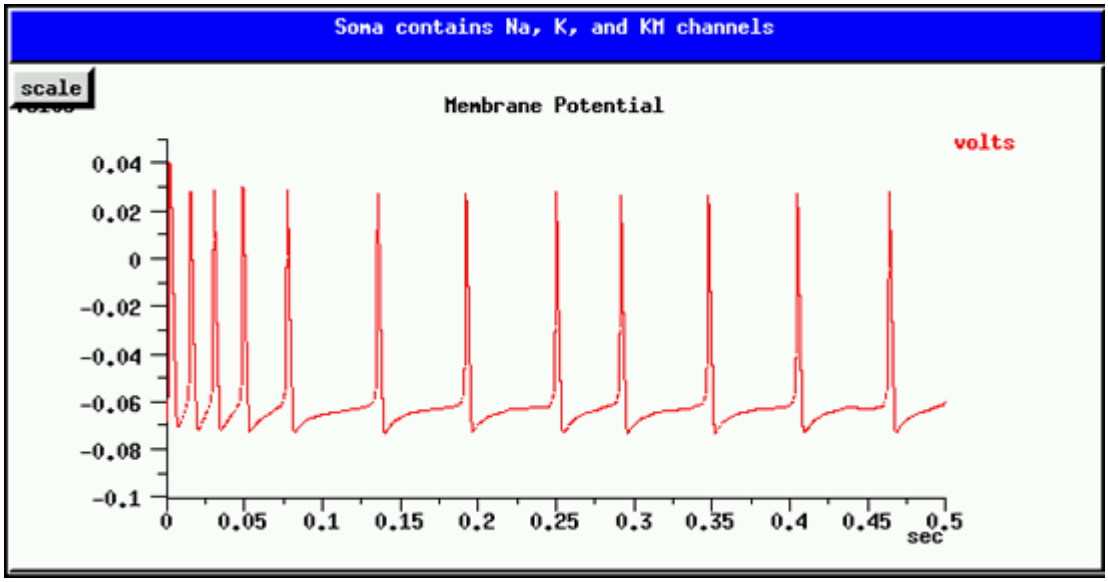

Figure 4: Spike frequency adaption. A current injection causes the cell to begin firing, the spiking rate decreases and reaches a steady slower rate.

Your task is to add a KM_bsg_yka to the simplecell model. Use the same K reversal potential EK as used by the K_hh_tchan channel. Chose conductance densities that achieve a result similar to the one above, when the injection current is 0.5 nA (as with the *simplecell* simulation). HINT: Keep the Na_hh_tchan maximum conductance density at its original value of 1200 S/m^2, but you will need to lower the K_hh_tchan maximum conductance considerably in order to compensate for the hyperpolarization contributed by KM_bsg_yka. If either of these potassium conductances are too large, the cell will not continue to fire. It will take a careful balance between them to produce the right result.

I give up -- tell me what conductance densities to use (EXERCISE ANSWER)**.**

Next, we will learn how to add synaptically activated channels and make synaptic connections, in order to build networks. At some point, you may want to come back and follow the

Detour: Making more realistic cell models

But, let's now move on to the next tutorial section Making synaptic connections so that we can get started on modeling neural circuits and networks.

Making synaptic connections

Adding synapses and providing synaptic input

We have already added an excitatory and an inhibitory synaptically activated channel to the */cell/dend* compartment, and a *spike* element to the soma, but haven't yet made any use of them.

Usually, we can treat an axon as a simple delay line for the delivery of spike events that last a single time step, as shown in Fig. 5. Only if we are interested in understanding the details of axonal propagation would it be necessary to model the axon as a series of linked compartments.

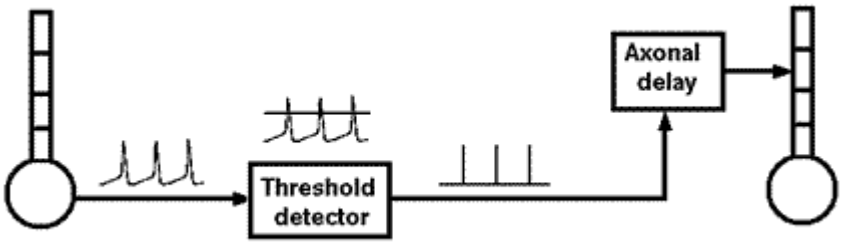

Figure 5: A threshold detector is used to convert the continuously varying action potentials to spike events that are propagated to a synapse after an appropriate axonal propagation delay. In GENESIS, the spikegen object provides the threshold detection, and the synchan object implements the delay.

The properties of an axon are split between two types of GENESIS objects. Spiking class elements (a *spikegen* or *randomspike*) create the spike events, either when Vm crosses a threshold during an action potential (*spikegen*), or as a random series of events generated at a specified average rate (*randomspike*). These send SPIKE messages to a *synchannel* class element (*synchan*, *hebbsynchan*, or *facsynchan*), which contains fields for the propagation delays and synaptic weighting for each synaptic connection.

For example, to send somatic action potentials in *cell1* to a *synchan* element "Ex_channel" in the dendrite compartment of *cell2*, you might use:

```
create spikegen /cell1/soma/spike
setfield /cell1/soma/spike thresh 0 abs_refract 0.005 output_amp 1
addmsg  /cell1/soma  /cell1/soma/spike  INPUT Vm
addmsg /cell1/soma/spike /cell2/dend/Ex_channel SPIKE
setfield /cell2/dend/Ex_channel synapse[0].weight 10 \
    synapse[0].delay 0.005
```

In this example, a spike is generated by the *spikegen* when the soma Vm exceeds the threshold value of 0. The absolute refractory period has been set to 0.005 (5 msec) in order to prevent multiple spikes from being generated during the time that Vm is above threshold. Normally, the field abs_refract will be set to something greater than the maximum width of the action potential at threshold, and less than the minimum expected interspike interval. Each time a new SPIKE message is added, it creates a new synapse within the *synchan*. Here, this synaptic connection is labeled as "synapse[0]", as it was the first (of possibly several) to be established with the SPIKE message.

In order to understand more about the use of these synaptically activated channels, you will need to read the documentation for Synaptic Connections in the GENESIS Reference Manual, and the documentation for synchan, spikegen, and randomspike. It will also be helpful to look at

*genesis/Scripts/neurokit/prototypes/*synchans.g in order to understand the properties of the channels *Ex_channel* and *Inh_channel* in */cell/dend*. The Advanced Tutorial on "*Simulating in vivo-like synaptic input patterns in multicompartmental models*" (Edgerton 2005, this volume) describes how realistic spike trains can be generated as probability distributions, or read from experimental data.

Then, try this simple exercise:

Modify the simplecell.g script to add a randomspike element with an average spike rate of 200 spikes per second. Connect it to */cell/dend/Ex_channel*, and set the soma current injection to zero. If you would like to play with XODUS graphics some more, plot the Ex_channel conductance Gk on another graph.

If you get stuck, look at tutorial4.g ("the hard way") or tutorial5.g (with *readcell*). In addition to the random spike input, these scripts illustrate the coupling of a cell's spike output to a *synchan*, by providing a feedback connection from the cell to itself. Chapter 15 of the BoG gives a detailed description of the steps in the construction of *tutorial4.g*.

Once you feel that you are ready, continue to the next section of this tutorial. This provides a more realistic exercise that connects two cells to each other to form a pattern generator circuit.

Building small networks and circuits

The goal of this exercise is to create a simple network of two cells that fire in alternate bursts. This will be made from two cells derived from the one created in the *simplecell* simulation, called */cell1* and */cell2*. After you feel that you understand *simplecell.g* and its included files, and have studied the documentation on the use of the *synchan* and *spikegen* objects, copy the *cells/simplecell* files into a directory of your own. Then, make the changes necessary to create a second cell with no current injection, and plot its Vm on the graph in a different color. Of course, the plot will be a flat line, as it is receiving no stimulus.

Then, use what you have learned about synaptic connections to connect the *spike* output of *cell1* to the excitatory **synchan** of *cell2*, and the *spike* output of *cell2* to the inhibitory input of *cell1*. Use an axonal propagation delay of 0.005 seconds for each connection. Finally, experiment with the synaptic weights for each synapse until you can achieve a pattern of alternate bursts of action potentials. To make it easy to change the weights, you may wish to add dialog boxes for entering weights to the control panel.

This approach may also be used to create larger networks. However, GENESIS has a number of commands that are intended specifically to create large arrays of cells and to connect them into a network, with just a few lines of scripting code. That is the subject of the next section.

Creating large networks with GENESIS

Now that we know how to make models of single neurons, and to make simple circuits of synaptically-connected neurons, it's time to get the next step in "modeling the brain" -- creating large networks of biologically realistic neurons, connected according to our best knowledge from physiology.

The procedure for constructing of large networks with GENESIS is covered in BoG Chapter 18. This chapter gives detailed descriptions for using the various options of the network creation commands that are summarized in the GENESIS Reference Manual section on Synaptic Connections. The WAM-BAMM 2005 Advanced Tutorial on *Constructing Large-scale Network Models* at http://www.wam-bamm.org/WB05/Tutorials provides some good advice on the issues encountered, and describes the process of constructing biologically realistic large networks of neurons. It uses examples taken from an improved "next-generation" model of the piriform cortex.

The chapter in the BoG uses examples from the venerable *genesis/Scripts/orient_tut* simulation, a very simple model of orientation selectivity, involving two layers of cells.

This tutorial uses a somewhat simpler example, consisting of a grid of simplified neocortical regular spiking pyramidal cells, each one coupled with excitory synaptic connections to its four nearest neighbors. This might model the connections due to local association fibers in a cortical network. The example simulation, in the *networks/RSnet* directory, was designed to be easily modified to allow you to use other cell models, implement other patterns of connectivity, or to augment with a population of inhibitory interneurons and the several other types of connections in a cortical network.

You may examine the cell model itself, and explore its response to different types of inputs by running and examining the scripts in the *cells/RScell* directory. This is a very simple one-compartment model that, like the exercise in “Building a cell the easy way”, uses a Muscarinic potassium current (KM) in order to achieve spike frequency adaption. This model, based on a paper and simulation by Destexhe et al. (2001), uses channels that give more realistic firing patterns than those in our exercise. The simplicity of this cell model allows our example network of 625 neurons to run fairly quickly.

But, it is important to note that single-compartment models with only these three ionic conductances have limitations. Although the KM current may play a role in spike frequency adaption of cortical pyramidal cells, the behavior of these cells is largely determined by calcium currents and at least two varieties of calcium-activated potassium currents. You may explore some more realistic cortical pyramidal cell models by running the simulations in the *cells/corticalcells* directory. The *genesis/Scripts/traub91* tutorial demonstrates the effects of these currents in burst-firing hippocampal pyramidal cells. For more on this subject, you can follow the detour:

Detour: Making more realistic cell models

The example simulation

Before we dissect the RSnet.g script, let's look at the simulation and its GUI. As with the scripts for *RScell* and *simplecell2*, the main script and the GUI (in graphics.g) were designed as fairly general templates that you can modify to experiment with your own network and cell models. It can be customized for another cell by changing strings that are defined in the main script.

When you run *RSnet.g* with the default parameters, you will see something like the display shown in Fig. 6:

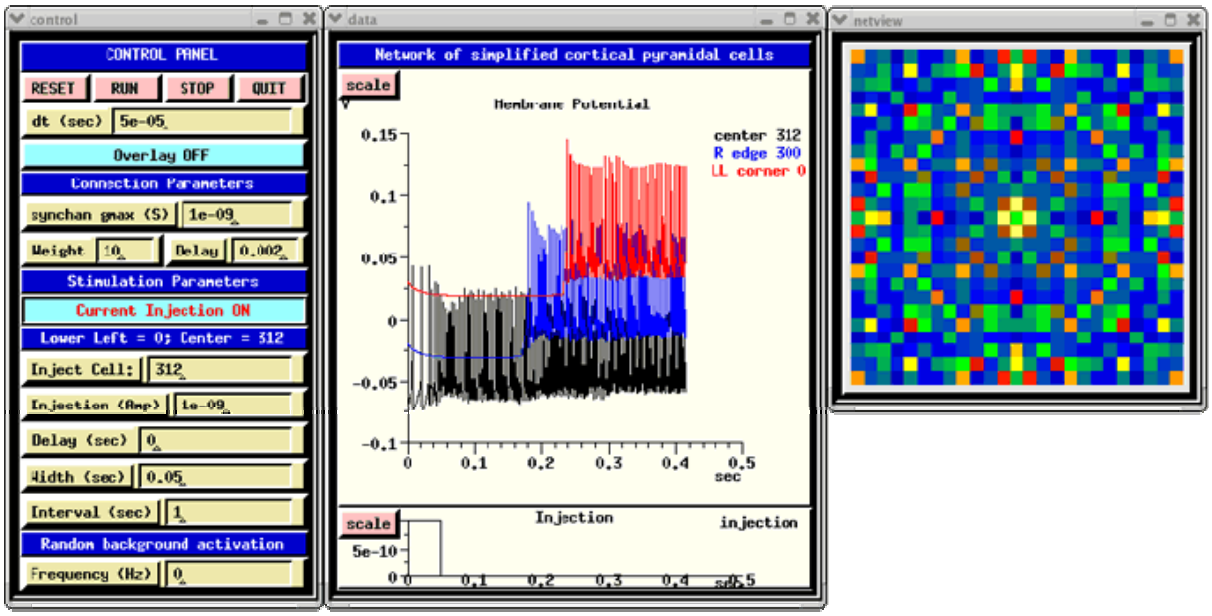

Figure 6: The *RSnet* simulation control panel (left) and the plot of membrane potential of cells at the center, right edge, and lower left corner of the network (middle). The display at the right shows the membrane potential in each cell of the network.

The CONTROL PANEL allows injection pulses to be applied to a selected cell in the network and/or random synaptic activation to be applied to each cell. The latter is done by setting the 'Frequency' dialog box to a non-zero value. This sets the frequency field of each synchan to the given value. The injection may be turned on and off by clicking on the 'Current Injection ON/OFF' toggle. Under 'Connection Parameters', 'synchan gmax' is used to set the gmax field of the synchan of each cell. The 'Weight' parameter acts as a multiplier of gmax for connections to the cells, but not for the random activation. Thus, the amplitude of the of the random synaptic input can be increased or decreased relative to the network synaptic input by appropriate scaling of 'synchan gmax' and 'Weight'.

The 'Delay' dialog is for setting the fixed axonal delay of each synchan to the same value. Comments in *RSnet.g* explain how to use a conduction velocity instead, to scale the delay according to the distance between cells.

You can explore the connections that are made by invoking the 'synapse_info' function at the genesis prompt. This function is defined, with further explanation, in the file *synapseinfo.g*, which is included by RSnet.g. For example,

```
genesis #5 > synapse_info /network/cell[312]/soma/Ex_channel
synapse[0] : src = /network/cell[287]/soma/spike weight =10 \
    delay=0.002
synapse[1] : src = /network/cell[311]/soma/spike weight =10 \
    delay=0.002
synapse[2] : src = /network/cell[313]/soma/spike weight =10 \
```

```
    delay=0.002
synapse[3] : src = /network/cell[337]/soma/spike weight =10 \
    delay=0.002
```

The 'Membrane Potenial' plot shows Vm for the center cell soma, and that for ones on the right edge and lower left corner. Each of these additional plots is displaced vertically by 0.5 V from the others, for easier viewing.

The function make_netview, defined in *graphics.g*, illustrates the use of the xview widget to display the Vm of each cell on the grid, using a cold to hot colorscale. To speed up the simulation somewhat, the line that invokes this function may be commented out.

Constructing the network

There are five steps to constructing the network. Each of these is described in a corresponding commented section of *RSnet.g*.

1. Create any prototype channels, compartments, etc. that will be used to build the cells.
2. Create a prototype cell, coupled to an *excitatory synchan* and a *spikegen*.
3. Use the createmap command to copy the prototype into a 2D array of cells.
4. Use the planarconnect command to connect each cell's *spikegen* to *synchans* on the four nearest neighbors.
5. Use the planardelay and planarweight commands to provide appropriate axonal delays and synaptic weights to the connections.

Here are the statements used in RSnet.g for each of these steps:

Step 1: Assemble the components to build the prototype cell under the neutral element */library*, all of this is done in the protodefs.g file, which is similar to those used in making the prototypes used for single cell models:

```
include protodefs.g  // This creates /library with the cell components
```

Step 2: Create the prototype cell specified in RScell.p, using *readcell*. Then, set the maximal conductance of the *excitatory synchan* in the appropriate compartment, and the threshold and absolute refractory period of the *spikegen* that is attached to the soma.

```
readcell RScell.p /library/cell
setfield /library/cell/soma/Ex_channel gmax {gmax}
setfield /library/cell/soma/spike thresh 0  abs_refract 0.004 \
    output_amp 1
```

In this case, the *synchan* is in the soma compartment, with the maximal conductance specified in the variable 'gmax'. A *spike* will be generated when Vm exceeds a voltage of zero, unless one has previously been generated within the last 4 msec.

Step 3: Make a two-dimensional array of cells with copies of */library/cell*.

```
createmap /library/cell /network {NX} {NY} -delta {SEP_X} {SEP_Y}
```

The usage of the `createmap` command is

```
createmap source dest Nx Ny -delta dx dy [-origin x y]
```

There will be NX cells along the x-direction, separated by SEP_X, and NY cells along the y-direction, separated by SEP_Y. The default origin is (0, 0). This will be the coordinates of cell[0]. The last cell, cell[{NX*NY-1}], will be at (NX*SEP_X -1, NY*SEP_Y-1).

Step 4: Now connect them up, using the planarconnect command. This command establishes synaptic connections between groups of elements based on the x-y positions of the elements. It does this by adding SPIKE messages between source and destination elements, using a large number of options to specify just which ones are to be included. Although this makes the syntax somewhat complex, it allows a wide variety of patterns of connections. The usage is of the form

```
planarconnect source-path destination-path
     [-relative]
     [-sourcemask {box,ellipse} xmin ymin xmax ymax]
     [-sourcehole {box,ellipse} xmin ymin xmax ymax]
     [-destmask   {box,ellipse} xmin ymin xmax ymax]
     [-desthole   {box,ellipse} xmin ymin xmax ymax]
     [-probability p]
```

These options are described in more detail in Chapter 18 of the BoG, and the documentation for `planarconnect`.

In this simulation, we want to connect each source spike generator to the excitatory *synchans* on the four nearest neighbors. To do this, we define the sourcemask to be a rectangle (box) with a very large range (-1 to +1 meters!), so that every cell in the network will be treated as a source. We want the destination, relative to the source to be an ellipse (or circle) that is large enough to include the four neighbors. It is generally a good idea to set the destmask ellipse axes or box size somewhat higher than the cell spacing, to be sure that the cells are included. Although this isn't a problem with our single-compartment cell, it can be an issue if the destination synapses are located in a distal dendrite compartment that is displaced by some amount from the cell origin. We also want to define a "destination hole" region that excludes the source cell, so that it doesn't connect to itself. This is implementented in *RSnet.g* with the statement:

```
planarconnect /network/cell[]/soma/spike
/network/cell[]/soma/Ex_channel \
-relative \ // Destination coordinates are measured relative to source
-sourcemask box -1 -1  1  1 \ // Larger than source area -> all cells
-destmask ellipse 0 0 {SEP_X*1.2}  {SEP_Y*1.2} \
-desthole box {-SEP_X*0.5} {-SEP_Y*0.5} {SEP_X*0.5} {SEP_Y*0.5} \
-probability 1.1 // set probability > 1 to connect to all in destmask
```

Note the use of the *wildcard notation* 'cell[]' to indicate all indices of the cell objects. Here, the '*desthole*' could just as well have been an ellipse. The variables SEP_X and SEP_Y had previously been set to the desired spacing between cells, 0.001 meters. To connect to nearest neighbors and the 4 diagonal neighbors, we would use a box for the destmask:

```
-destmask box {-SEP_X*1.01} {-SEP_Y*1.01} {SEP_X*1.01} {SEP_Y*1.01}
```

For all-to-all connections with a 10% probability, set both the sourcemask and the destmask to have a range much greater than NX*SEP_X using options

```
    -destmask box -1 -1  1  1
    -probability 0.1
```

Step 5: Set the axonal propagation delay and weight fields of the target synchan synapses for all spikegens, to the values previously defined for 'prop_delay' and 'syn_weight':

```
planardelay /network/cell[]/soma/spike -fixed {prop_delay}
planarweight /network/cell[]/soma/spike -fixed {syn_weight}
```

To scale the delays according to distance instead of using a fixed delay, use

```
planardelay /network/cell[]/soma/spike -radial {cond_vel}
```

and change the dialogs in *graphics.g* to set 'cond_vel'. This would be appropriate when connections are made to more distant cells.

Other options described in the documentation for planardelay and planarweight allow some randomized variations in the delay and weight, to make a more realistic simulation of a biological network. There are also three-dimensional equivalents to *planarconnect*, *planardelay*, and *planarweight*, called *volumeconnect*, *volumedelay*, and *volumeweight*.

Some other things to try

The example script in *genesis/Scripts/examples/fileconnect* gives an example of reading in a network connection matrix from a file with the fileconnect command.

If you would like to experiment with models having spike timing dependent plasticity, see the documentation for the *hebbsynchan* and facsynchan objects, and the examples in the GENESIS *Scripts/examples* directory.

More realistic cortical cell models tend to have a more pronounced hyperpolarization after the action potentials and a "more absolute" refractory period. This makes it possible to have propagating rings of activation generated by injection pulses, rather than continuous firing, as in this model. (For example, see Kudela et al. 1999). Modify the RSnet simulation to use the more detailed BDK5cell neocortical pyramidal model from the *cells/corticalcells* directory, and see if you can produce this effect. What effect does the propagation delay have on the waves?

Another 'exercise for the reader' would be to use the GENESIS parameter search routines to vary the *RScell* parameters, in order to create a cell that more closely duplicates the current injection behavior seen in a specific set of experimental data. Then compare the two models when used in a network.

Of course, a realistic cortical network will have a large number of inhibitory connections, mediated by interneurons that receive excitatory inputs and then make inhibitory connections to pyramidal cells. The lack of inhibition in this example network is responsible for the fact that, once a wave of excitation begins to propagate, the cells are firing near their maximum frequency and, as seen in the 'Membrane Potenial' plot, the amplitude of the action potentials is somewhat reduced because of this overstimulation. Inhibitory interneurons are generally of the "Fast Spiking" category, with little or no spike frequency adaptation, such as the simplecell model that we examined previously. Try adding a layer of these cells to the network, and make suitable excitatory connections to them from the RScells, and connections from them to inhibitory synchans in the RScells. For suggestions on possible "wiring diagrams" to use, see Douglas and Martin (1989), or Shepherd (1990).

What next?

Now you have the tools to begin "modeling the brain". The last section of this tutorial points you towards some information about other useful GENESIS features that we haven't yet discussed.

Where do we go from here?

Here are some suggestions and resources for learning more advanced GENESIS programming techniques.

Using implicit numerical methods

The default integration method (exponential Euler) is fine for simple models with just a few compartments. Models with many compartments should use an implicit method (e.g. Crank-Nicholson) with the Hines algorithm in order to avoid numerical instabilities. This is implemented, along with other speedups, in the GENESIS *hsolve* object. This is covered in BoG Chapter 20 and the GENESIS documentation for hsolve. For example scripts, see *genesis/Scripts/examples/hines* for examples.

Spike train objects in GENESIS

To understand single neuron computation it is desirable that realistic input patterns be given to model neurons in the study of the input-output function. Spike train objects can be used to generate input patterns in place of a full network model, which is often not available. Other GENESIS objects can be used to generate histograms of cross-correlations, auto-correlations, interspike intervals, and others. The Advanced Tutorial on *Simulating in vivo-like synaptic input patterns in multicompartmental models* (Edgerton 2005, this volume) describes how realistic input patterns can be given to model neurons, using the tools available in GENESIS.

Using GENESIS on parallel computers

Parallel GENESIS (PGENESIS) is an extension of GENESIS for use on parallel computers and networks of workstations. It is useful for simulations that must be run many times independently, such as parameter searching, and is used for large scale models that can benefit from the speed advantages of parallelism, especially large network models. The Advanced Tutorial on *Parallel GENESIS* (Hood 2005, this volume) presents in-depth example scripts, and discusses topics such as efficient network partitioning, synchronization issues, parallel I/O, parallel parameter searching, load balancing, scaling behavior, and debugging strategies. PGENESIS is also covered in considerable detail in BoG Chapter 21.

Other examples and GENESIS features

For performing parameter searches to "tune" a model, see *genesis/Scripts/param*, and the documentation for the GENESIS Parameter Search Library given in the GENESIS Reference Manual. The Advanced Tutorial on *Parameter Searching Tools in GENESIS* at http://www.wam-bamm.org/WB05/Tutorials gives a good overview of parameter searching and a discussion of the issues involved, suggestions, hints, and pitfalls.

A demonstration of the use of GENESIS for modeling biochemical reactions such as occur in biochemical signaling pathways can be found in *genesis/Scripts/kinetikit*, and in the Advanced Tutorial on *Modeling Calcium and Biochemical Reactions* (Blackwell 2005, this volume). BoG Chapter 10 provides an introduction to the biochemistry involved, and a tutorial on Kinetikit and the GENESIS kinetics library.

If you need to create your own new GENESIS objects and commands, see the documentation on Customizing GENESIS in the GENESIS Reference Manual.

The *genesis/Scripts/examples* directory has examples of other genesis capabilities such as Hebbian and facilitating synapses, Markovian channels, Ca diffusion in spines, and various types of device objects for input and output, or for applying stimuli to model neurons.

For a summary of all the objects that are available in GENESIS, see the Objects section in the GENESIS Reference Manual. To simply see a list of available objects, type "listobjects" from within GENESIS. To see a list of commands, type "listcommands".

Appendix: Detours

Detour 1: Building a cell without the cell reader

Chapter 14 of the BoG develops the script *tutorial3.g* to add Hodgkin-Huxley Na and K channels to the single soma compartment that was created in *tutorial2.g*. The newer version newtutorial3.g is similar, but uses the preferred **tabchannel** object.

In either case, the channel is created by calling a function in an included file, as described later in the "Building a cell the easy way" tutorial. However, instead of using the cell reader to put the channels in the right place in the element hierarchy and to connect them to the soma, the scripts use statements like:

```
// Create two channels, "/cell/soma/Na_squid_hh" and
"/cell/soma/K_squid_hh"

pushe /cell/soma

make_Na_hh_tchan

make_K_hh_tchan

pope

// The soma needs to know the value of the channel conductance

// and equilibrium potential in order to calculate the current

// through the channel.  The channel calculates its conductance

// using the current value of the soma membrane potential.

addmsg /cell/soma/Na_hh_tchan /cell/soma CHANNEL Gk Ek

addmsg /cell/soma /cell/soma/Na_hh_tchan VOLTAGE Vm

addmsg /cell/soma/K_hh_tchan /cell/soma CHANNEL Gk Ek

addmsg /cell/soma /cell/soma/K_hh_tchan VOLTAGE Vm
```

In tutorial4.g, developed in BoG Chapter 15, a dendrite compartment is created, and then connected to the soma with messages illustrated in Fig. A1, and the GENESIS statements

```
addmsg /cell/dend /cell/soma RAXIAL Ra previous_state

addmsg /cell/soma /cell/dend AXIAL  previous_state
```

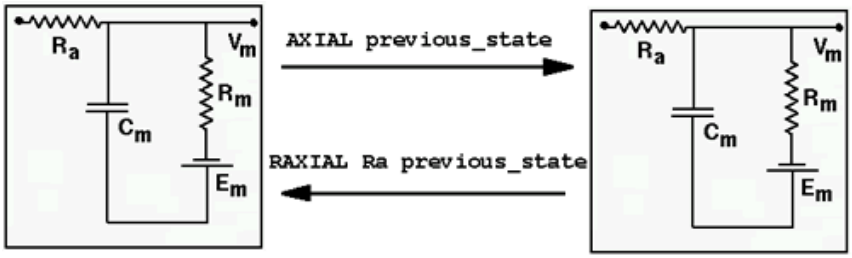

Figure A1: The messages and GENESIS commands that couple adjacent neural compartments. These allow the simulator to calculate the current flow between compartments.

In the first message, the dendrite compartment is linked to the soma with a message of the type RAXIAL, and a message link is established whereby two value fields, Ra and previous_state, will be sent from the dendrite to the soma at each simulation step. This allows the soma to calculate the current entering from the dendrite compartment. The previous_state field gives the value of the membrane potential at the previous integration step. We use this field rather than Vm because we want each compartment to update its data fields using data from the previous simulation step.

This establishes the information flow from the dendrite to the soma. In the reverse direction, the dendrite needs to receive the value of the soma's previous membrane potential in order to update its own state. (The dendrite already knows its own axial resistance to the soma, so the AXIAL message need not include information regarding axial resistance.)

The section of this tutorial on Making synaptic connections describes the spike generator that the cell reader adds to the soma. In tutorial4.g, this is accomplished with the statements:

```
// add a spike generator to the soma
create spikegen /cell/soma/spike
setfield /cell/soma/spike  thresh 0  abs_refract 0.010  output_amp 1
/* use the soma membrane potential to drive the spike generator */
addmsg  /cell/soma  {path}/soma/spike  INPUT Vm
```

The use of the **spikegen** object is described in the GENESIS Reference Manual section on Synaptic Connections and in the documentation for the spikegen object.

Detour 2: Creating and modifying channel models

You can find prototype definitions for many specific types of channels in the *genesis/Scripts/neurokit/prototypes* directory. The files in this directory, LIST and LIST.description, summarize the ones that are available.

Many of these prototype files make use of the variable EREST_ACT, which can be changed to another value, in order to shift the voltage dependence of the steady state activation and time constant up or down. For example, *hh_tchan.g* was designed for a mitral cell simulation with a resting potential of -0.06 volts. The simplecell simulation changed this to -0.07 volts for use in a cell that has a resting potential of -0.07 volts.

At some point, you may need to make more extensive changes in these scripts, or write your own. As a start, we will examine the K channel that is implemented with a **tabchannel** object by the *make_K_hh_tchan* function in *hh_tchan.g*. Older GENESIS simulations, such as *tutorial3.g*, implement this type of channel with the **hh_channel** object. We recommend that you use the faster and more versatile **tabchannel**, instead. *genesis/Scripts/tutorials/newtutorial3.g* shows how to use **tabchannels** instead of the **hh_channels** that are used in *tutorial3.g*.

The basic equations that determine the conductance of the K channel in the squid giant axon are:

$$G_K = \bar{g}_K n^4 \quad , \quad \frac{dn}{dt} = \alpha_n(V)(1-n) - \beta_n(V)\,n$$

$$\alpha_n(V) = \frac{0.01(10-V)}{exp\left(\frac{10-V}{10}\right)-1} \quad , \quad \beta_n(V) = 0.125\ exp(-V/80)$$

(3)

The simulator will solve the equation for *dn/dt*, so it is only necessary to specify the maximum conductance, represented by the field *Gbar* in the **tabchannel** object, the exponents for up to three gates (of which we only have one, *n*), and tables to represent the voltage dependence of the last two equations for the rate variables alpha and beta.

The relevant part of *hh_tchan.g* is

```
str chanpath = "K_hh_tchan"
create tabchannel {chanpath}
setfield ^ Ek {EK} Gbar {360.0*SOMA_A} Ik 0 Gk 0 \
Xpower 4 Ypower 0 Zpower 0
setupalpha {chanpath} X {10e3*(0.01 + EREST_ACT)} -10.0e3  \
    -1.0 {-1.0*(0.01 + EREST_ACT)} -0.01 125.0 0.0 0.0  \
    {-1.0*EREST_ACT} 80.0e-3
```

This sets the reversal potential Ek to the value previously assigned to the variable **Ek**, and the exponent for the *n* gate (represented by the X gate field of the **tabchannel**) to 4. As there is no inactivation or other gate, the exponents for the Y and Z gates are set to 0 (the default). *Gbar* will normally be set by the cell reader, but it is given the Hodgkin-Huxley value of 360 S/m^2 times the soma area, in case it is used without the cell reader, with an appropriate value of SOMA_A. Setting Ik and Gk is not really necessary, as they will be recalculated by the simulator.

The function *setupalpha* uses a generalized version of the Hodgkin-Huxley rate variables α and β, namely (A + B*Vm)/(C + exp((Vm + D)/F)), in order to set up the **tabchannel** tables. The 10 arguments correspond to the A, B, C, D, and F parameters for α and for β. A similar function, *setuptau*, allows this form to represent the voltage-dependent activation time constant and steady state activation, instead. A large percentage of published voltage-dependent Hodgkin-Huxley type channel models fit this general form. For the others, one has to fill the tables with either an equation evaluated in a loop, or a set of experimentally measured values. The documentation for tabchannel gives the details. It would also be useful to look at the documentation for setupalpha, setuptau, tweakalpha, and tweaktau.

Chapter 19 of the BoG covers the use of **tabchannels** to make calcium-dependent and other types of channels, using examples from the traub91chan.g and ASTchan.g prototype files. These files are extensively commented, and illustrate many of the ways to use a **tabchannel**. For Ca-dependent channels, be sure to read the documentation for the Ca_concen object.

Detour 3: Making more realistic (cortical) cell models

Why bother?

Jim Bower has discussed the value of structurally realistic modeling in his introductory remarks on WAM-BAMM and the modeling philosophy behind the GENESIS approach to modeling in: "*Looking for Newton: Realistic Modeling in Modern Biology*" (Bower 2005, this volume). You can also read his thoughts on choosing the level of detail to use in modeling in BoG Chapter 11.

The simple cell model that we have been using so far, described in Building a cell the easy way, fires at a steady rate, with equal intervals between spikes. With the addition of a non-inactivating muscarinic potassium current, it was possible to produce a non-uniform firing pattern with spike frequency adaptation. The RScell simulation in the *cells/RScell* directory is another simple one-compartment model that has a somewhat more realistic firing pattern. We might ask, how important is it to accurately reproduce the firing pattern of a typical pyramidal cell when picking a cell model to use in a cortical network? What is the effect of the spike latency and initial interspike interval (ISI) vs. the final ISI in determining the behavior of a network of neurons that display spike frequency adaptation? Is the RScell model good enough to use in a realistic network model?

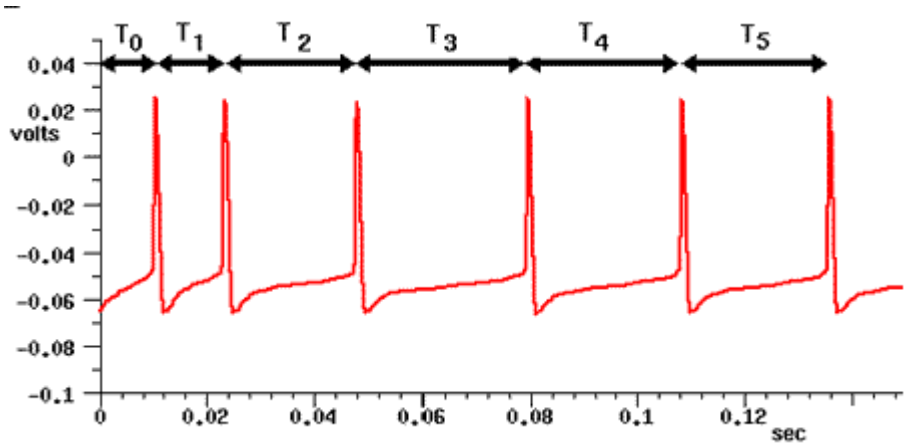

Figure A2: Spike frequency adaptation, showing the latency to the first spike T_0, and increasing time intervals between spikes.

Current clamp experiments on neocortical pyramidal cells often show results similar to the simulation results shown in Fig. A2 (generated from the detailed *BDK5cell* simulation in the *cells/corticalcells* directory). Is it necessary that a model neuron used in a large network accurately fit the timing of these action potentials? Or will the large variation in properties of individual neurons somehow "wash out" these details in the network, and allow us to use much simpler models? This is still an open question that may be answered by further modeling studies. However, there are indications that this variable spike timing can significantly affect network behavior.

Here T_0 is the spike latency, or time between the application of the injection pulse and the first spike. Under conditions of low excitation, this could act as an additional propagation delay, and affect the behavior of the network. The increasing interspike intervals T_1 - T_5 can also affect the behavior of the network. Under conditions of high excitation, when the neuron is firing continuously, the later ones will be more relevant than the early ones.

Spike frequency adaption may also be used as a mechanism for processing behaviorally relevant stimuli in the presence of many other sources of synaptic input. For example, Benda et al. (2005) have presented evidence that spike frequency adaption is used as a high pass filter to separate transient signals from slower oscillatory signals in the electrosensory system of weakly electric fish.

Building the model

The process of building a biologically realistic compartmental model of a neuron involves three steps:

1. Build a suitably realistic passive cell model, without the variable conductances. This subject is treated briefly in the *Introduction to Realistic Neural Modeling* section on *Constructing the passive cel*l (Beeman 2005), and in detail in the Advanced Tutorial on *Realistic Single Cell Modeling* (Jaeger 2005, this volume).

2. Add voltage and/or calcium activated conductances. See BoG Chapter 7 for an overview of the various types of ionic conductances, such as calcium conductances, calcium-activated potassium conductances, and inactivating potassium conductances, and how they affect firing properties

3. Tune the model to better fit passive properties and channel parameters that are known only approximately from experiment. Chapter 7 of the BoG describes how the cell and channel editing features of Neurokit may be used to perform manual parameter searches. The GENESIS Reference Manual section on the GENESIS Parameter Search Library and the example scripts in *genesis/Scripts/param* describe powerful methods for performing automated parameter searches in GENESIS. The Advanced Tutorial on *Parameter Searching Tools in GENESIS* at http://www.wam-bamm.org/WB05/Tutorials gives a good overview of parameter searching and a discussion of the issues involved, suggestions, hints, and pitfalls. Also see Vanier and Bower (1999).

The Advanced Tutorial on *Realistic Single Cell Modeling* (Jaeger 2005, this volume) examines the complete process of constructing a structurally realistic neuron model, using specific examples of modeling cerebellar neurons.

References

Beeman D (2005). Introduction to Realistic Neural Modeling. Brains, Minds & Media, this volume.

Benda J, Longtin A and Maler L (2005). Spike-frequency adaptation separates transient communication signals from background oscillations, J. Neurosci. 25: 2312-2321

Blackwell KT (2005). Modeling Calcium and Biochemical Reactions. Brains, Minds And Media, this volume.

Bower JM and Beeman D (1998). The Book of GENESIS: Exploring Realistic Neural Models with the GEneral NEural SImulation System, 2nd Edition, Springer-Verlag, New York.

Bower JM (2005). Looking for Newton: Realistic Modeling in Modern Biology. Brains, Minds and Media, Vol. 1, this volume.

Destexhe, A., Rudolph, M., Fellous, J. M. and Sejnowski, T. J. Fluctuating synaptic conductances recreate in-vivo-like activity in neocortical neurons. Neuroscience 107: 13-24 (2001).

Douglas RJ, Martin KAC, and Whitteridge D (1989). A canonical microcircuit for neocortex. Neural Computation 1: 480-488.

Edgerton J (2005). Simulating in vivo-like synaptic input patterns in multicompartmental models. Brains, Minds and Media, this volume.

Hood G (2005). Using P-GENESIS for Parallel Simulation of GENESIS Models: A Brief Overview. Brains, Minds and Media, this volume.

Jaeger D (2005). Realistic Single Cell Modeling. Brains, Minds and Media, this volume.

Kudela P, Franaszczuk PJ, and Bergey GK (1999). Model of the propagation of synchronous firing in a reduced neuron network. Neurocomputing 25-27: 411-418.

Koch C and Segev I (eds.) (1998), Methods in Neuronal Modeling, 2nd Edition, MIT Press, Cambridge, MA.

Shepherd, GM (1990). The Synaptic Organization of the Brain, 3rd edition, Oxford University Press, NY.

Traub RD, Wong RK, Miles R, and Michelson H (1991). A model of a CA3 hippocampal pyramidal neuron incorporating voltage-clamp data on intrinsic conductances, J. Neurophysiology, 66: 635-50.

Vanier, MC and Bower, JM (1999). A Comparative Survey of Automated Parameter-Search Methods for Compartmental Neural Models. J. Comput. Neurosci. 7: 149-171.

Yamada WM, Koch C, and Adams PR (1989). Multiple channels and calcium dynamics, in Koch C and Segev I (eds.): Methods in Neuronal Modeling, 1st edition, MIT Press.