Chapter 19
Implementing Other Types of Channels

DAVID BEEMAN

19.1 Introduction

So far, we have been using "squid-like" Hodgkin-Huxley channels for our voltage-dependent channels. In Chapter 7, you were introduced to a much wider variety of ionic conductances. There are scripts that contain functions for creating these and many other channel models in the *Scripts/neurokit/prototypes* directory. If you are constructing a realistic cell model and are lucky, you will find a function to create the channel you need in one of these scripts. Sometimes you may need to write your own function or, at least, modify a function that is similar to the one you need.

In Chapter 14, we learned how to implement voltage-dependent channels with the **hh_channel** object. This object is fairly easy to use, and has a straightforward correspondence between its internal fields and the parameters used in the Hodgkin-Huxley model. However, there are several good reasons for preferring another GENESIS channel object, the **tabchannel**. Although this object solves differential equations of the Hodgkin-Huxley form, given in Eq. 14.3, the rate parameters are provided by a table lookup, rather than from a fit to one of the three functional forms (Eqs. 14.4 – 14.6) used by the **hh_channel**. For further flexibility, GENESIS offers a related object, the **tab2Dchannel**, which uses two-dimensional tables.

Some channel models use expressions for the rate constants α and β that are not in one of these forms. Even if the rate constants can be expressed in this manner, it is much

faster to use a table lookup than to evaluate the functional form of the rate variable. Thus, a model that contains a great many channels will run much faster if it uses **tabchannels** or **tab2Dchannels**, rather than **hh_channels**. In addition, the **tabchannel** and the **tab2Dchannel** may be used with the fast implicit numerical integration methods discussed in Chapter 20. Not only will these methods further increase the speed of your simulation, but they are required in order to obtain numerically stable and accurate solutions for large cell models that contain many compartments. For these reasons, the **tabchannel** and the **tab2Dchannel** are the preferred objects to use for implementing voltage-dependent channels in large models.

In the following sections, we use the **tabchannel** to model channels from experimental data that have not been fitted to equations. We use a similar procedure to implement models for rate parameters that are not in one of the three standard forms (Eqs. 14.4–14.6). With the **tabchannel**, **tab2Dchannel** and some other GENESIS objects, we model channels that have a conductance that depends on the intracellular concentration of calcium ions. The **synchan** object, introduced in Chapter 15, provides a fairly general way to implement most synaptically activated channels. Towards the end of this chapter we discuss other approaches that may be used to implement NMDA channels, gap junctions, and dendrodendritic synapses.

In this chapter, we give examples of script language functions that may be used to create various types of channels. These are in a form suitable for use as prototypes with current versions of *readcell* (Chapter 16) and *Neurokit* (Chapter 17).

19.2 Using Experimental Data to Make a tabchannel

Smith and Thompson (1987) used voltage clamp experiments similar to those described in Chapter 4 to measure the characteristics of the slow inward tail current (I_B) that is found in bursting pacemaker cells of *Tritonia diomedia*. This current is often called the "B-current" because it is believed to be responsible for maintaining the prolonged depolarization that allows bursts of action potentials to occur. We will use results taken from this paper to construct a channel model, using the **tabchannel** object. This model was used in the simulation of the bursting molluscan neuron in Chapter 7. The GENESIS functions that implement this channel model and the others that are used in the simulation can be found in the script *ASTchan.g* in the *neurokit/prototypes* directory.

Figure 19.1A replots the Smith and Thompson data for the experimentally measured time constant (squares). As is typical with these sorts of measurements, there is a large amount of experimental uncertainty in these values, as well as variation from sample to sample. Therefore, we will want to draw a smooth curve to fit the data points, as shown in the dotted line, and take our values from this curve. Although a curve-fitting program could be used here, an "eyeball" fit by hand is adequate, considering the amount of noise in the

data. Ideally, we would like to have similar data for the steady-state value of the activation state variable, m_∞. As is often the case, we are only given the steady-state current

$$I_B = \overline{g}_B m^p (E_{rev} - V). \tag{19.1}$$

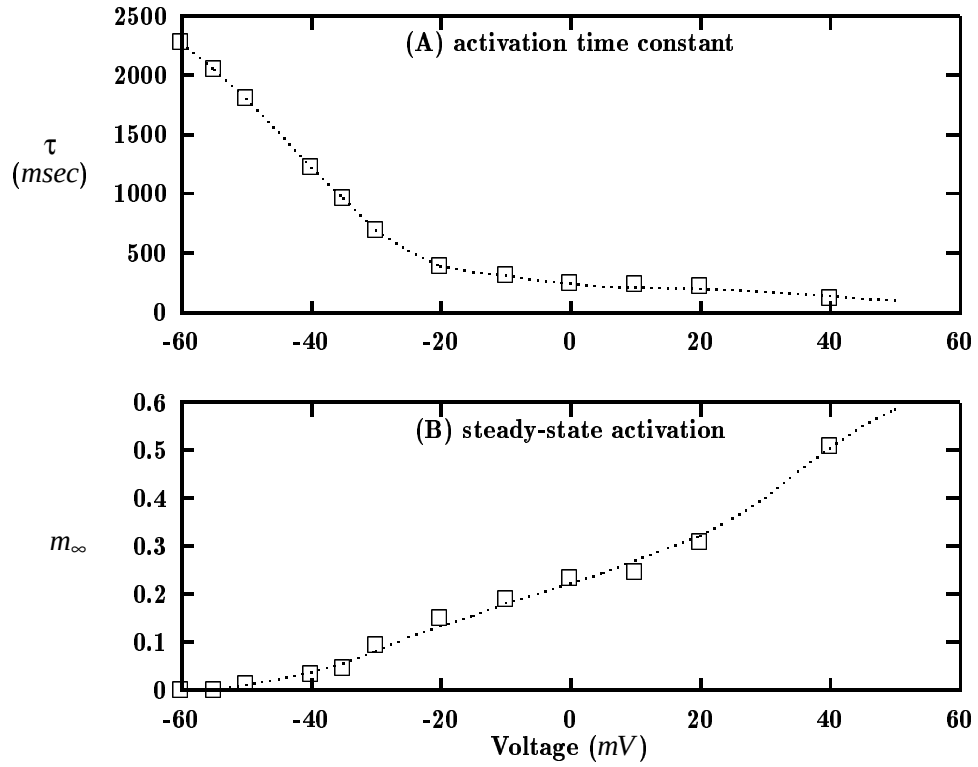

Figure 19.1 Data taken from Smith and Thompson (1987) for the B-current time constant (A) and steady-state activation (B). The squares represent experimental data and the dotted lines represent a fit to the data.

They estimated E_{rev} to be approximately 68 mV, but did not perform any fitting to determine the exponential p, or the maximum conductance $\overline{g}_B$. When fitting m_∞ to a sigmoid or other analytical function, it is customary to choose a value of the power p that gives the best fit, as discussed in Sec. 4.4.1 and illustrated in Fig. 4.5. In this case, we have used $p = 1$. The maximum conductance $\overline{g}_B$ is also unknown. Often, we can estimate it by using Eq. 19.1 to calculate and plot $\overline{g}_B m^p$ and making use of our expectation that m_∞ will asymptotically approach 1.0 at large values of V. This plot, shown in Fig. 19.1B, shows no sign of having

reached a maximum, although there is some hint of sigmoidal behavior. The best we can do is to guess that m has reached about half of its maximum value at 40 mV, and take $\overline{g}_B$ to be about 0.1 μS. This uncertainty will affect the scaling of the activation parameter, but will not affect any calculations using our channel model, as the channel current is the quantity of interest. By drawing a smooth curve through the data, as in Fig. 19.1B, we can estimate values of m_∞ to use in our channel model.

19.2.1 Setting the tabchannel Internal Fields

Like the **hh_channel**, the **tabchannel** has fields for *Ek* and *Gbar*, the activation state variables X and Y, and their associated exponents, *Xpower* and *Ypower*. There is an additional variable Z and exponent *Zpower* that may be used to provide calcium concentration-dependent activation or inactivation. Thus, the channel conductance is calculated from an equation analogous to Eq. 14.1,

$$Gk = Gbar \cdot X^{Xpower} Y^{Ypower} Z^{Zpower} \tag{19.2}$$

and the channel current is calculated from Eq. 14.2. Each of these three state variables obeys an equation of the form

$$\frac{dX}{dt} = A - BX. \tag{19.3}$$

In order to make the calculation more efficient, the notation is slightly different from that used in Eq. 14.3. A comparison of the two equations shows that $A = \alpha$ and $B = \alpha + \beta$. Each of the three gates (X, Y, and Z) can have associated tables to contain the voltage dependencies of the A and B variables.

We can illustrate the use of the **tabchannel** by constructing a script along the lines of *hhchan.g*, listed in Appendix B, which we used in Chapter 14. We will use the script to define a function to create a prototype channel that can be used with *Neurokit* and the cell reader, so the initial statements in the script will be fairly similar. Following the naming convention used with the other channel prototype scripts, we can call the channel *B_trit_st* and the function that creates it *make_B_trit_st*. As there is no inactivation, and we decided to let $p = 1$ in Eq. 19.1, the initial statements in our script should look something like

```
//genesis
float EB     = 0.068 // reversal potential in volts
float SOMA_A = 1e-9  // arbitrary value in sq m
function make_B_trit_st
    str chanpath = "B_trit_st"
    if (exists {chanpath})
        return
    end
```

```
create  tabchannel  {chanpath}
setfield  {chanpath}       \
     Ek     {EB}           \
     Gbar   {0.35*SOMA_A}  \
         Xpower 1          \
         Ypower 0          \
         Zpower 0
```

At this point, the *hhchan.g* script contains statements for setting the parameters used to calculate α and β. In our script, we will create tables for *A* and *B* and fill them with data.

The **tabchannel** and other GENESIS objects that make use of tables keep the tabular data in a data structure called an *interpol_struct*. With some minor variations, the procedures for manipulating and accessing the contents of the interpol_struct are similar for all these objects. The first step is to allocate space for the tables in the interpol_struct and to set the values of fields that specify the number of table divisions (*xdivs*), the *x*-value corresponding to the first entry in the table (*xmin*), and that of the last (*xmax*). This is done by invoking the object's TABCREATE action with the *call* command. For a **tabchannel**, this is done with a command of the form

```
call <path> TABCREATE <gate> <xdivs> <xmin> <xmax>
```

where the "gate" is *X*, *Y*, or *Z*. Before writing the rest of your script, it would be a good idea to experiment with the **tabchannel** by entering some commands interactively to the GENESIS prompt. Try giving the commands

```
create tabchannel /foo
call /foo TABCREATE X 30 -0.100 0.050
```

This will create both an *A* table and a *B* table for the *X* gate. The two tables will be named *X_A* and *X_B* and will have identical values of *xdivs*, *xmin*, and *xmax*. The indices of the entries run from 0 to *xdivs*, so the *xdivs* field is literally the number of *intervals* in the table. The number of *entries* is *xdivs* + 1. The following commands, which you should try for yourself, illustrate the notation used for accessing these fields and table entries.

```
showfield /foo X_A->xdivs
showfield /foo X_A->xmin
showfield /foo X_B->xmax
setfield /foo X_A->table[0] 2.27
setfield /foo X_A->table[30] 0.104
showfield /foo X_A->table[0]
showfield /foo X_A->table[30]
```

Once the *A* and *B* tables have been set up and filled with the proper values, a VOLTAGE message can be sent from a compartment to the channel, giving the membrane potential to be used to calculate the channel conductance. If the voltage is -0.1 or less, the *table[0]* values will be accessed. A voltage of 0.05 or greater will access the *table[30]* values.

In principle, one would use tabulated values of m_∞ and τ in order to calculate the values of *A* and *B* that go into the tables. These would be given by

$$A = \alpha = m_\infty/\tau \tag{19.4}$$

$$B = \alpha + \beta = 1/\tau. \tag{19.5}$$

However, GENESIS has a function, *tweaktau*, which allows us to fill the *A* table with τ and the *B* table with m_∞. After the tables are filled, we can use this function to "tweak" the tables by performing the calculations given in Eqs. 19.4 and 19.5. There is an analogous function, *tweakalpha*, which would let us fill the tables with α and β and then invoke the function to refill them with the proper *A* and *B* values. We can make use of the interpolation capability of GENESIS tabular objects in order to minimize the number of data points that we need to enter into the tables. Data points taken at 0.005 *V* intervals will give us a fairly smooth curve. Given the uncertainty of the experimental data, no further precision is justified. The range of -0.1 to 0.05 volts used in the example above covers the membrane potentials we would expect to find. Setting *xdivs* to 30 gives us the desired voltage increment. The data in Fig. 19.1A begin at -60 *mV*, where the smoothed value of τ is 2.27 *sec*. Rather than trying to extrapolate to lower voltages, we will fill the first eight entries of the *A* table with this value. This can be done by setting each of these table entries individually, but there is an easier way.

The *neurokit/prototypes/defaults.g* script, which is included when *Neurokit* is run, defines a GENESIS script function, *settab2const*, to do this sort of thing. If you are not using *Neurokit*, you may include *defaults.g* in your simulation, or copy the definition of *settab2const* into your own script. If we make use of this function, fill the tables with data from Fig. 19.1, and "tweak" the tables to convert the entries to those given by Eqs. 19.4 and 19.5, the remainder of our function definition will look something like

```
call        {chanpath}      TABCREATE X 30 -0.100 0.050 // in volts
settab2const {chanpath}  X_A  0  7  2.270
setfield {chanpath} X_A->table[8]  2.270 \  //  -60 mV
     X_A->table[9]  2.040  \ //  -55 mV
     X_A->table[10] 1.800  \ //  -50 mV
     .
     .
     X_A->table[29] 0.115  \ //   45 mV
     X_A->table[30] 0.104    //   50 mV
settab2const {chanpath}  X_B  0  8  0.0
```

```
    setfield {chanpath} X_B->table[9]  0.0 \  //  -55 mV
         X_B->table[10] 0.011 \  //  -50 mV
         X_B->table[11] 0.0224\  //  -45 mV
         .
         .
         X_B->table[29] 0.550 \ //   45 mV
         X_B->table[30] 0.585   //   50 mV
    tweaktau {chanpath} X
end
```

The GENESIS Reference Manual describes another alternative to filling the tables with these many *setfield* commands. The data could also have been entered into a file and read into the **tabchannel** tables with the *file2tab* command.

There is one last trick that we can apply to wring the most efficiency out of our **tabchannel** simulations. If we were to set *xdivs* to 3000 instead of 30, and were to fill the tables with 3001 interpolated values, we could do without interpolation while the simulation is being run. Although this would result in a slight increase in the time required to set up the simulation, the execution would be a lot faster. The **tabchannel** has a TABFILL action that can be used to perform the table expansion and interpolation with the single command

```
call {chanpath} TABFILL X 3000 0
```

The last argument specifies that "fill mode" 0 (interpolation and smoothing with B-splines) will be used to fill the tables. Other options are to interpolate with cubic splines (mode 1) or to use linear interpolation (mode 2, the default). The interpol_struct has a *calc_mode* field that can be set to determine whether or not interpolation will be used for a specified table. The following statement will set the mode to 0 (no interpolation) for the two tables

```
setfield {chanpath} X_A->calc_mode 0 X_B->calc_mode 0
```

These two statements should be added just before the final "end" statement in the function *make_B_trit_st*.

19.2.2 Testing and Editing the Channel

We should now test our channel model. The easiest way to do this is to make a simple cell containing this channel and use the *Neurokit* `edit channel` menu option to examine the channel properties. As we will only be looking at the channel itself and don't care very much about the properties of the cell, we can modify any handy cell parameter file and *userprefs.g* file to construct the cell. An easy choice is to start with the *cell.p* file that was used in Chapter 17. The only thing we need to do to the *cell.p* file is to add the name of the channel *B_trit_st* and an arbitrary channel density to the line that specifies the contents

of the soma compartment. The *Na_squid_hh* and *K_squid_hh* channels can either remain or be deleted. Although this is not a reasonable thing to do in the context of this model, put a *B_trit_st channel* in the dendrite compartment, also. This will make it possible to experiment with separately modifying the behavior of the two B current channels. The only thing that needs to be done to the *userprefs.g* file is to add an *include* statement for the file containing the function *make_B_trit_st* and to invoke the function.

Once you have copied these files into the directory that contains your script for *make_B_trit_st* and have edited them to your satisfaction, load the model into *Neurokit*, following the procedure described in Section 17.4. Instead of choosing the `run cell` option from the title bar, click on `edit channel`. When the Cell Window appears in the lower left portion of the screen, click on the green spherical soma in order to select it. It should turn red, and icons representing the soma, the *B_trit_st* channel, and any other channels that are in the soma will appear in the Compartment Window at the upper left. Click on the *B_trit_st* channel in this window.

The Channel Parameters Window will appear in the lower right portion of the screen, below the Compartment Window. Figure 19.2 shows the group of dialog boxes, buttons, and toggles that are below the graphs in this window. The `gate` dialog box in this window will show the default gate to be examined, the *X* gate. As this is the one we want, position the cursor in this box and hit "Return". If all has gone well, the two lower graphs will show plots for τ and m_∞ that agree with those in Figures 19.1A and 19.1B.

Figure 19.2 The control panel for the *Neurokit* "Channel Parameters Window".

After making any corrections that are needed in order to reproduce the experimental data, we can try out some of the channel editing features. These will allow you to modify the characteristics of **tabchannels**, without having to edit the scripts that create them. The lower row of dialog boxes allows you to enter parameters for scaling or offset of any of the rate or state parameters.

We will begin by applying a vertical offset to uniformly increase the value of τ. Change the `oy` dialog box to read "0.5" and click on the button labeled `tau` to the right of the `MODIFY:` label. Notice that τ has been replotted and that the values have been shifted up by 0.5 *sec*.

Now, click on the dendrite compartment in the Cell Window. The Compartment Window should now show the dendrite, with the *B_trit_st* channel still selected. Hit "Return" in the `gate` dialog box, or click anywhere inside the box. Observe the resulting plot of τ for this gate. Apparently, the copy of the channel in the dendrite compartment has also

been modified. As they say in the software industry, this is a "feature," not a "bug." In a large compartmental model or large network, one may have many copies of a particular prototype channel. Usually, one wants these to behave identically. It is also desirable to minimize the amount of storage space used by the internal tables. For these reasons, copies of the channel that are created by the *copy* command or by *readcell* use the same tables as the original prototype, rather than new copies of the tables. This is true of all objects that contain tabular fields.

You may reverse this change by changing `oy` to -0.5 and clicking on `tau` again. Note that the scale and offset is always relative to the last operation, and not to the original values. Therefore, it is a good idea to set the dialog boxes back to offsets of 0.0 and scaling of 1.0 after making a change. This will prevent you from inadvertently making further changes.

Sometimes you may want to change just the one copy of the channel. GENESIS has a function that allows you to duplicate a **tabchannel** gate and its tables before modifying it. To try this, click on `Dup Gate` and repeat the experiment. You might also try using the `sy` to scale the values of τ, m_∞, α, or β. To recover the old value, use the reciprocal of the previous scale factor.

m_∞ becomes non-zero shortly above $-0.055\ V$. Suppose that we would like to give I_B a more rapid onset by shifting this point downward to $-0.065\ V$. Start by clicking on `Dup Gate`, so that we won't change the dendrite channel or the prototype in */library*. Then set the `ox` dialog to -0.01 and click on the `m_inf` button. After you have studied the results, reverse this change by using an offset of 0.01. Then, repeat this a few times, shifting the curve to higher voltages each time you click on `m_inf`. Try restoring the original curve by changing `ox` to -0.01 and repeatedly clicking on `m_inf`. Do you see a problem?

One can reverse changes in `oy` and `sy`, because they just shift and scale the table values. However, `ox` and `sx` perform offsets and scaling of the horizontal axes by moving data in the tables. This can cause data to spill out of the ends of the tables and be lost. Thus, large changes in the *x*-axis should be avoided. Fortunately, there is a button labeled `Restore`. If you had the foresight to duplicate the gate, clicking on this button will restore the original from the prototype in */library*. If not, the prototype will have also been messed up and you will need to reload the cell. If you have the time, you might try another experiment with changing the *x*-axis. τ drops rapidly between -60 and $-20\ mV$. Suppose that we would like to decrease the slope. We can do this by spreading the voltage scale out a bit. Experiment with the `sx` dialog to do this.

Most of these buttons and dialog boxes have a direct correspondence with GENESIS functions that you can use in your own simulation scripts. The scaling and offsets can be provided with the *scaletabchan* command. Bring the terminal window with the GENESIS prompt to the foreground and try the command "`scaletabchan -u`". Like most commands that require additional arguments, this will result in a "usage" message. In this case, you should see

```
usage: scaletabchan channel-element gate mode sx sy ox oy [-duplicate]
Valid modes : a[lpha] b[eta] t[au] m[inf].
```

Now try

```
ce /cell/soma
scaletabchan B_trit_st X tau 1.0 1.0 0.0 0.5 -d
```

Put the terminal window into the background again so that you can see the Channel Parameters Window and click on the `gate` label of the dialog box that contains "`X`". Notice that this produces the same results as if you had performed the vertical offset to τ by using the dialog boxes and `tau` button. After having "fine-tuned" your channel from within *Neurokit*, you may wish to permanently include these changes in your model. One way would be to edit your scripts that create the channels, changing the data that goes into the tables. An easier solution would be to use the original channel creation functions and to then use the *scaletabchan* function to perform the desired scaling and offset operations. If you are using scripts from the *Neurokit* prototypes library, or would like to use slightly different versions of the same channel in different simulations, it will be most convenient to use the *scaletabchan* function in your *userprefs.g* file, after you have created the prototype channels. Of course, you will probably not want to use the "duplicate" option of the function, as we did in the example above.

The `Dup Gate` button invokes the *duplicatetable* command for both the internal tables of the specified channel and gate. If you like, you may try it out on one of the tables by giving the following commands.

```
pushe /cell/soma
duplicatetable B_trit_st X_A
showfield B_trit_st X_A->table[900]
setfield B_trit_st X_A->table[900] 0.5555
showfield B_trit_st X_A->table[900]
pope /cell/dend
showfield B_trit_st X_A->table[900].
```

There is yet another way in which offsets and scaling may be provided to a **tabchannel** or **tabgate**. The **tabchannel** has fields *ox*, *oy*, *sx*, and *sy* for the *A* and *B* tables of each gate. These can be set with statements of the form

```
setfield B_trit_st X_A->ox 0.02.
```

This method is less convenient for modifying α, β, τ, or m_∞, because it can only be used to change the scaling of the *A* and *B* tables. Note that changing *A* is not the same as changing α, because $B = \alpha + \beta$ will still use the old α value. However, this approach

has some advantages when the voltage scales for both τ and m_∞ are to be modified equally. Unlike the *scaletabchan* function, setting the *ox* and *sy* fields of a table simply changes the values of *xmin* and *xmax*, without moving data within the table. This procedure is not suitable for changing the scale of a single rate or state parameter, but can be used if both the *A* and *B* *x*-axes are modified equally. The *ox* field is particularly useful if one wants to adapt a channel for use in another cell that has a significantly different resting potential.

19.3 Using Equations for the Rate Constants

Often, you will be lucky enough to find a channel for which someone has already analyzed the experimental data and fitted the rate constants α and β to some function. This is the case for the channels that are used in the tutorial on the bursting hippocampal neuron in Chapter 7. The examples given here and in the next section are taken from the script *traub91chan.g* in the *neurokit/prototypes* directory. The functions that are defined in this script implement the channel models described in the paper by Traub et al. (1991).

The high voltage-activated calcium channel model used in this paper is typical of one that we might want to convert to a GENESIS **tabchannel**. In the notation of the paper, the current contributed by the channel is given by

$$I_{Ca} = \overline{g}_{Ca} s^2 r (V - V_{Ca}). \tag{19.6}$$

The Hodgkin-Huxley rate constants that give rise to the activation variable *s* and the inactivation variable *r* have been fitted to the expressions

$$\alpha_s = \frac{1.6}{1 + \exp(-0.072(V - 65))} \tag{19.7}$$

$$\beta_s = \frac{0.02(V - 51.1)}{\exp\left(\frac{V-51.1}{5}\right) - 1} \tag{19.8}$$

$$\alpha_r = \begin{cases} 0.005 & [V \leq 0] \\ \frac{\exp(-V/20)}{200} & [V > 0] \end{cases} \tag{19.9}$$

$$\beta_r = \begin{cases} 0 & [V \leq 0] \\ 0.005 - \alpha_r & [V > 0]. \end{cases} \tag{19.10}$$

Although Eq. 19.6 uses an opposite convention for the direction of positive current flow than that used in Eq. 14.2, there is a straightforward correspondence between the variables used in Eq. 19.6 and the **hh_channel** and **tabchannel** field variables that are used in Eqs. 14.1 and 14.2. However, this paper uses the physiological units listed in Table 14.1, rather than SI units. With some care, we could also use physiological units in our channel model, but for consistency we will stick to SI units. In addition to converting the voltages

from *mV* to volts, we will have to scale the rate parameters by a factor of 1000, in order to make the conversion from inverse *msec* to inverse seconds. In the paper, all voltages are expressed with respect to a resting potential that is defined to be 0 *mV*, corresponding to an actual potential of -60 *mV* relative to the outside of the cell. We can make use of a GENESIS global variable, *EREST_ACT*, to give us some more flexibility in our function to create the channel. If we replace V by $V - EREST\_ACT$ in Eqs. 19.7–19.10, we may use the channel with any assumed resting potential. In the script *traub91chan.g* and the example given below, *EREST_ACT* is set to -0.06.

We would expect our function that creates this channel to use a *for* loop to fill each table. Because of the piece-wise continuous form of the inactivation rate constants, this will be necessary. However, GENESIS has a command, *setupalpha*, that can be used to create and fill the *A* and *B* tables when a gate has rate constants of the general form

$$y = \frac{A + Bx}{C + \exp((x + D)/F)}. \tag{19.11}$$

This is the case for Eqs. 19.7–19.8, so we will set up the tables for the *X* gate with a command of the form

```
setupalpha chan gate AA AB AC AD AF BA BB BC BD BF
```

where the parameters *AA*–*AF* correspond to those in Eq. 19.11 when it represents $\alpha(V)$ and the parameters *BA*–*BF* correspond to those for $\beta(V)$. The *setupalpha* function then performs many of the operations given in the example above for the *B_trit_st* channel. Namely, it

- calls the TABCREATE action for the specified gate (*X*, *Y*, or *Z*), setting *xdivs* to 49, *xmin* to –0.1, and *xmax* to 0.05,
- loops over the *xdivs* + 1 table entries, using Eq. 19.11 to fill the *A* table with values for $\alpha(V)$ and the B table with values for $\beta(V)$,
- "tweaks" the *B* table, so that it contains $\alpha(V) + \beta(V)$,
- calls the TABFILL action to expand the table to 3000 entries, and sets the *calc_mode* field for each table to "no interpolation."

Other optional parameters let you change the range and sizes given above.

Figure 19.3 shows a function that will create the channel *Ca_hip_traub91* using *setupalpha* to create the *X* (activation) gate. The *Y* (inactivation) gate and the *Y_A* and *Y_B* tables are created "the hard way" using the steps itemized above. As it is easy to go astray when converting between different units, you should make sure that you can reconcile Eqs. 19.7–19.8 with the values of the parameters used with *setupalpha*.

```
float EREST_ACT = -0.060 /* hippocampal cell resting potl */
float ECA = 0.140 + EREST_ACT // 0.080 volts
float SOMA_A = 3.320e-9       // soma area in square meters
function make_Ca_hip_traub91
    if ({exists Ca_hip_traub91})
        return
    end
    create  tabchannel  Ca_hip_traub91
    setfield            ^       \
        Ek              {ECA}   \               // V
        Gbar            { 40 * SOMA_A } \       // S
        Xpower  2       \
        Ypower  1       \
        Zpower  0
    setupalpha Ca_hip_traub91 X 1.6e3  \
        0 1.0 {-1.0 * (0.065 + EREST_ACT) } -0.01389 \
        {-20e3 * (0.0511 + EREST_ACT) }  \
        20e3 -1.0 {-1.0 * (0.0511 + EREST_ACT) } 5.0e-3
    float   xmin = -0.1
    float   xmax = 0.05
    int     xdivs = 49
    call Ca_hip_traub91 TABCREATE Y {xdivs} {xmin} {xmax}
// Fill Y_A table with alpha and Y_B table with (alpha+beta)
    int i
    float x,dx,y
    dx = (xmax - xmin)/xdivs
    x = xmin
    for (i = 0 ; i <= {xdivs} ; i = i + 1)
        if (x > EREST_ACT)
            y = 5.0*{exp {-50*(x - EREST_ACT)}}
        else
            y  = 5.0
        end
        setfield Ca_hip_traub91 Y_A->table[{i}] {y}
        setfield Ca_hip_traub91 Y_B->table[{i}] 5.0
        x = x + dx
    end
    setfield Ca_hip_traub91 Y_A->calc_mode 0   Y_B->calc_mode 0
    call Ca_hip_traub91 TABFILL Y 3000 0
end
```

Figure 19.3 An example script containing a function to create the high voltage-activated calcium channel.

19.4 Implementing Calcium-Dependent Conductances

Much of the interesting behavior of the bursting neurons discussed in Chapter 7 arises from potassium channels that have conductances depending on the intracellular concentration of calcium ions. In this section, we look at ways to obtain the calcium concentration from the calcium channel current, and examine the implementation of two types of calcium-dependent currents.

19.4.1 Calculating the Calcium Concentration

Several processes affect the concentration of calcium ions in a neural compartment (Yamada, Koch, and Adams 1989, Sala and Hernández-Cruz 1990). After calcium ions enter the compartment through calcium-selective ionic channels, they may diffuse into neighboring compartments. They may also bind to various buffers, such as the protein, calmodulin. Specialized ionic channels act as pumps to extrude calcium ions from the cell. GENESIS allows you to model one-dimensional diffusion of calcium with the **difshell** object. Non-mobile buffers can be simulated with the **fixbuffer**, and diffusible buffers with the **difbuffer** object. Two types of calcium pumps can be modeled, which can be made electrogenic. The **mmpump** object models a Ca-ATPase pump obeying Michaelis-Menten kinetics (Sec. 10.2.1), and the *setupNaCa* command uses a **tabcurrent** object to model the Na^+-Ca^{2+} exchanger current. The **tabcurrent** object allows you to model any non-ohmic currents, and it can also be used to compute the solution to the Goldman-Hodgkin-Katz equation, for which the appropriate tables are set up with the *setupghk* command. The mathematical equations implemented by these objects for calcium dynamics can be found in De Schutter and Smolen (1997). Examples of the use of these objects can be found in the spines tutorial (based on a model by Holmes and Levy (1990)) and the Purkinje cell tutorial. These, and other recently developed tutorials are available through the GENESIS Users Group (see Appendix A).

Often it is sufficient to use a simpler model, or there arenot enough experimental data to adequately model the processes described above. The two simulations described in Chapter 7 represent the rate of change of the concentration of calcium ions by the equation

$$\frac{d[Ca^{2+}]}{dt} = BI_{Ca} - \frac{[Ca^{2+}]}{\tau}. \tag{19.12}$$

Here, $[Ca^{2+}]$ is the concentration, which we express in the SI units of moles per m^3. This conveniently works out to be the same in milli-moles per liter. Note that the commonly used *Molar* concentration is expressed in moles per liter. Thus, the typical resting intracellular Ca^{2+} concentration of 50 *nM* would be 50×10^{-6} in our units. The first term on the right gives the rate of increase of $[Ca^{2+}]$ due to an inward channel current I_{Ca}. The second represents an attempt to fit the various processes that deplete $[Ca^{2+}]$ to an exponential decay

with a single time constant τ. An estimate of the value of τ may often be obtained by optical measurements of the rate of decay of $[Ca^{2+}]$, using Ca-sensitive dyes. In principle, one may find the constant B by calculating the flux of ions into a thin shell near the membrane surface produced by I_{Ca}, as in Exercise 3. This calculation leads to the result that for a shell of surface area A and thickness d,

$$B = 5.2 \times 10^{-6}/(Ad), \tag{19.13}$$

where the shell dimensions are given in meters and the current is in amperes. However, there are factors that will modify this value, causing it to be just a rough approximation. Buffering ties up a large percentage of the entering Ca^{2+}, often reducing the effective value of B by a factor of 100 or more. The calcium concentration will not be uniform throughout the compartment, and the concentration in the shell close to the membrane surface is most relevant for behavior of channels that are activated or inactivated by the presence of calcium ions. As it is difficult to estimate an appropriate value to use for the thickness of the shell, d becomes yet another free parameter to be fitted. For the model of the bursting molluscan neuron of Chapter 7, the best available estimate of τ was used, and B was chosen so that Eq. 19.12 would yield maximum values of $[Ca^{2+}]$ which were in agreement with typical experimental measurements of the intracellular concentration of free Ca^{2+}.

GENESIS has an object, **Ca_concen**, which solves Eq. 19.12 for $[Ca^{2+}]$. There are data fields *B*, *tau*, and *Ca* for the corresponding quantities in Eq. 19.12. The calcium current appearing in the equation, I_{Ca}, is provided by summing current values that are provided by "I_Ca" messages from all channels which carry calcium currents. In addition, there is a *Ca_base* field that is added to the resulting value of *Ca* in order to provide a "base" or resting value of the concentration.

In Chapter 16, Sec. 16.3 discusses the channel "density" parameter that is used in the cell descriptor file. When the "channel" that is specified in this file is the name of a prototype library element formed from a **Ca_concen** object, this value of the "density" is used to calculate the value of the *B* field, ignoring the value that was set when the prototype was created. This is analogous to the procedure used to set the *Gbar* field of a **tabchannel**. For **Ca_concen** objects, *B* is set to the "density" divided by the volume of the parent compartment, with the volume calculated in m^3 from the dimensions given (in microns) in the cell descriptor file. By taking the actual compartment volume into account and setting an appropriate density, you can then set *B* for each compartment to be whatever you want. The object has an additional field *thick*, for the shell thickness d that appears in Eq. 19.13. If d is non-zero, the shell area A is calculated from the compartment dimensions, and *B* is set to the "density" divided by the volume Ad. Thus, the cell reader can also scale *B* as for a true shell.

The following statements define a function using the **Ca_concen** object to calculate the concentration of calcium ions resulting from a current flow through the *Ca_hip_traub91*

channel.

```
function make_Ca_hip_conc
    if ({exists Ca_hip_conc})
            return
    end
    create Ca_concen Ca_hip_conc
    setfield Ca_hip_conc \
            tau     0.01333   \      // sec
            B       17.402e12 \      // Curr to conc for soma
            Ca_base 0.0
    addfield Ca_hip_conc addmsg1
    setfield Ca_hip_conc \
            addmsg1        "../Ca_hip_traub91 . I_Ca Ik"
end
```

In their model of the CA3 pyramidal cell, Traub et al. expressed $[Ca^{2+}]$ in arbitrary units with a resting concentration of 0, current in μA, and set $\tau = 13.33$ *msec*. If we use the same concentration units, but express current in amperes and τ in seconds, our *B* constant is then 10^{12} times the constant (called ϕ) used in the paper. The actual value of *B* used for each compartment will typically be determined by the cell reader from the cell parameter file. However, for the prototype channel we will use Traub's value for the soma. (A misprint in the paper gives it as 17,402 rather than 17.402.) In our units, this is 17.402×10^{12}.

The *Ca_hip_conc* element created by this function should receive an "I_Ca" message from the calcium channel, accompanied by the value of the calcium channel current. If this message were stated explicitly in a simulation script, it would be

```
addmsg {path}/Ca_hip_traub91 {path}/Ca_hip_conc I_Ca Ik
```

Here, the variable *path* indicates the location of these two elements in the hierarchy. For example, *path* might evaluate to "*/cell/soma*". However, we will ordinarily use the cell reader to create copies of these prototype elements in one or more compartments. We need some way to be sure that the needed messages are established. Although the cell reader has enough information to create the messages that link compartments to their channels and to other adjacent compartments, it must be provided with the information needed to establish additional messages. This is done by using the *addfield* command to place the message string in a user-defined field of one of the elements that is involved in the message. In our case, we use the statements

```
addfield Ca_hip_conc addmsg1
setfield Ca_hip_conc \
    addmsg1        "../Ca_hip_traub91 . I_Ca Ik"
```

The cell reader recognizes the added fields *addmsg1*, *addmsg2*, etc. as indicating that they are to be evaluated and used to set up messages. The paths are relative to the element that contains the added message. Thus, "`../Ca_hip_traub91`" refers to the sibling element *Ca_hip_traub91* and "`.`" refers to the *Ca_hip_conc* element itself. Here we have chosen to attach the message to *Ca_hip_conc*. If we had attached it to *Ca_hip_traub91* instead, it would have been

```
addfield Ca_hip_traub91 addmsg1
setfield Ca_hip_traub91 \
    addmsg1         ".  ../Ca_hip_conc I_Ca Ik"
```

We will use the value of $[Ca^{2+}]$ produced by *Ca_hip_conc* to influence the conductance of two different types of potassium channels. Another use of the **Ca_concen** object would be to use the resulting concentration to change the value of an ionic equilibrium potential in situations where large changes in concentration can modify the driving force on the ions. The **nernst** object, which is described in the GENESIS Reference Manual, typically receives the intracellular ionic concentration with a CIN message and calculates the resulting equilibrium potential using the Nernst equation. The **tabchannel**, **tab2Dchannel** and **vdep_channel** objects (described below) can receive the new value of *Ek* from the **nernst** object with an EK message so that this field will be continually updated.

19.4.2 The AHP Current

The AHP current (Sec. 7.6) is a slow potassium current that depends only on $[Ca^{2+}]$. When time is expressed in seconds, the model used in the paper gives a rate constant α that increases linearly from 0 to 10 as $[Ca^{2+}]$ increases from 0 to 500. When $[Ca^{2+}]$ is greater than 500, in these arbitrary units, α has a constant value of 10. The β parameter has a constant value of 1.0 over the entire range of $[Ca^{2+}]$.

This concentration-dependent activation can be modeled with the **tabchannel** *Z* gate. The *Z* gate acts just like the *X* and *Y* gates, except that it gets its input value from a CONCEN message, instead of a VOLTAGE message. The parameter that is sent is usually an ionic concentration, coming from a **Ca_concen** object. (There is no reason that this message couldn't be used to send a voltage or other variable, however.)

The *traub91chan.g* script defines a function to create the channel *Kahp_hip_traub91*. This is very similar to the function shown in Fig. 19.3, which creates the *Ca_hip_traub91* channel. However, the exponents *X power* and *Y power* are set to zero, and *Z power* is set to 1. A *for* loop is used to fill the *A* and *B* tables for the *Z* gate and the table is expanded with TABFILL, as before. In addition, the cell reader needs to be given the information necessary for it to set up a CONCEN message from the *Ca_hip_conc* element. This is accomplished with the statement

```
addfield Kahp_hip_traub91 addmsg1
setfield Kahp_hip_traub91 \
    addmsg1        "../Ca_hip_conc . CONCEN Ca"
```

As with the added field that was defined for the *Ca_hip_conc* element, the string assigned as the value of *addmsg1* is used only by the cell reader.

19.4.3 The C-Current

The Z gate can be used only if the conductance has a factor Z^{Zpower}, where Z obeys Hodgkin-Huxley rate equations like those for X and Y, but with α and β being functions of concentration. However, the concentration dependence is not always of this form. The C-current (Chapter 7) is a fast potassium current that depends on both voltage and the calcium concentration. The model used by Traub, et al. (1991) for the conductance has a typical voltage and time dependent activation gate, where the time dependence arises from the solution of a differential equation containing the rate constants α and β. It is multiplied by a function of calcium concentration that is given explicitly, rather than being obtained from a differential equation. Therefore, we need a way to multiply the activation by a concentration-dependent value that is determined from a lookup table.

GENESIS doesn't have a way to implement this with a **tabchannel**, so we use the **vdep_channel** object here. These channels contain no gates and get their activation gate values from external gate elements, via a MULTGATE message. These gates are usually created with **tabgate** objects, which are similar to the internal gates of the **tabchannels**. However, any object that can send the value of one of its fields to the **vdep_channel** can be used as the gate. Here, we use the **table** object. This generality makes the **vdep_channel** very useful, but it is slower than the **tabchannel** because of the extra message passing involved. The following function illustrates the steps needed to implement this channel.

```
float EREST_ACT = -0.060 /* hippocampal cell resting potl */
float EK = -0.015 + EREST_ACT // -0.075
float SOMA_A = 3.320e-9       // soma area in square meters
function make_Kc_hip_traub91
    if ({exists Kc_hip_traub91})
        return
    end
    create  vdep_channel    Kc_hip_traub91
    setfield            ^       \
        Ek              {EK}    \               //      V
        gbar            { 100.0 * SOMA_A }      //      S
    float   xmin = 0.0
    float   xmax = 1000.0
    int     xdivs = 50
```

```
    create table            Kc_hip_traub91/tab
    call Kc_hip_traub91/tab TABCREATE {xdivs} {xmin} {xmax}
    int i
    float x,dx,y
    dx = (xmax - xmin)/xdivs
    x = xmin
    for (i = 0 ; i <= {xdivs} ; i = i + 1)
        if (x < 250.0)
            y = x/250.0
        else
            y = 1.0
        end
        setfield Kc_hip_traub91/tab table->table[{i}] {y}
        x = x + dx
    end
    setfield Kc_hip_traub91/tab table->calc_mode 0
    call Kc_hip_traub91/tab TABFILL 3000 0
// Now make a tabgate for the voltage-dependent activation parameter.
    float   xmin = -0.1
    float   xmax = 0.05
    int     xdivs = 49
    create  tabgate         Kc_hip_traub91/X
    call Kc_hip_traub91/X TABCREATE alpha {xdivs} {xmin} {xmax}
    call Kc_hip_traub91/X TABCREATE beta  {xdivs} {xmin} {xmax}
    int i
    float x,dx,alpha,beta
    dx = (xmax - xmin)/xdivs
    x = xmin
    for (i = 0 ; i <= {xdivs} ; i = i + 1)
        if (x < EREST_ACT + 0.05)
            alpha = {exp {53.872*(x - EREST_ACT) - 0.66835}}/0.018975
            beta = 2000*{exp {(EREST_ACT + 0.0065 - x)/0.027}} - alpha
        else
            alpha = 2000*{exp {(EREST_ACT + 0.0065 - x)/0.027}}
            beta = 0.0
        end
        setfield Kc_hip_traub91/X alpha->table[{i}] {alpha}
        setfield Kc_hip_traub91/X beta->table[{i}] {beta}
        x = x + dx
    end
    setfield Kc_hip_traub91/X alpha->calc_mode 0 beta->calc_mode 0
    call Kc_hip_traub91/X TABFILL alpha 3000 0
    call Kc_hip_traub91/X TABFILL beta  3000 0
    addmsg Kc_hip_traub91/tab  Kc_hip_traub91 MULTGATE output 1
    addmsg Kc_hip_traub91/X  Kc_hip_traub91  MULTGATE m 1
```

```
    addfield Kc_hip_traub91 addmsg1
    addfield Kc_hip_traub91 addmsg2
    setfield Kc_hip_traub91 \
            addmsg1         "../Ca_hip_conc  tab INPUT Ca" \
            addmsg2         ".. X  VOLTAGE Vm"
end
```

First, we create the **vdep_channel**, *Kca_hip_traub91*, and set the usual fields. Then, we create the lookup table for the function of concentration, $f([Ca^{2+}]) = \min(1, [Ca^{2+}]/250)$. It is made from a **table** object, and filled in a manner similar to that used for the internal tables of the **tabchannel** object. Note that the internal field for the table is called *table*. When expanding the table so that it may be used in the "no interpolation" calculation mode, note that the TABFILL syntax is slightly different from that used with **tabchannels**. Here, there is only one internal table, so the table name is not specified.

Next, we make a **tabgate** for the voltage-dependent rate constant for activation. The **tabgate** has two internal tables, *alpha* and *beta*. These are filled like those of the **tabchannel**. Note that the *beta* table is really β, not $\alpha + \beta$, as with the *B* table of the **tabchannel**. Finally, we set up the required messages. The MULTGATE message is used to give the **vdep_channel** the value of the activation variable and the power to which it should be raised. As we have created the **tabgate** and **table** as subelements of the channel, they and their messages to the channel will accompany it when copies are made. However, we also need to provide for messages that link to external elements. The message that sends the Ca^{2+} concentration to the **table** and the one that sends the compartment membrane potential to the **tabgate** are stored in added fields of the channel, so that they may be found by the cell reader.

Later in this chapter, we will see how the **tab2Dchannel** allows the efficient modeling of more complex relationships between the channel conductance and V_m and $[Ca^{2+}]$, and how future versions of the **tab2Dchannel** might be used to implement this channel model without the use of the **vdep_channel**.

19.4.4 Other Uses of the table Object

The **table** object is quite versatile. There is also a similar **table2D** object that has an internal two-dimensional table. These may both be used in many situations where there is no GENESIS object that will perform a given calculation, or as a faster alternative to another object. For example, Section 19.4.1 mentioned the use of the **nernst** object to continually update the equilibrium potential of a **tabchannel**. The script *mitproto.g* in the *neurokit/prototypes* directory shows how this may also be accomplished with a **table**. The *Cable* tutorial uses this object as an intermediary to the **xgraph** object in order to generate logarithmic plots. In Chapter 22 (Sec. 22.4.2) we will see an example of its use as a function generator. Another use of the **table** would be to implement an "instantaneous" gate that has an activation

depending only on voltage, rather than having a time and voltage dependence given by the solution of Eq. 19.3. You can find further examples in the *Scripts/examples/table* directory.

19.4.5 The vdep_gate Object

For completeness, we should mention the **vdep_gate** object, which may also be used with a **vdep_channel**. This gate is very much like the **tabgate**, except that it calculates the rate constants α and β directly from Eq. 19.11, instead of from internal tables. Although this form is slightly more general than the three forms used by the **hh_channel**, an **hh_channel** implementation will execute somewhat faster than the **vdep_channel** and **vdep_gate** combination. If it is necessary to use separate channel and gate objects, as with the C-current, a combination of a **vdep_channel** and **tabgate** will give the most speed and flexibility. Thus, there is little reason to use the **vdep_gate**. It is seen mainly in older GENESIS simulations that were written before the development of the **tabchannel** and **tabgate** objects.

19.4.6 Using the tab2Dchannel Object

Some channel models require rate constants that depend on another variable in addition to V_m. For example, Moczydlowski and Latorre (1983) have described the kinetics of a calcium-activated potassium channel that has been widely used as a model for the C-current. Unlike the current described in Sec. 19.4.3, the dependence of the conductance on $[Ca^{2+}]$ arises not from a multiplicative function of $[Ca^{2+}]$, but from the dependence of α and β on both V_m and $[Ca^{2+}]$.

In GENESIS, such models may be implemented with the **tab2Dchannel**, which contains two-dimensional tables for the variables *A* and *B*. Like the **tabchannel**, and unlike the **vdep_channel** or **hh_channel**, the **tab2Dchannel** may be used with the fast and highly accurate implicit numerical integration methods that are described in Chapter 20. The listing of the script *MoczydKC.g* in Fig. 19.4 reveals some of the significant features of the **tab2Dchannel**, and allows us to compare it with the **tabchannel**.

In comparing this listing with that in Fig. 19.3, we see that most of the channel fields are the same. However, the TABCREATE action now takes additional arguments *ydivs*, *ymin* and *ymax* in order to allocate the two-dimensional tables for *A* and *B*. The tables now have two indices, where the first one runs from 0 to *xdivs* and the second one from 0 to *ydivs*.

When using two-dimensional tables, it may be necessary to experiment with the table size in order to obtain the desired accuracy without using an excessively large table. The setup time needed to fill the tables using a *for* loop is also a consideration, as there are presently no utilities like the *setuplapha* command for filling **tab2Dchannel** tables. (But, check the GENESIS Reference Manual for new developments.)

When using a one-dimensional table, it is customary to use a large table, either by setting *xdivs* to a large value, or by using TABFILL to expand the table with interpolated

```
/* non-inactivating BK-type Ca-dependent K current
** Moczydlowski and Latorre 1983, J. Gen. Physiol. 82:511-542.
** Implemented by Erik De Schutter BBF-UIA,
** with original parameters scaled for units: V, sec, mM */
float EK = -0.085   // volts
function make_Moczyd_KC
    int xdivs = 100
    int ydivs = {xdivs}
    float xmin, xmax, ymin, ymax
    xmin = -0.1; xmax = 0.05; ymin = 0.0; ymax = 0.0030
    int i, j
    float x, dx, y, dy, a, b
    float Temp = 37
    float ZFbyRT = 23210/(273.15 + Temp)
    if (!({exists Moczyd_KC}))
        create tab2Dchannel Moczyd_KC
        setfield Moczyd_KC Ek {EK} Gbar 0.0  \
            Xindex {VOLT_C1_INDEX} Xpower 1 Ypower 0 Zpower 0
        call Moczyd_KC TABCREATE X {xdivs} {xmin} {xmax} \
            {ydivs} {ymin} {ymax}
    end
    dx = (xmax - xmin)/xdivs
    dy = (ymax - ymin)/ydivs
    x = xmin
    for (i = 0; i <= xdivs; i = i + 1)
        y = ymin
        for (j = 0; j <= ydivs; j = j + 1)
            a = 480*y/(y + 0.180*{exp {-0.84*ZFbyRT*x}})
            b = 280/(1 + (y/(0.011*{exp {-1.00*ZFbyRT*x}})))
            setfield Moczyd_KC X_A->table[{i}][{j}] {a}
            setfield Moczyd_KC X_B->table[{i}][{j}] {a + b}
                y = y + dy
        end
        x = x + dx
    end
    setfield Moczyd_KC X_A->calc_mode {LIN_INTERP}
    setfield Moczyd_KC X_B->calc_mode {LIN_INTERP}
    addfield Moczyd_KC addmsg1
    setfield Moczyd_KC addmsg1 "../Ca_conc . CONCEN1 Ca"
end
```

Figure 19.4 An example script using a **tab2Dchannel** to implement the Moczydlowski and Latorre (1983) Ca-dependent K current.

values. Then, as was done in Fig. 19.3, the *calc_mode* field for each table is set to zero (*NO_INTERP*), in order to save computation time. In the present example, the *calc_mode* is set to *LIN_INTERP* (a predefined global variable equal to one), so that linear interpolation is performed at run time, allowing the use of a smaller table.

This script uses a maximum value of $[Ca^{2+}]$ (*ymax*) of 0.003 *mM* (milli-moles per liter), because it was intended to be used with a Ca channel and associated **Ca_concen** element that produced concentrations in this range. The channel parameters were fitted to experimental data for a range of concentrations up to 10 *mM*, so your value of *ymax* could be much larger. As always, it is important to take some care in choosing the value of the parameter *B* in Eq. 19.12 and the corresponding field in the **Ca_concen** element, and then determine the maximum value of $[Ca^{2+}]$ that the **Ca_concen** element will generate for your particular model.

As with the **tabchannel**, a message carrying the membrane voltage or a concentration is sent to the channel so that the channel can retrieve the appropriate *A* and *B* table values to calculate the gate activations (X, Y and Z) and the resulting channel conductance. However, we can now have two messages, in order to specify both the *x* and *y* variables. There are two new messages for sending concentrations (or anything else), CONCEN1 and CONCEN2. Another message, DOMAINCONC, provides a highly simplified model to obtain the ionic concentration directly, using the current sent from another channel and the surface area of the parent compartment (De Schutter and Smolen 1997). There are also three new fields *Xindex, Yindex* and *Zindex*. These fields are used for each gate to define which message refers to the *x* variable and which refers to the *y* variable. These index fields may each be assigned to one of the predefined global variables *VOLT_INDEX*, *C1_INDEX*, *C2_INDEX*, *DOMAIN_INDEX*, *VOLT_C1_INDEX*, *VOLT_C2_INDEX*, *VOLT_DOMAIN_INDEX*, *C1_C2_INDEX* and *DOMAIN_C2_INDEX*.

The first four of these are used when a gate depends on only one variable. In this case, *xdivs* should be set to zero for that gate, and the *y* variable (corresponding to the second index) is used to fill the *A* and *B* tables. Then, the prefix (VOLT, C1, C2 or DOMAIN) specifies whether the VOLTAGE, CONCEN1, CONCEN2, or DOMAINCONC message is used to provide the *y* variable. The remaining five of these variables are of the form *x_y_INDEX*, and similarly specify which of two messages are used to specify the *x* and *y* variables.

In our example, *Xindex* is set to *VOLT_C1_INDEX*. This means that a VOLTAGE message will specify the *x* variable of the *X_A* and *X_B* tables, and a CONCEN1 message will specify the *y* variable. The cell reader will automatically provide the VOLTAGE message from the parent compartment. However, as with the C-current model described in Sec. 19.4.3, we need to use an added field *addmsg1* to hold a string that tells the cell reader to create a CONCEN1 message from a sibling **Ca_concen** element (which we have called *Ca_conc*) to our channel, giving the Ca concentration. If our model required a second gate *Y* that depended solely on another ionic concentration, we could also send a CONCEN2

message, and set *Yindex* to *C2_INDEX*.

Future versions of the **tab2Dchannel** will allow the *Xindex, Yindex* and *Zindex* fields to take on values to indicate that the corresponding gate should be given a value instantaneously calculated from the *A* table values, rather than from the solution of Eq. 19.3. This will allow the fast and efficient modeling of "instantaneous" channels and C-current models like those described in Sec. 19.4.3 without the use of **table** or **vdep_channel** objects.

19.5 NMDA Channels

Postsynaptic receptors that are activated by N-methyl-D-aspartate (NMDA) have characteristics that differ significantly from other receptors that give rise to EPSPs. NMDA channel conductances have rise times on the order of 5–10 *msec* and decay slowly over periods of 70–100 *msec*. In addition to showing this prolonged conductance, NMDA channels have a voltage-dependent conductance that increases with depolarization from the resting potential. As a significant synaptic current requires both a presynaptic input and a postsynaptic depolarization, NMDA synapses can act like Hebbian synapses, showing a use-dependent facilitation. NMDA channels pass a combination of sodium, potassium and calcium ions. Experiments have shown that mechanisms for long term potentiation (Sec. 15.4) depend upon a postsynaptic increase in the intracellular Ca^{2+} concentration. This increase is thought to be brought about by calcium influx through NMDA channels. For these reasons, there has been a great deal of interest in the modeling of NMDA channels.

The voltage dependence of NMDA channels is caused by a blockage of the channel by magnesium ions. At membrane potentials on the order of -70 *mV*, the driving force for these large ions to try to enter the channels is quite high. As the membrane is depolarized, the blockage is relieved. The behavior of the magnesium block has been modeled by Jahr and Stevens (1990). Zador, Koch and Brown (1990) have used this model and data from hippocampal neurons to fit the NMDA channel conductance to

$$g_{syn}(V,t) = g_n \frac{\exp(-t/\tau_1) - \exp(-t/\tau_2)}{1 + \eta[Mg^{2+}]\exp(-\gamma V)}. \tag{19.14}$$

In this expression, the constants $g_n = 0.2$ *nS*, $\tau_1 = 80$ *msec*, $\tau_2 = 0.67$ *msec*, $\eta = 0.33/mM$, and $\gamma = 0.06/mV$. Typical values of extracellular $[Mg^{2+}]$ in hippocampal slice experiments are about 2 *mM*. The ratio of sodium, potassium and calcium ions passed by this channel is such that the reversal potential is close to the resting potential.

The time dependence of Eq. 19.14 is the same as that of the dual exponential form of the **synchan** object (Eqs. 6.17 and 15.2). In GENESIS, we can implement NMDA channels by multiplying the conductance of a **synchan** by a voltage-dependent term which is provided by another object, the **Mg_block**. This object has a field *CMg* that corresponds to $[Mg^{2+}]$, a

field *KMg_A* corresponding to $1/\eta$, and a field *KMg_B* corresponding to $1/\gamma$. Note that any units may be used for the concentrations *CMg* and *KMg_A*, as long as they are consistent.

The following statements illustrate the use of a **synchan** and **Mg_block** object to implement an NMDA channel.

```
    float CMg = 2                       // [Mg] in mM
    float eta = 0.33                    // per mM
    float gamma = 60                    // per Volt
    create      synchan               {compartment}/{channel}
    setfield         ^ \
                Ek                      {Ek} \
                tau1                    {tau1} \
                tau2                    {tau2} \
                gmax                    {gmax}
    create Mg_block {compartment}/{channel}/block
        setfield         ^ \
                CMg                     {CMg} \
                KMg_A                   {1.0/eta} \
                KMg_B                   {1.0/gamma}
    addmsg   {compartment}/{channel} {compartment}/{channel}/block \
                CHANNEL Gk Ek
    addmsg   {compartment}/{channel}/block {compartment} CHANNEL Gk Ek
    addmsg   {compartment}   {compartment}/{channel}/block VOLTAGE Vm
// Even though we don't use the channel current, CHECK expects this message
    addmsg   {compartment}   {compartment}/{channel} VOLTAGE Vm
```

After creating the two elements and setting the fields for the various constants that appear in Eq. 19.14, we set up a CHANNEL message to pass the unmodified conductance from the **synchan** to the **Mg_block**. Another CHANNEL message passes the blocked conductance to the parent compartment from the **Mg_block**. The **Mg_block** needs the compartment membrane potential to calculate the voltage-dependent factor in Eq. 19.14, so it gets it via a VOLTAGE message. The **synchan** expects to receive a VOLTAGE message in order to calculate a channel current. Although the value of the current in the absence of blocking is of no interest to us, we send the message anyway, in order to avoid error messages produced by the CHECK action of the channel.

19.6 Gap Junctions

In addition to communicating via chemically activated synapses, neurons may communicate through direct electrical connections. These *gap junctions*, which are common in lower vertebrates and found in several sites of the mammalian brain, are large macromolecules that extend through the membranes of both cells. Pores in these molecules allow the rapid

exchange of ions between the two neurons without the delays that occur with synaptic coupling. As they offer a very low resistance with little leakage to the extracellular space, the postsynaptic potential is nearly the same as the presynaptic potential.

The speed of this type of coupling makes it useful for modifying the behavior of networks of coupled oscillators. Studies of the pyloric network of the lobster stomatogastric ganglion have shown that the frequency of the anterior burster neuron is influenced by electrical coupling to other neurons (Hooper and Marder 1987).

Neurons in the inferior olivary nucleus are extensively connected with gap junctions, most of which seem to exist between the spines that populate the dendritic trees of these cells. These have been simulated in GENESIS by using an RAXIAL message passed directly from each dendritic compartment to the other, with the *Ra* field replaced by a number representing the resistance of the gap junction.

A typical specific membrane resistance for a gap junction is on the order of $10^{-4}\ \Omega m^2$, which is much lower than that for a typical cell membrane. For two compartments that are coupled by a junction with a 1 μm diameter, the resistance of the coupling would be about 30 $M\Omega$. In such a case, the following messages might be used to couple two cells with a gap junction.

```
float Rgap = 3e7
addmsg /cell1/dendrite  /cell2/dendrite RAXIAL {Rgap} Vm
addmsg /cell2/dendrite  /cell1/dendrite RAXIAL {Rgap} Vm
```

If the value of *Rgap* is changed, these messages must be deleted and re-sent with the new value.

19.7 Dendrodendritic Synapses

Mitral cells in the olfactory bulb interact with axonless granule cells via dendrodendritic synapses (Rall and Shepherd 1968, Shepherd and Greer 1990). In GENESIS, these synapses may be modeled either by a combination of a **synchan** and a **table** object, or by the **ddsyn** object. Both approaches are illustrated by a demonstration in the *Scripts/examples/ddsyn* directory.

The **ddsyn** object is much like a **synchan**, with the same internal fields and messages. However, it also has an internal table that maps a presynaptic potential to the channel activation. In a typical usage, it gets its activation, not in the form of a δ-function spike from an axonal connection, but from its internal table in response to a voltage sent from the presynaptic dendrite compartment with a V_PRESYN message. This activation is then used to generate a channel conductance governed by two time constants, τ_1 and τ_2, as with the **synchan**.

19.8 Exercises

1. Convert one of the channels in *hhchan.g* into a **tabchannel**. Test your channel and verify that it gives the same results as the original.

2. Yamada, Koch and Adams (1989) give a slightly different model for the AHP current from the one used in the Traub model. In MKS units, they use

$$\alpha(C) = f(C)$$

$$\beta = 2.5(1/sec) = const$$

$$f(C) = 1.25 \times 10^8 C^2,$$

   with $C = [Ca^{2+}]$ in $mmoles/liter = moles/m^3$. Use the **tabchannel** *Z* gate to implement a model of this current. Incorporate it into a simple cell and describe its effects.

3. The term BI_{Ca} in Eq. 19.12 represents the number of moles of Ca^{2+} ions that enter a unit volume each second, due to a current I_{Ca}. Assume that the volume is a thin shell with a surface area A and a thickness d. Show that when SI units are used, $B = 5.2 \times 10^{-6}/Ad$.

4. Modify the script from Chapter 15 so that it uses an NMDA channel. Use the parameters values given with Eq. 19.14, but expressed in SI units. Choose a value of *gmax* that results in occasional action potentials, but mostly subthreshold EPSPs. Also modify the graph for the conductance plot so that it plots both the unblocked conductance of the **synchan** element and the net conductance of the **Mg_block** element. Compare these two plots and the plot of the soma membrane potential. Can you verify that the NMDA channel is behaving as one would expect during the course of an action potential?

5. The original Traub et al. (1991) CA3 pyramidal cell model had a quisqualate-activated conductance and an NMDA conductance that were not implemented in the model used in Chapter 7. From the description given in their paper, implement these conductances and use *Neurokit* to provide them with synaptic input. Verify that the model behaves as expected.