Chapter 20
Speeding Up GENESIS Simulations

ERIK DE SCHUTTER and DAVID BEEMAN

20.1 Introduction

As your GENESIS simulations grow in complexity, either through increased level of detail in multi-compartmental cell models, or through increased size of network models, you will eventually begin to look for ways to increase the speed of your simulations. In the following section, we present some general hints, methods and "tricks" that you can use to make your simulations run faster. For models that contain many compartments, you can obtain large speedups by using a more efficient numerical method for the integration of the compartmental equations. In fact, as we will see in Sec. 20.3.4, the default integration method that is used in GENESIS is much better suited to simple cell models with few compartments. The rest of this chapter describes how you may implement these improved methods, with emphasis on the fast *implicit* methods that are associated with the GENESIS **hsolve** object.

20.2 Some General Hints

Here are some general suggestions for speeding up a GENESIS simulation:

Use multiple clocks A typical neural simulation has a variety of elements whose behavior spans a range of time scales. As we have seen in Chapter 7, action potentials produced by sodium channels have very rapid rise times, whereas some varieties of potassium channels and changes in calcium concentration have time scales that can be on the

order of minutes. Many simulators use a single integration step size for all parts of the simulation and often automatically choose the integration step that is to be used. Although this is a convenience to the user, it gives you less control over factors that affect the speed and accuracy of the simulation. Chapter 13 described the use of *useclock* and *setclock* for assigning different clocking rates to the simulation of different elements. You can greatly speed up a simulation by using slower clocks for simulation elements that can be updated at a slower rate. This is particularly useful in the case of graphical elements. Some loss in resolution in plotted results can greatly increase the speed of the simulation with no cost to the overall accuracy of the results. It is not advisable to run computational elements (**channels**, **compartments**, ...) at different clock speeds, as this will often reduce the simulation accuracy to that of the element with the slowest clock. In case of doubt, you should make sure that your simulation is still giving the same results after making any changes to clocking rates.

Perform runs in "batch" mode Graphical output is very slow, because many plot messages are being exchanged at every simulation step. Most large research simulations perform very little graphical output. The important data are written to a file to be plotted and analyzed later. After you have tested your simulation interactively, modify the scripts to allow it to run without user input or graphical output when you are ready for long runs. The *Piriform* and *MultiCell* simulation scripts give examples of how you may cleanly separate the graphical and non-graphical simulation code. This is also discussed in Sec. 22.7. The *save* and *restore* commands (described in the GENESIS Reference Manual) can be used if you need to split the simulation into a number of runs that begin at the state where the previous one stopped.

Simplify cell models Using the simplest possible cell model that has the desired characteristics can make your network simulations run faster. For example, the somata in the original Wilson and Bower (1989) olfactory cortex model had no Hodgkin-Huxley Na and K channels. Instead of generating true action potentials, a soma simply passed its V_m to a spike generator that fired a spike when V_m crossed a threshold. Often one does some short runs with a fairly detailed model, and then starts making simplifications and approximations until an acceptable simple model is found. The *Piriform* simulation scripts include the capability of using either this *integrate-and-fire* model or the Hodgkin-Huxley model for action potential generation.

Use table lookup Many calculations can be greatly speeded up by looking up numerical values in a table, rather than calculating them. This is why **tabchannel** and **tab2Dchannel** elements are processed much faster than **hh_channel** elements. Section 19.4.4 mentions ways to use the **table** object to replace other objects that normally perform a calculation.

Use compiled functions Any GENESIS script that involves a large number of steps in a loop is likely to be slow, because it is being processed by an interpreter. That is why most GENESIS scripts merely create instances of precompiled objects and set up messages to be passed between them. All of the real computation is done with compiled code that is either part of compiled function definitions or part of the actions defined for an object. If you find that you are using script language functions in a manner that causes them to be invoked repeatedly during the course of a simulation, it may be a good idea to write new functions in C, or write C code to define new actions for an object. These may then be compiled into your own version of GENESIS, as described in the GENESIS Reference Manual section on Customizing GENESIS.

Reduce "swapping" to disk If the progress of your simulation slows down significantly and you begin hearing the sounds of increased disk activity, this is a sign that your simulation has used up the amount of conventional memory that is available and is swapping out parts of your simulation to disk. The GENESIS *showstat -process* command can be used to tell how much memory your simulation is using at any point. If closing down other applications that are running at the same time doesn't give you enough memory, it may be worth it to install more memory in your machine, or to find ways to reduce the memory requirements of your simulation.

Use parallel computation GENESIS has been ported successfully to several parallel computer platforms and to networked workstations. Both networks and single neuron models have been parallelized by researchers using GENESIS. A version of GENESIS, called PGENESIS, is available for use with the Parallel Virtual Machine (PVM), allowing GENESIS to run on the many systems that support the PVM standard. Using parallel computers has several potential advantages. First, they can increase execution speed, but there is a trade-off as increasing the number of computing processors also increases the communication overhead between these processors. Second, because the model is distributed over a lot of processors' memory, huge models can be implemented. The simplest way to use parallel computers is as a "CPU-farm," without any communication between nodes. This is useful in situations where you need to run the same simulation many times, using different sets of parameters, in order to perform parameter space searches to "fine-tune" the model (Bhalla and Bower 1993, De Schutter and Bower 1994c, Vanier and Bower 1996). This does not require any special code and is equivalent to running GENESIS on a lot of workstations.

To get true parallelism, PGENESIS defines a new object, called the **postmaster**, which takes care of all communication between processors (either on a parallel computer or between different workstations). Using the **postmaster** is rather transparent. For example, one can send a message to an element on another processor. For more information about using PGENESIS, see Chapter 21 and the PGENESIS documen-

tation.

Use faster numerical methods We have used the default integration method in the simulations that we have constructed so far. However, you may have noticed that the *Neurokit* implementation of the *Burster* tutorial of Chapter 7 has a `method` dialog box showing "`11`" and that the *Cable* tutorial of Chapter 5 provides a menu for selecting the numerical method to be used. As we mentioned earlier (Sec. 2.4.2), there are tradeoffs between speed, accuracy, and flexibility when choosing a numerical method. The remainder of this chapter describes how you may use much faster numerical methods in your simulations, and how to deal with some of the restrictions that they place on your ability to modify the simulations.

20.3 Numerical Methods Used in GENESIS

In order to explain some of the issues involved in choosing an appropriate numerical integration method, we present here a discussion of the use of numerical methods for solving the equations that arise in neural simulations. You may skip the rest of this section if you are not interested in this level of detail. If you are interested in learning more about this subject, you may wish to read the review by Mascagni (1989).

20.3.1 The Differential Equations Used in GENESIS

Equation 2.1, giving the membrane potential in a generalized neural compartment, and Eq. 4.4 for the state of a Hodgkin-Huxley channel gate, are typical of the differential equations to be solved in a neural simulation. Both of these are members of a set of N coupled first-order ordinary differential equations of the general form

$$\frac{dy_i}{dt} = f(t, y_1, y_2, \ldots, y_N), \text{ with } i = 1, \ldots, N. \tag{20.1}$$

In order to simplify the notation, we drop the subscript i and write the general equation to be solved as

$$\frac{dy}{dt} = f(t). \tag{20.2}$$

However, you should keep in mind that the dependence on the time t usually enters implicitly through the time dependence of the various y's. For example, in Eq. 2.1, the membrane potentials in the adjacent compartments V_m'' and V_m', the conductances G_k, and the injection current I_{inject} all depend on t.

20.3.2 Explicit Methods

Forward Euler Method

The *forward Euler* method is the simplest of the numerical methods available to GENESIS for the solution of Eq. 20.2. For a time increment Δt, we approximate $y(t+\Delta t)$ by

$$y(t+\Delta t) = y(t) + f(t)\Delta t. \tag{20.3}$$

This approximation is equivalent to keeping only the first derivative in a Taylor series expansion,

$$y(t+\Delta t) = y(t) + \frac{dy}{dt}\Delta t + \frac{1}{2}\frac{d^2y}{dt^2}(\Delta t)^2 + \frac{1}{6}\frac{d^3y}{dt^3}(\Delta t)^3 + \ldots \tag{20.4}$$

These terms that involve the higher derivatives can often be very large. We will later see that the forward Euler method suffers from instability problems. Thus, this method is a relatively poor approximation to the solution, and is rarely used in GENESIS simulations.

Adams-Bashforth Methods

The *Adams-Bashforth* methods approximate these missing higher derivatives by making use of past values of $f(t)$ in the approximation for $y(t+\Delta t)$. The general form is

$$y(t+\Delta t) = y(t) + \Delta t(a_0 f(t) + a_1 f(t-\Delta t) + a_2 f(t-2\Delta t) + \ldots + a_n f(t-n\Delta t)), \tag{20.5}$$

where the coefficients a_n may be found by expanding $f(t-n\Delta t)$ in a Taylor's series and comparing Eq. 20.5 with Eq. 20.4. If we evaluate f at n previous times in Eq. 20.5, we say that this is an $(n+1)th$ order Adams-Bashforth method, because it corresponds to keeping terms through the $(n+1)th$ derivative in the Taylor's series expansion. GENESIS lets you choose between second- through fifth-order Adams-Bashforth methods. These methods are computationally very efficient, as they achieve higher accuracy by making use of "free" information that has already been calculated at previous time steps.

A higher-order Adams-Bashforth method is said to have a small *truncation error*, as it includes many terms in the Taylor's series expansion. This means that the error introduced at each step by the use of a finite value of Δt will be small. However, the repeated use of these equations may produce a large *cumulative error* after many integration steps. This is because they depend on extrapolation from past values of f that may have little relevance for the future. These methods tend to give the best results in cases where $f(t)$ varies smoothly, without sharp changes. If $f(t)$ varies rapidly with time, a lower-order method may have a lower cumulative error.

Exponential Euler Method

The *exponential Euler* method (MacGregor 1987) is the default integration method for GENESIS simulations. The efficiency and accuracy of this method depends on the fact that typically encountered equations such as Eqs. 2.1 and 4.4 assume the form

$$\frac{dy}{dt} = A - By. \tag{20.6}$$

Although A and B may depend on y and t in very complicated ways, we will see that the special case where $f(t) = A - By$ results in a considerable simplification of the problem to be solved. For a time step Δt, we can approximate the solution at a time $t + \Delta t$ by

$$y(t + \Delta t) = y(t)D + (A/B)(1 - D), \tag{20.7}$$

where we define

$$D = e^{-B\Delta t}. \tag{20.8}$$

This result follows from the fact that there is an exact solution for the differential equation when A and B are constant. In this case, it may be verified by substitution that we can express the value of y at a time t_2 in terms of its value at an earlier time t_1 using the relationship

$$y(t_2) = y(t_1)e^{-B(t_2 - t_1)} + \frac{A}{B}(1 - e^{-B(t_2 - t_1)}). \tag{20.9}$$

In fact, A and B are usually *not* constants. However, if we assume that they change very little between the time $t_1 = t$ and $t_2 = t + \Delta t$, we may use this result to obtain the approximate solution given above. Although it is difficult to rigorously analyze the error introduced by this approximation, simulation results show that it is highly accurate for most models that contain active channels and only a few compartments. In these cases, it allows much larger integration steps than the methods discussed so far.

20.3.3 Implicit Methods

The methods used in Eqs. 20.3, 20.5 and 20.7 are called *explicit* methods because the new values are given explicitly in terms of functions of the old values.

Backward Euler Method

The *backward Euler* method is an example of an *implicit* method. In this case we use

$$y(t + \Delta t) = y(t) + f(t + \Delta t)\Delta t. \tag{20.10}$$

For implicit methods, the right-hand side of the equation involves a function of the new value of y, which has yet to be determined. Thus, Eq. 20.10 gives an implicit definition of $y(t+\Delta t)$, rather than an explicit expression that can be evaluated. This means that we need some additional method to solve the equations that arise at each step. It would seem that there would be no point in using such an implicit method. From a Taylor's series expansion, you should be able to verify that the truncation error in Eq. 20.10 is the same as that of Eq. 20.3, but is of opposite sign. However, we will see that the use of implicit methods can lead to a much lower cumulative error under certain conditions.

Crank-Nicholson Method

In addition to the backward Euler method, GENESIS allows the use of another implicit method, the *Crank-Nicholson* method. This method is based upon the trapezoidal rule of numerical integration. It uses an average of the forward and backward Euler methods in order to achieve a partial cancellation of errors. This occurs because the neglected second derivative terms are equal and opposite in the two approximations. The approximation is then

$$y(t+\Delta t) = y(t) + (f(t) + f(t+\Delta t))\Delta t/2. \tag{20.11}$$

20.3.4 Instability and Stiffness

Some coupled differential equations present particular problems when they are solved by stepwise numerical integration. For example, the exact solution to the differential equation may have a well-behaved slowly varying solution $y(t) = g(t)$, which we would expect to be able to obtain with a reasonably large step size. However, the difference equation that is used to approximate the differential equation may have another solution of the form $y(t) = g(t) + Cexp(Bt)$, where B has a large magnitude. The initial conditions are such that $C = 0$, so we would expect the difference equation to give the same result. However, as a result of the finite step size, truncation error will cause small errors in the computed solution that have the effect of modifying the initial conditions so that C is not exactly zero. The only way to ensure that this rapidly varying spurious solution remains negligibly small is to use a very small time step in order to minimize the error in its computation. Thus, we are forced to use a time step that is appropriate for a time scale much shorter than that of the actual solution. Examples showing how this type of instability may arise are given by Acton (1970) and by Press, Flannery, Teukolsky and Vetterling (1986).

This sort of behavior leads to the definition of a *stiff* differential equation as one whose solution contains a wide range of characteristic time scales. It turns out that the coupled sets of equations used in compartmental modeling often possess this type of stiffness. Unless a very small time step is used, this can lead to numerical instabilities, resulting in wild

oscillations in the computed solution. For example, the equations describing compartments become stiff when you use compartments with a small size (e.g., in dendritic spines) and/or inject large currents. In such cases, if your time step is not small enough, the simulation will blow up if the current increases beyond a certain threshold (as, for example, during an action potential). In general, explicit methods are much more prone to this type of instability than implicit methods.

As we have seen in Chapter 5, a uniform passive cable composed of identical compartments of length l and diameter d is described by the more specific form of Eq. 2.1,

$$C_M \frac{dV_i}{dt} = \frac{d}{4R_A} \frac{V_{i+1} - 2V_i + V_{i-1}}{l^2} - \frac{V_i}{R_M}. \tag{20.12}$$

(Here, we have taken Eq. 5.28 in the absence of channel currents, and have used Eqs. 5.4–5.6 to express it in terms of the specific membrane capacitance and resistance and the specific axial resistance, C_M, R_M and R_A, respectively.) Mascagni (1989) has analyzed the solution to this equation using the forward Euler, backward Euler, and Crank-Nicholson methods. When the forward Euler method is used, the solution is numerically unstable for $\Delta t > 2R_A C_M l^2/d$. Thus, instability can be a problem for compartments that have a small length l, unless the step size Δt is very small. However, the backward Euler method is stable for any value of Δt. Of course, numerical accuracy will suffer if Δt is too large, but the solution will not display the catastrophic instability produced by the forward Euler method.

As the Crank-Nicholson method involves an average of the forward and backward methods, one might expect it to be less stable than the backward Euler method. Although the analysis shows that it is also stable for any value of Δt, simulation results show that it comes closer to the point of instability. Numerical solutions for stiff equations obtained with the Crank-Nicholson method often show spurious damped oscillations or over/undershoot when large values of Δt are used. Nevertheless, its greater accuracy per step (smaller truncation error) coupled with its moderately good stability make it a good choice for high precision calculations if the time step is sufficiently small.

Although it is an explicit method, the default exponential Euler method does not suffer the dramatic onset of instability shown by the forward Euler method (see Exercise 2). Interestingly, as you may verify in Exercise 3, it is nevertheless not as accurate as the forward Euler method with the same step size when there are many small compartments. Although it gives fairly accurate results for integration of the channel conductance equations, and is a good choice for simple cell models with few compartments, it will require a much smaller time step than the backward Euler or Crank-Nicholson methods if the equation to be solved is stiff. The lesson to be learned from this analysis is that you should choose a numerical method that is appropriate for the simulation to be performed, and should *always* experiment with different values of the integration step size before trusting your results.

20.3.5 Implementation of the Implicit Methods

So far we have neglected the question of how one implements an implicit method. The right-hand sides of Eqs. 20.10 and 20.11 involve functions that depend on the unknown quantity on the left-hand side. In general, we would have to use iterative methods, such as *predictor-corrector* or Newton's methods (Acton 1970) in order to find the solution.

Equation 20.12 and its more general form Eq. 2.1 each have the convenient property that the right-hand side of the equation involves the unknown membrane potential in the compartment and that in the immediately adjacent compartments. Thus, the matrix that represents the coupled sets of equations to be solved is a tridiagonal matrix — a sparse matrix with non-zero elements along the diagonal and just above and below the diagonal. This greatly simplifies the problem of solving the matrix equation. A method due to Hines (1984) provides a very efficient way to solve these equations when the coupled equations describe a branching tree-like structure (such as the dendrites of a neuron) without closed loops. In GENESIS, the Hines method is embodied in a special object **hsolve** which is used in conjunction with the backward Euler and Crank-Nicholson methods.

20.4 The *setmethod* Command

The *setmethod* command is used to specify the numerical integration method to be used in a GENESIS simulation. It takes an optional argument to specify the path to the elements to which the method will apply, followed by an integer identifying the integration method to use. If the path is omitted, the specified method will be applied to all currently existing elements in the simulation. (Any subsequently created elements will use the default method, however.) For example,

```
setmethod 2
```

will cause method 2 (the second order Adams-Bashforth method) to be used for all elements in the simulation. If this command is followed by

```
setmethod /cell 0
```

method 0 (exponential Euler) will be used for all elements in the */cell* element tree, and the rest of the simulation elements will be treated as before. Just as the *useclock* and *setclock* commands let us pick an integration step that is appropriate to particular simulation elements, we can use *setmethod* in this way to choose the most appropriate integration method. Table 20.1 gives the integers that correspond to the currently implemented GENESIS integration methods. The two implicit methods, methods 10 and 11, must be used in conjunction with an **hsolve** object, as described in the next section.

Method number	*Description*
-1	Forward Euler
0	Exponential Euler (default)
2	Adams-Bashforth 2nd-order
3	Adams-Bashforth 3rd-order
4	Adams-Bashforth 4th-order
5	Adams-Bashforth 5th-order
10	Backward Euler
11	Crank-Nicholson

Table 20.1 Presently available GENESIS numerical integration methods.

20.5 Using the hsolve Object

The **hsolve** method and its associated GENESIS object implement the Hines method of solving the equations for branched neuronal structures. By using **hsolve** with the backward Euler or Crank-Nicholson integration method, you will be able to use much larger integration steps in simulations of multi-compartmental models. In addition, the **hsolve** object provides a highly optimized code besides the use of the Hines implicit integration schemes. Even with the same size integration step, the **hsolve** method results in a speedup of at least 50% over the usual exponential Euler method.

However, this greater speed comes at a price. The matrix methods that are used to solve the coupled differential equations are incompatible with the object-oriented nature of GENESIS, in which elements communicate only by the exchange of messages. In order to carry out the solution, we create an **hsolve** element that takes over the calculations which are performed by a cell or other tree of linked compartments. The entire cell then acts as a single object, since the individual compartments within the cell are no longer responsible for their own computations. However, this is done in such a way as to preserve the *illusion* of object orientedness. Within certain limits, you may continue to set element fields and create and delete messages between objects. The action of the **hsolve** object on the simulation elements has been compared to one of those 1950's science fiction films in which alien beings have taken over your friends. Outwardly, they seem normal. Only occasional departures from normal behavior reveal that something very different is in control (Bhalla 1990).

In order to make the **hsolve** method as efficient as possible, only a few of the GENESIS objects can be taken over by **hsolve**. These are the ones that typically account for the greatest computational load in a simulation, and are most likely to lead to numerically stiff equations. At present, the objects that are supported by **hsolve** are the **compartment**, **tabchannel**, **tab2Dchannel**, **tabcurrent**, **synchan**, **spikegen**, **Ca_concen**, **nernst**, **ghk**, **difshell**, **fixbuffer** and **difbuffer** objects. Other types of elements continue to be treated as

they were before the **hsolve** element took over the cell.

The Hines algorithm is only applicable to branched structures that do not form closed loops of interactions between their elements. This prevents it from being applied to gap junctions (Sec. 19.6). This also means that the method should be applied to individual cells in a network, but not to the network as a whole, unless the network has no loops of interactions.

Because of these various restrictions and the loss of flexibility in setting fields and changing messages, we suggest that you first build and test your model using the default exponential Euler method. Once it is working properly and is unlikely to be changed, you may add the statements necessary to implement the **hsolve** method.

Future releases of GENESIS will provide a *readsolve* command. This command reads the same files as the *readcell* command and creates the corresponding **hsolve** element, without creating the other elements in the cell tree. As a consequence it uses less memory than the standard way of invoking **hsolve**, but the special *findsolvefield* function must be used to output simulation values.

20.5.1 Modes of Operation

The **hsolve** object has various modes of operation, referred to as *comptmodes*, *chanmodes*, *calcmodes* and *storemodes*. The *comptmode* field is a flag that determines the way compartment computations will be performed. For best performance, this field should be set to 1 (the default). This, however, increases memory use significantly. If you run out of memory, try setting *comptmode* to zero *before* setting up the **hsolve** element.

The *chanmode* field controls channel computations. At present, there are four *chanmodes* available in GENESIS, labeled with the integers 0 through 4. After an element is created from the **hsolve** object, the mode to be used is selected by setting the *chanmode* field in the element. In general, the higher mode numbers are faster, but place more restrictions on your ability to modify the simulation. All of the modes have the same accuracy, and thus allow you to use comparable time steps. In general, these will be an order of magnitude larger than the maximum step size allowable for the default exponential Euler method.

chanmode 0

This is the default mode of operation. Although it uses the least amount of memory, it is also the slowest. To implement the Hines method, it takes over the actions of compartments only, computing all other object types as before. As a consequence, all computed fields of the original elements are updated, and all user-setable fields may be set, just as before the element was taken over by the **hsolve** element. This means that you can add and delete outgoing messages to compartments or other elements whenever you like and easily change parameters during the course of the simulation.

One significant limitation is that you cannot add or delete AXIAL, RAXIAL or CHANNEL messages once the **hsolve** element has been created. If you can live with these limitations, *chanmode* 0 is the easiest one to use. It is also the most compatible mode of operation, and is guaranteed to work with any future new object type.

chanmode 1

This *chanmode* uses optimizations for the computation of **tabchannel** current to the **hsolve** computations. It has the same limitations as *chanmode* 0, but achieves a large increase in computation at the cost of increased memory use. This is the preferred *chanmode* if you employ **tabchannels** in your model and do not want to use *chanmodes* 2 through 4.

chanmode 2

The *chanmodes* 2 through 4 achieve a significant speedup at the expense of greatly increased memory usage. Under these modes, the original elements describing channels, concentrations, etc. are not used at all. Instead, all simulation parameters are stored in a huge array, called the *chip-array*. This reorganization optimizes the speed of the use of CPU cache memory. If your computer has limited memory and is forced to perform increased swapping to disk after changing to *chanmode* 2, this may negate the speed advantage. These *chanmodes* also assume that *comptmode* is set to 1 and will change the *comptmode* field if necessary. As a consequence of the use of the chip-array, the element tree of your cell (or other element tree to be taken over by hsolve) must not contain any non-hsolvable elements when these *chanmodes* are used. However, within the limitations described below, messages may be exchanged with non-hsolvable elements that lie outside the hsolved element tree.

chanmodes 2 through 4 require integer exponents for **tabchannel** and **tab2Dchannel** gate variables. In addition, the maximum exponent allowed is 6. Incoming and outgoing messages from the disabled elements are supported, providing that they are established *before* setting up the **hsolve** element. If you try to add new messages afterwards, they will go to the old disabled elements only, and will be ignored. Likewise, using *deletemsg* to remove an existing message after the set up will have no effect.

Under these modes, you can no longer assume that all the fields of the elements that are taken over by hsolve will be updated. This will make it harder to display these fields graphically, or to output their values to a file. However, the *Vm* field of all compartments will be automatically updated when using *chanmode* 2. Any new outgoing messages to non-hsolved elements are from the original objects. Thus, whether the message works depends on whether their fields get updated. You can always establish a new outgoing message, but it may not give current information. You can prevent this problem from occurring by using the **hsolve** element itself as the source of outgoing messages, using the *findsolvefield*

function.

Another consequence of this "takeover" is that *setfield* commands will be ignored until you perform a *reset*. The GENESIS Reference Manual describes how this restriction may be overcome with the **hsolve** object's HPUT and HRESTORE actions or by using the *findsolvefield* function.

chanmode 3

This mode is very much like *chanmode* 2, except that elements are automatically updated if they had outgoing messages to non-hsolved objects prior to the setup. This means that you can plot or otherwise access fields other than *Vm*, but the *Vm* is not automatically updated as in *chanmode* 2. On the other hand, if any new messages are added after setup, the old values are sent. Note also that several fields (e.g., *Gk*, *Ik*, *Ek* and *Im*) are not available for output in *chanmode* 3, so you have to use *chanmode* 4 if you want to output these fields

Mode 3 offers a little more efficiency because it allows you to specify the clock that is used to update fields in the original elements. This means that if you want to send the value of a field to a graph with a PLOT message, the element fields can be updated at a slower rate than the one used by the simulation clock. Of course, the elements that have been taken over by **hsolve** will exchange messages at the faster rate specified by the simulation clock. The update clock is specified by setting the *outclock* field of the **hsolve** element. For example,

```
setclock 1  10*{dt}
setfield /cell/solve outclock 1
useclock /graphs/conductance_plot 1
```

Note that if you want to output *Vm* only, but from multiple compartments, or to create and delete plots of *Vm*, you should use *chanmode* 2.

chanmode 4

This mode is identical to *chanmode* 3, but allows you to output additional fields at the cost of a reduction in computation speed. In *chanmodes* 2 and 3, the hsolve element "forgets" values like *Ek* and does not compute values like *Ik* and *Im* because they are not needed during the present or next integration step. As a consequence, the *Ek* and *Ik* fields of a disabled channel element cannot be updated. If you had specified an outgoing SAVE or PLOT message for one of these fields prior to setup, the output would always be zero. In *chanmode* 4 the values of *Ek*, *Ik* and *Im* are computed and stored in a *givals-array* so that the proper values will be available for SAVE or PLOT messages, when used with *findsolvefield*. Additionally, in *chanmode* 4 a field *leak* is available for each compartment, which gives the leak current flowing through the compartment *Rm*.

To give an idea of the effect of *comptmodes*, *chanmodes* and *calcmodes* on computation speed, we provide a comparison of computation times in seconds for 1000 steps of the standard 1600 compartment Purkinje cell model of De Schutter and Bower (1994a) on a HyperSparc platform in Table 20.2.

		calcmode 0	*calcmode* 1
comptmode 0	chanmode 0	n.a.	137.9
comptmode 0	chanmode 1	n.a.	82.8
comptmode 1	chanmode 0	n.a.	139.1
comptmode 1	chanmode 1	n.a.	76.4
comptmode 1	chanmode 2	23.0	24.0
comptmode 1	chanmode 3	21.4	21.8
comptmode 1	chanmode 4	26.9	27.3

Table 20.2 Comparison of computation times (in seconds) for various combinations of *chanmodes*, *comptmodes* and *calcmodes*. 1000 steps of simulation were performed for a 1600 compartment Purkinje cell.

storemode

With chanmode 4, the *storemode* field allows the output of total currents and conductances from a compartmental model. This technique was used extensively by Jaeger, De Schutter and Bower (1997) to study the role of voltage-gated currents in the control of Purkinje cell spiking. For each type of voltage-gated channel in the model, the total currents or conductances are the sum of the corresponding *Ik* or *Gk* fields for all compartments where the channel is present. This assumes that these channels have the same name in each compartment. Total currents are computed if *storemode* is set to 1, and total conductances if it is set to 2. These are stored in an array called *itotal*. When the **hsolve** element is set up, a message will be output giving a list of channel names and corresponding *itotal* indices if the *silent* command has previously been given with a negative argument. Alternatively, you may start GENESIS by giving the *genesis* command with the *-silent* option and a negative value.

calcmode

The *calcmode* field affects operations for *chanmodes* 2 through 4 only. It is initialized to 1 (LIN_INTERP), which is the recommended mode. In the LIN_INTERP mode, all values obtained from lookup tables will be interpolated, ensuring a higher accuracy. The *calcmode* exists mainly to provide backward compatibility, as older versions of the **hsolve** object in GENESIS versions 2.0 and earlier did not use interpolation, which is equivalent to setting *calcmode* to 0 (NO_INTERP).

20.5.2 Rules for Table Dimensions

The *chanmodes* 2 through 4 of the **hsolve** object also reorganize the lookup tables present in **tabchannel**, **tab2Dchannel** and **tabcurrent** objects to achieve an extra increase in computation speed. However, this works only if *all* lookup tables are of the same dimensions. Specifically, the *xdivs* field in all tables should be identical and if **tab2Dchannels** are used, the *ydivs* should equal the *xdivs*. Moreover, all voltage-indexed tables should have identical *xmin* and *xmax* values. The same is true for all concentration-indexed tables. The **hsolve** element will alert you during the setup phase if these rules are not obeyed.

In practice, when **tab2Dchannels** are used in the simulation, all tables will have a few hundred *xdivs* instead of a few thousand because otherwise you will rapidly run out of memory. Because of the small size of the tables it is very important to use the LIN_INTERP *calcmode*. Obviously, you should not use the TABFILL action on such small tables, but instead compute every entry of the table.

20.6 Setting up hsolve

The preferred location of the **hsolve** element is as the root element of the cell. Although the actual location of the **hsolve** element within the tree of elements does not matter in GENESIS 2.1, it will do so in future GENESIS versions. If you want to use *chanmodes* 2 through 4, you should also organize the tree of elements describing your model properly:

1. The **hsolve** element should be the root of the cell element tree.
2. The next level of the tree should contain compartments only.
3. All channel, concentration, etc. objects should be children or grandchildren of the compartment to which they are attached.
4. To make the *findsolvefield* command behave properly, all grandchildren of a particular compartment should have different names.
5. The cell element tree should not contain any elements that are not computed by the **hsolve** object except for *neutral* elements.

The **hsolve** method may be applied by following these steps:

1. Create the **hsolve** element and name it as you would name the cell using a **neutral** object, e.g., */cell*. If you plan to use the readcell command to create */cell* you should use the *-hsolve* option to achieve the same effect:

```
readcell {filename} /cell -hsolve
```

2. If you do not use readcell, create the cell or other structure that is to be treated with the **hsolve** element using it as the root element. For example,

```
create hsolve /cell
create compartment /cell/soma
create tabchannel /cell/soma/Na_channel
...
```

   would be used to have an **hsolve** element called */cell* compute the */cell* element tree.

3. If your **hsolve** element is not the root element of the cell, you should set the path to the elements that will be treated with **hsolve**. This may be best done using a wildcard specification such as

```
setfield /cell path /cell/##[][TYPE=compartment]
```

   to set the *path* field of *solve* to an expression describing all **compartment** elements in the */cell* hierarchy. Any subelements of the compartments that are treatable by **hsolve** will be taken over by **hsolve** as well. Thus, it is not necessary to list any **tabchannel** or **Ca_concen** elements that are associated with the compartments.

4. If you use a *chanmode* other than the default *chanmode* 0, set the *chanmode* field of the **hsolve** element to the mode number. For example,

```
setfield /cell/solve chanmode 3
```

5. Then, tell the **hsolve** element to create all the solution arrays and tables by calling its SETUP action:

```
call /cell SETUP
```

6. Set the integration method to be used with the *setmethod* command. For example,

```
setmethod 11
```

   will cause the Crank-Nicholson method to be used for all elements that are taken over by **hsolve**. The default is the backward Euler method. Those elements that are not treatable by **hsolve** will continue to be treated by the method which was previously in use, usually the default exponential Euler method.

7. Finally, it is essential to call *reset* after setting up an hsolver, so that the process list gets updated.

```
reset
```

Now you are ready to try running the simulation, starting with the original step size and decreasing it until errors begin to creep in. Often, the **hsolve** method will allow you to use time steps that are an order of magnitude larger than those required for the exponential Euler method.

Sometimes you may need to modify certain element fields or messages during the course of the simulation. If the *chanmode* you are using does not allow this, the easiest solution is to delete the **hsolve** element, make the changes, and then recreate the element by repeating the steps given above.

20.6.1 The *findsolvefield* Function

The *findsolvefield* function is used together with *chanmodes* 2 through 4 to access the **hsolve** arrays directly to output field values instead of having to use the old disabled elements. For example, to output the *Gk* field of *Na_channel* from the **hsolve** element */cell*:

```
call /cell SETUP
...
create disk_out /output
addmsg /cell /output OUTPUT {findsolvefield /cell /cell/soma/Na_channel Gk}
...
reset
```

The *findsolvefield* function returns a string containing the **hsolve** field corresponding to the disabled element field. The syntax of *findsolvefield* is:

```
findsolvefield hsolve element_path field
```

The *findsolvefield* function is always safe to use for output. Of course, whether you can output a particular field will depend on the *chanmode* used. The *findsolvefield* function will alert you if the field is not available. In general, it is not advisable to use the *findsolvefield* function for *setfield* commands, except in this particular example:

```
// inject 5 nA of current
setfield /cell {findsolvefield /cell /cell/soma inject} 0.5e-9
```

It is generally dangerous to use *findsolvefield* in a *setfield* command because the **hsolve** object modifies many field values before storing them in its arrays.

20.6.2 The DUPLICATE Action

If you are creating a network of cells, each cell must have its own **hsolve** element. If the cells are identical, you can save a great deal of memory by making copies of the original cell (and attached **hsolve** element). Then, use the DUPLICATE action for each of the copied **hsolve** elements. This conserves memory by allowing many of the tables in the original **hsolve** element to be used for the copies. For example, if */cell2* is to be created from */cell*, use

```
// this copies the element tree including the hsolve root element
copy /cell /cell2
call /cell DUPLICATE /cell2
```

As we described in Chapter 18, large networks may be created with a *for* loop, or with one of the specialized functions such as *createmap*.

20.7 Experiments with the hsolve Object

This experiment and those in the following exercises reveal some of the advantages and restrictions of the various *chanmodes*. There is a bewildering variety of combinations of *chanmodes*, other **hsolve** field settings, and actions that you may call. Performing these experiments will not only help you to be sure that you understand their documentation, but it will also reveal any changes that may have occurred in later versions of GENESIS. The **hsolve** object is under continuous development, so you should consult the **hsolve** documentation that accompanies your GENESIS distribution if you intend to use any of the advanced features of **hsolve**. It is possible that some of the restrictions described above will have been removed.

Begin by creating a 20-compartment passive cable. Provide current injection to the first compartment by setting its *inject* field and generate plots of the membrane potential in the first and last compartments. The script for *tutorial4.g* from Chapter 15 would be a good starting point, as it has the graphics and control widgets you will need, as well as a function *makecompartment* for creating the compartments. The parameters used in this simulation for the single dendrite compartment will be suitable. The easiest way to create the cable of linked compartments is to borrow a function from the *Cable* tutorial, which we used in Chapter 5. The script *addcable.g* defines a function *make_cable* that uses a *for* loop to construct a cable with a specified number of compartments. Use this function to construct the cable in place of the two compartment neuron used in *tutorial4.g*. You will also find it useful to provide a dialog box for changing the integration step size.

Test your simulation using the default exponential Euler method, without creating an **hsolve** element. If you have any doubts that it is behaving properly, compare it with the

results of the *Cable* simulation, using the same parameters. Then set up an **hsolve** element and run the simulation using either the backward Euler or Crank-Nicholson method and *chanmode* 0. How much larger a time step can you use for the same accuracy?

After verifying that the simulation also works under the other *chanmodes*, see what happens if you interactively add new PLOT messages under the various modes. You may use the INJECT message to provide injection pulses to a compartment by using the function *inject_compt* from the *inputs.g* script used in the *Cable* simulation. If you prefer, you may provide a constant injection by creating a **neutral** element, setting its *x* field to the desired injection current, and sending it to the compartment with an INJECT message. Do the various *chanmodes* behave as described in Sec. 20.5.1?

20.8 Exercises

1. Add a soma with voltage dependent Na and K channels to the cable that you created in the experiment described above. As you will have to use **tabchannels** instead of **hh_channels** with **hsolve**, create your channels from functions defined in the *neurokit/prototypes* script *mitchan.g*, rather than *hhchan.g*. Describe the results under the exponential Euler and the Crank-Nicholson methods for various step sizes, and present some evidence that your simulation is working as expected.

2. Run the *Cable* simulation from Chapter 5, using a cable of 20 identical compartments, each having a diameter of 2 μm and a length of 100 μm. Select the forward Euler method from the numerical methods menu and vary the step size `dt` in order to determine the point at which numerical instabilities begin. Take some care to locate this point fairly precisely. How does this value compare with the value of Δt predicted in Sec. 20.3.4? (If you prefer, you may use the simulation that you created in this tutorial instead. Rather than setting up the **hsolve** element, use *setmethod* to set the method to "-1".)

3. Use the backward Euler method with the cable described in the previous exercise. Experiment with increasing values of Δt, until you start noticing slight changes in the plotted results. Determine the largest value of Δt that should be used for accurate results. Then try the forward Euler method, exponential Euler, and the second- through fourth-order Adams-Bashforth methods. For each one, list the largest value of Δt that produces a plot indistinguishable from that produced by the backward Euler method. What do you conclude about the relative accuracies of these methods when applied to this problem?