REVIEW Communicated by William Lytton

Expanding NEURON's Repertoire of Mechanisms with NMODL

M. L. Hines
Department of Computer Science, Yale University, New Haven, CT 06520, U.S.A.

N. T. Carnevale
Department of Psychology, Yale University, New Haven, CT 06520, U.S.A.

Neuronal function involves the interaction of electrical and chemical signals that are distributed in time and space. The mechanisms that generate these signals and regulate their interactions are marked by a rich diversity of properties that precludes a "one size fits all" approach to modeling. This article presents a summary of how the model description language NMODL enables the neuronal simulation environment NEURON to accommodate these differences.

1 Introduction

Recently we described the core concepts and strategies that are responsible for much of the utility of NEURON as a tool for empirically based neuronal modeling (Hines & Carnevale, 1997). That article focused on the strategy used in NEURON to deal with the problem of mapping a spatially distributed system into a discretized (compartmental) representation in a manner that ensures conceptual control while maintaining numeric accuracy and computational efficiency. Now we shift our attention to another important feature of NEURON: its special facility for expanding and customizing its library of biophysical mechanisms.

The need for this facility stems from the fact that experimentalists are applying an ever-growing armamentarium of techniques to dissect neuronal operation at the cellular level. There is a steady increase in the number of phenomena that are known to participate in electrical and chemical signaling and are characterized well enough to support empirically based simulations. Since the mechanisms that underlie these phenomena differ across neuronal cell class, developmental stage, and species (e.g., chapter 7 in Johnston & Wu, 1995; also see McCormick, 1998), a simulator that is useful in research must provide a flexible and powerful means for incorporating new biophysical mechanisms in models. It must also help the user remain focused on the model instead of programming. Such a means is provided to the NEURON simulation environment by NMODL, a high-level language that was originally implemented for NEURON by Michael Hines, which

he and Upinder Bhalla later extended to generate code suitable for linking with GENESIS (Wilson & Bower, 1989).

The aim of this article is to provide a high-level summary of those aspects of NMODL that we regard as unique and most pertinent to the creation of representations of biophysical mechanisms. These include features that enable the user to construct mechanisms using differential equations or kinetic schemes, specify continuous (density) or delta function (point process) spatial distributions, and implement models of synaptic connectivity, while preserving mass balance for each ionic species. A detailed explanation of the unusual structure and features of NMODL requires numerous, specific illustrations. Therefore we have posted an expanded version of this document at http://www.neuron.yale.edu/nmodl.html. The expanded version is organized around a sequence of examples of increasing complexity and sophistication that introduce important topics in the context of problems of scientific interest, including models of the following mechanisms:

- A passive "leak" current and a localized transmembrane shunt (density mechanisms versus point processes)
- An electrode stimulus (discontinuous parameter changes with variable time-step methods)
- Voltage-gated channels (differential equations versus kinetic schemes)
- Ion accumulation in a restricted space (e.g., extracellular K^+)
- Ion buffering, diffusion, and active transport (e.g., Ca^{2+})
- Use-dependent synaptic plasticity
- Multiple streams of synaptic input and output (network fan-in and fan-out)

This article makes extensive use of specialized concepts and terminology that pertain to NEURON itself. For definitive treatment of these, see Hines and Carnevale (1997) and NEURON's on-line help files, which are available through links at http://www.neuron.yale.edu.

2 Describing Mechanisms with NMODL

A brief overview of how NMODL is used will clarify its underlying rationale. The first step is to write a text file (a "`mod` file") that describes a mechanism as a set of nonlinear algebraic equations, differential equations, or kinetic reaction schemes. The description employs a syntax that closely resembles familiar mathematical and chemical notation. This text is passed to a translator that converts each statement into many statements in C, auto-

matically generating code that handles details such as mass balance for each ionic species and producing code suitable for each of NEURON's integration methods. The output of the translator is then compiled for computational efficiency. This achieves tremendous conceptual leverage and savings of effort not only because the high-level mechanism specification is much easier to understand and far more compact than the equivalent C code, but also because it spares the user from having to bother with low-level programming issues, like how to "interface" the code with other mechanisms and with NEURON itself.

NMODL is a descendant of the MOdel Description Language (MODL; Kohn, Hines, Kootsey, & Feezor, 1994), which was developed at Duke University by the National Biomedical Simulation Resource project for the purpose of building models that would be exercised by the Simulation Control Program (SCoP; Kootsey, Kohn, Feezor, Mitchell, & Fletcher, 1986). NMODL has the same basic syntax and style of organizing model code into named blocks as MODL. Variable declaration blocks, such as `PARAMETER`, `STATE`, and `ASSIGNED`, specify names and attributes of variables that are used in the model. Other blocks are directly involved in setting initial conditions or generating solutions at each time step (the equation definition blocks, e.g., `INITIAL`, `BREAKPOINT`, `DERIVATIVE`, `KINETIC`, `FUNCTION`, `PROCEDURE`). There is also a provision for inserting C statements into a mod file to accomplish implementation-specific goals. NMODL recognizes all the keywords of MODL, but we will limit this discussion to those that are relevant to NEURON simulations, with an emphasis on changes and extensions that were necessary to endow NMODL with NEURON-specific features.

2.1 The NEURON Block. The principal extension that differentiates NMODL from its MODL origins is that there are separate instances of mechanism data, with different values of states and parameters, in each segment (compartment) of a model cell. The `NEURON` block was introduced to make this possible by defining what the model of the mechanism looks like from the "outside" when there are many instances of the model sprinkled at different locations on the cell. The specifications entered in this block are independent of any particular simulator, but the detailed "interface code" requirements of a particular simulator determine whether the output C file is suitable for NEURON (NMODL) or GENESIS (GMODL).

For example, consider the accumulation of potassium in the extracellular space adjacent to squid axon (see Figure 1). Satellite cells and other extracellular structures act as a diffusion barrier that prevents free communication between the bath and this "Frankenhaeuser-Hodgkin space" (F-H space; Frankenhaeuser & Hodgkin, 1956).

When there is a large efflux of K^+ ions from the axon, as during the repolarizing phase of an action potential or in response to injected depolarizing current, K^+ builds up in the F-H space. This elevation of $[K^+]_o$ shifts E_K in

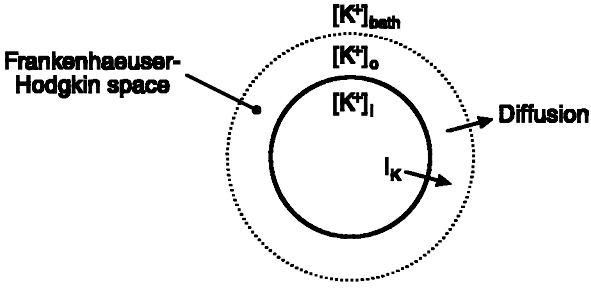

Figure 1: Extracellular potasasium may accumulate adjacent to the cell membrance due to restricted diffusion from the Frankenhaeuser-Hodgkin space.

a depolarized direction, which has two important consequences. First, it reduces the driving force for K^+ efflux and causes a decline of the outward I_K. Second, when the action potential terminates or the injected depolarizing current is stopped, the persistent elevation of E_K causes a slowly decaying depolarization or inward current. This depolarizing shift dissipates gradually as $[K^+]_o$ equilibrates with $[K^+]_{bath}$.

```
NEURON {
  SUFFIX kext
  USEION k READ ik WRITE ko
  GLOBAL kbath
  RANGE fhspace, txfer
}
```

This is the NEURON block for the `kext` mechanism, which emulates the effects of the F-H space. (Readers who desire more detailed information, for example, how the ODE for K^+ accumulation is specified, are referred to http://www.neuron.yale.edu/nmodl.html for the expanded version of this article, which contains complete code and descriptions for this and other mechanisms.)

The `SUFFIX` keyword has two consequences. First, it identifies this as a density mechanism, which can be incorporated into a NEURON cable section by an `insert` statement (see Hines & Carnevale, 1997). Second, it tells the NEURON interpreter that the names for variables and parameters that belong to this mechanism will include the suffix `_kext`, so there will be no conflict with similar names in other mechanisms.

2.1.1 Mechanisms That Interact with Ionic Concentrations. Many biophysical mechanisms involve processes that have direct interactions with ionic concentrations (e.g., diffusion, buffers, pumps). Furthermore, any compart-

ment in a model may contain several such mechanisms. Therefore, total ionic currents and concentrations must be computed consistently. This is facilitated by the `USEION` statement, which sets up the necessary bookkeeping by automatically creating a separate mechanism that keeps track of four essential variables: the total outward current carried by an ion, the internal and external concentrations of the ion next to the membrane, and its equilibrium potential. In this case the name of the ion is "`k`" and the automatically created mechanism is called "`k_ion`" in the hoc interpreter. The `k_ion` mechanism has variables `ik`, `ki`, `ko`, and `ek`, which represent I_K, $[K^+]_i$, $[K^+]_o$, and E_K, respectively. These do not have suffixes; furthermore, they are `RANGE` variables so they can have different values in every segment of each section in which they exist (see the discussion of sections, segments, and `RANGE` variables in Hines & Carnevale, 1997). In other words, the K^+ current through Hodgkin-Huxley potassium channels near one end of the section `cable` would be `cable.ik_hh(0.1)`, but the total K^+ current generated by all sources, including other ionic conductances and pumps, would be `cable.ik(0.1)` (the 0.1 signifies a location that is one-tenth of the distance along the length of the section away from its origin).

Since mechanisms can generate transmembrane fluxes that are attributed to specific ionic species by the `USEION` *x* `WRITE` *ix* syntax, modeling restricted diffusion is straightforward. A mechanism needs a separate `USEION` statement for each of the ions that it affects or is affected by. For example, the Hodgkin-Huxley mechanism `hh` has separate `USEION` statements for `na` and `k`, and the `USEION` statement for potassium includes `READ ek` because the potential gradient that drives `ik_hh` depends on the equilibrium potential for K^+. Since the resulting ionic flux may affect local $[K^+]$, the `USEION` statement for `hh` also includes `WRITE ik` so that NEURON can keep track of the total outward current that is carried by potassium, $[K^+]_i$ and $[K^+]_o$, and E_K.

The `kext` mechanism computes $[K^+]_o$ from the outward potassium current, so it `READ`s `ik` and `WRITE`s `ko`. When a mechanism `WRITE`s a particular ionic concentration, this means that it sets the value for that concentration at all locations in every section into which it has been inserted. This has an important consequence: in any given section, no ionic concentration should be "written" by more than one mechanism.

The bath is assumed to be a large, well-stirred compartment that envelops the entire "experimental preparation." Therefore `kbath` is a `GLOBAL` variable so that all sections that contain the `kext` mechanism will have the same numeric value for $[K^+]_{bath}$. Since this would be one of the controlled variables in an experiment, the value of `kbath` is specified by the user and will remain constant during the simulation. The thickness of the F-H space is `fhspace`, the time constant for equilibration with the bath is `txfer`, and both are `RANGE` variables so they can vary along the length of each section.

2.2 General Comments About Kinetic Schemes. Kinetic schemes provide a high-level framework that is perfectly suited for compact and intuitively clear specification of models that involve discrete states in which "material" is conserved. The basic notion in such mechanisms is that flow out of one state equals flow into another. Almost all models of membrane channels, chemical reactions, macroscopic Markov processes, and ionic diffusion are elegantly expressed through kinetic schemes.

The unknowns in a kinetic scheme, which are usually concentrations of individual reactants, are declared in the `STATE` block. The user expresses the kinetic scheme with a notation that is similar to a list of simultaneous chemical reactions. The NMODL translator converts the kinetic scheme into a family of ODEs whose unknowns are the `STATE`s. Hence the simple

```
STATE { mc     m }
KINETIC scheme1 {
  ~ mc <-> m (a(v), b(v))
}
```

is equivalent to

```
DERIVATIVE scheme1 {
  mc' = -a(v)*mc + b(v)*m
  m'  =  a(v)*mc - b(v)*m
}
```

The first character of a reaction statement is the tilde, ~, which is used to distinguish this kind of statement from other sequences of tokens that could be interpreted as an expression. The expression to the left of the three-character reaction indicator, `<->`, specifies the reactants, and the expression immediately to the right specifies the products. The two expressions in parentheses specify the forward and reverse reaction rates (here, the rate functions `a(v)` and `b(v)`). After each reaction, the variables `f_flux` and `b_flux` are assigned the values of the forward and reverse fluxes, respectively. These can be used in assignment statements such as

```
~ cai + pump <-> capump (k1,k2)
~ capump <-> pump + cao (k3,k4)
ica = (f_flux - b_flux)*2*Faraday/area
```

In this case, the forward flux is `k3*capump`, the reverse flux is `k4*pump*cao`, and the positive-outward current convention is consistent with the sign of the expression for `ica` (in the second reaction, forward flux means positive ions move from the inside to the outside).

More complicated reaction sequences, such as the wholly imaginary

```
KINETIC scheme2 {
  ~ 2A + B <-> C (k1,k2)
  ~  C + D <-> A + 2B (k3,k4)
}
```

begin to show the clarity of expression and suggest the comparative ease of modification of the kinetic representation over the equivalent but stoichiometrically confusing

```
DERIVATIVE scheme2 {
  A' = -2*k1*A^2*B + 2*k2*C   + k3*C*D   - k4*A*B^2
  B' =   -k1*A^2*B   + k2*C + 2*k3*C*D - 2*k4*A*B^2
  C' =    k1*A^2*B   - k2*C   - k3*C*D   + k4*A*B^2
  D' =                        - k3*C*D   + k4*A*B^2
}
```

Clearly a statement such as

```
~ calmodulin + 3Ca <-> active (k1, k2)
```

would be easier to modify (e.g., so it requires combination with four calcium ions) than the relevant term in the three differential equations for the STATEs that this reaction affects. The kinetic representation is easy to debug because it closely resembles familiar notations and is much closer to the conceptualization of what is happening than the differential equations would be.

Another benefit of kinetic schemes is the simple polynomial nature of the flux terms, which allows the translator to perform easily a great deal of preprocessing that makes implicit numerical integration more efficient. Specifically, the nonzero elements $\partial y_i'/\partial y_j$ (partial derivatives of dy_i/dt with respect to y_j) of the sparse matrix are calculated analytically in NMODL and collected into a C function that is called by solvers to calculate the Jacobian. Furthermore, the form of the reaction statements determines if the scheme is linear, obviating an iterative computation of the solution.

Kinetic scheme representations provide a great deal of leverage because a single compact expression is equivalent to a large amount of C code. One special advantage from the programmer's point of view is that these expressions are independent of the solution method. Different solution methods require different code, but the NMODL translator generates this code automatically. This saves the user time and effort and ensures that all code expresses the same mechanism. Another advantage is that the NMODL translator handles the task of interfacing the mechanism to the remainder of the program. This is a tedious exercise that would require the user to have

special knowledge that is not relevant to neurophysiology and may change from version to version.

2.3 Models with Discontinuities.

2.3.1 Discontinuities in PARAMETERs. In the past, abrupt changes in PARAMETERs and ASSIGNED variables, such as the sudden change in current injection during a current pulse, have been implicitly assumed to take place on a time-step boundary. This is inadequate with variable time-step methods because it is unlikely that a time-step boundary will correspond to the onset and offset of the pulse. Worse, the time step may be longer than the pulse itself, which may thus be entirely ignored.

For these reasons, a model description must explicitly notify NEURON, via the `at_time()` function, of the times at which any discontinuities occur. The statement `at_time(`*event_time*`)` guarantees that during simulation with a variable time-step method, as `t` advances past *event_time*, the integrator will reduce the step size so that it completes at `t = `*event_time*` - `ε, where $\varepsilon \sim 10^{-9}$ ms. The next step resets the integrator to first order, thereby discarding any previous solution history, and immediately returns after computing all the dy_i/dt at `t = `*event_time*` + `ε. Note that `at_time()` returns "true" only during the "infinitesimal" step that ends at `t = `*event_time*` + `ε.

During a variable time-step simulation, a missing `at_time()` call may cause one of two symptoms. If a PARAMETER changes but returns to its original value within the same interval, the pulse may be entirely missed. More often a single discontinuity will take place within a time-step interval, in which case what seems like a binary search will start for the location of the discontinuity in order to satisfy the error tolerance on the step; this is very inefficient.

Time-dependent PARAMETER changes at the hoc interpreter level are highly discouraged because they cannot currently be properly computed in the context of variable time steps. For instance, with fixed time steps, it was convenient to change PARAMETERs prior to `fadvance()` calls, as in

```
proc advance() {
  IClamp[0].amp = imax*sin(w*t)
  fadvance()
}
```

With variable time-step methods, all time-dependent changes must be described explicitly in a model, in this case with

```
BREAKPOINT { i = imax*sin(w*t) }
```

A future version of NEURON may provide a facility to specify time-dependent and discontinuous PARAMETER changes safely at the hoc level in the context of variable time step methods. (The BREAKPOINT block is discussed fully in the expanded version of this article. Its purpose is to declare the integration method for calculating the time evolution of STATEs and assign values to the currents declared by the mechanism.)

2.3.2 Discontinuities in STATEs. Some kinds of synaptic models process an event as a discontinuity in one or more of their STATE variables. For example, a synapse whose conductance follows the time course of an alpha function (for more detail about the alpha function itself see Rall, 1977, and Jack, Noble, & Tsien, 1983) can be implemented as a kinetic scheme in the two-state model

```
KINETIC state {
  ~ a <-> g (k, 0)
  ~ g -> (k)
}
```

where a discrete synaptic event is handled as an abrupt increase of STATE a. This formulation has the attractive property that it can handle multiple streams of events with different weights, so that g will be the sum of the individual alpha functions with their appropriate onsets.

However, because of the special nature of states in variable time-step ODE solvers, it is necessary not only to notify NEURON about the time of the discontinuity with the at_time(onset) call, but also to notify NEURON about any discontinuities in STATEs. If onset is the time of the synaptic event and gmax is the desired maximum conductance change, this would be accomplished by including a state_discontinuity() call in the BREAKPOINT block, as follows:

```
BREAKPOINT {
  if (at_time(onset)) {
    : scale factor exp(1) = 2.718... ensures
    : that peak conductance will be gmax
    state_discontinuity(a, a + gmax*exp(1))
  }
  SOLVE state METHOD sparse
  i = g*(v - e)
}
```

The first argument to state_discontinuity() will be assigned the value of its second argument just *once* for any time step. This is important, since for several integration methods, BREAKPOINT assignment statements are often executed twice to calculate the di/dv terms of the Jacobian matrix.

Although this synaptic model works well with deterministic stimulus trains, it is difficult for the user to supply the administrative hoc code for managing the `onset` and `gmax` variables to take advantage of the promise of multiple streams of events with different weights. The most important problem is how to save events that have significant delay between their generation and their handling at time `onset`. As is, an event can be passed to this model by assigning values to `onset` and `gmax` only after the previous onset event has been handled.

Discussion of the details of how NEURON now treats streams of synaptic events with arbitrary delays and weights is beyond the scope of this article. Let it suffice that from the local view of the postsynaptic model, the state discontinuity should no longer be handled in the `BREAKPOINT` block, and the above synaptic model is more properly written in the form

```
BREAKPOINT {
  SOLVE state METHOD sparse
  i = g*(v - e)
}
NET_RECEIVE(weight (microsiemens)) {
  state_discontinuity(a, a + weight*exp(1))
}
```

in which event distribution is handled internally from a specification of network connectivity (see the next section).

2.4 General Comments About Synaptic Models. The examples so far have been of mechanisms that are local in the sense that an instance of a mechanism at a particular location on the cell depends only on `STATE`s and `PARAMETER`s of the model *at that location*. Of course, they normally depend on voltage and ionic variables as well, but these also are *at that location* and automatically available to the model. Synaptic models have an essential distinguishing characteristic that sets them apart: in order to compute their contribution to membrane current at the postsynaptic site, they require information from another place, for example, presynaptic voltage. Models that contain `LONGITUDINAL_DIFFUSION` are perhaps also an exception, but their dependence on adjacent compartment ion concentration is handled automatically by the translator.

In the past, model descriptions could only use `POINTER` variables to obtain their presynaptic information. A `POINTER` in NMODL holds a reference to another variable; the specific reference is defined by a hoc statement such as

```
setpointer postcell.synapse.vpre, precell.axon.v(1)
```

in which `vpre` is a `POINTER`, declared in the indicated `POINT_PROCESS`

synapse instance, which references the value of a specific membrane voltage—in this case, at the distal end of the presynaptic axon. Gap junctions or ephaptic synapses can be handled by a pair of POINT_PROCESSes on the two sides of the junction that point to each other's voltage, as in

```
section1 gap1 = new Gap(x1)
section2 gap2 = new Gap(x2)
setpointer gap1.vpre, section2.v(x2)
setpointer gap2.vpre, section1.v(x1)
```

This kind of detailed piecing together of individual components is acceptable for models with only a few synapses, but larger network models have required considerable administrative effort from users to create mechanisms that handle synaptic delay, exploit very great simulation efficiencies available with simplified models of synapses, and maintain information about the connectivity of the network.

The experience of NEURON users, especially Alain Destexhe and William Lytton, in creating special models and procedures for managing network simulations has been incorporated in a new built-in network connection (NetCon) class, whose instances manage the delivery of presynaptic threshold events to postsynaptic POINT_PROCESSes. The NetCon class works for all NEURON integrators, including a local variable time-step method in which each cell is integrated with a time step appropriate to the state changes occurring in that cell. With this event delivery system, model descriptions of synapses never need to queue events, and they do not have to make heroic efforts to work properly with variable time-step methods. These features offer enormous convenience to users.

NetCon connects a presynaptic variable such as voltage to a synapse with arbitrary (individually specified on a per NetCon instance) delay and weight. If the presynaptic variable passes threshold at time t, a special NET_RECEIVE procedure in the postsynaptic POINT_PROCESS is called at time t + delay. The only constraint on delay is that it be nonnegative. Events always arrive at the postsynaptic object at the interval delay after the time they were generated, and there is no loss of events under any circumstances.

This new class also reduces the computational burden of network simulations, because the event delivery system for NetCon objects supports unlimited fan-in and fan-out (convergence and divergence). That is, many NetCon objects can be connected to the same postsynaptic POINT_PROCESS (fan-in). This yields large efficiency improvements because a single set of equations for synaptic conductance change can be shared by many streams of inputs (one input stream per connecting NetCon instance). Likewise, many NetCon objects can be connected to the same presynaptic variable (fan-out), thus providing additional efficiency improvement since the presynaptic variable is checked only once per time step, and when it crosses

threshold in the positive direction, events are generated for each connecting `NetCon` object.

3 Discussion

The model description framework has proved to be a useful, efficient, and flexible way to implement computational models of biophysical mechanisms. The leverage that NMODL provides is amplified by its platform independence, since it runs in the MacOS, MSWindows, and UNIX/Linux environments. Another important factor is its use of high-level syntax, which allows it to incorporate advances in numerical methods in a way that is transparent to the user.

NMODL continues to undergo revision and improvement in response to the evolving needs of computational neuroscience, particularly in the domain of empirically based modeling. One recent example of the extension of NMODL to encompass new kinds of mechanisms is longitudinal diffusion. Another is kinetic schemes in a form that can be interpreted as Markov processes (Colquhoun & Hawkes, 1981), that is, linear schemes, which are now translated into single-channel models. By removing arbitrary limits related to programming complexity, such advances give NEURON the ability to accommodate insights derived from new experimental findings and enable modeling to keep pace with the broad arena of "wet-lab" neuroscience.

Acknowledgments

This work was supported in part by NIH grant NS11613. We thank John Moore for inspiration and encouragement, Alain Destexhe and William Lytton for invaluable contributions to the convenient and efficient simulation of networks, Ragnhild Halvorsrud for helpful suggestions regarding the manuscript for this article, and the many users of NEURON who have provided indispensable feedback, presented challenging problems that have stimulated new advances in the program, and developed their own enhancements.

References

Colquhoun, D., & Hawkes, A. G. (1981). On the stochastic properties of single ion channels. *Philosophical Transactions of the Royal Society of London Series B, 211*, 205–235.

Frankenhaeuser, B., & Hodgkin, A. L. (1956). The after-effects of impulses in the giant nerve fibers of *Loligo*. *J. Physiol., 131*, 341–376.

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

Jack, J. J. B., Noble, D., & Tsien, R. W. (1983). *Electric current flow in excitable cells*. London: Oxford University Press.

Johnston, D., & Wu, S. M.-S. (1995). *Foundations of cellular neurophysiology*. Cambridge, MA: MIT Press.

Kohn, M. C., Hines, M. L., Kootsey, J. M., & Feezor, M. D. (1994). A block organized model builder. *Mathematical and Computer Modelling, 19*, 75–97.

Kootsey, J. M., Kohn, M. C., Feezor, M. D., Mitchell, G. R., & Fletcher, P. R. (1986). SCoP: An interactive simulation control program for micro- and minicomputers. *Bulletin of Mathematical Biology, 48*, 427–441.

McCormick, D. A. (1998). Membrane properties and neurotransmitter actions. In G. M. Sheperd (Ed.), *The synaptic organization of the brain* (pp. 37–75). New York: Oxford University Press.

Rall, W. (1977). Core conductor theory and cable properties of neurons. In E. R. Kandel (Ed.), *Handbook of physiology, vol. 1, part 1: The nervous system* (pp. 39–98). Bethesda, MD: American Physiological Society.

Wilson, M. A., & Bower, J. M. (1989). The simulation of large scale neural networks. In C. Koch & I. Segev (Eds.), *Methods in neuronal modeling* (pp. 291–333). Cambridge, MA: MIT Press.

Received September 30, 1998; accepted April 20, 1999.