To appear in The Handbook of Brain Theory and Neural Networks, second edition
(M.A. Arbib, ed.), Cambridge, MA: The MIT Press, 2002.
http://mitpress.mit.edu

The NEURON Simulation Environment

M.L. Hines[1] and N.T. Carnevale[2]

Departments of [1]Computer Science and [2]Psychology

Yale University

michael.hines@yale.edu

ted.carnevale@yale.edu

Short title: The NEURON simulation environment

Address correspondence to:

Michael L. Hines
Department of Computer Science
Yale University
PO Box 208001
New Haven, CT 06520−8001
telephone 203−737−4232
fax 203−785−6990

INTRODUCTION

NEURON is designed to be a convenient and efficient environment for simulating models of biological and artificial neurons, individually and in networks. Great care has been exercised at every point in its development to achieve computational efficiency and robustness while helping users maintain conceptual clarity, i.e. the knowledge that what has been instantiated in the computer is an accurate implementation of one's conceptual model. NEURON's application domain extends beyond continuous system simulations of models of individual neurons with complex anatomical and biophysical properties, and includes discrete−event and hybrid simulations that combine "biological" and "artificial" neuronal models. Here we review features of NEURON that are of special interest to prospective users.

WHO USES NEURON, AND WHY?

At the time of this publication, over 300 papers have reported research performed with NEURON (see http://www.neuron.yale.edu/neuron/bib/usednrn.html). Among these are descriptions of models of individual neurons and networks of neurons with properties such as complex branching morphology, multiple channel types, inhomogeneous channel distribution, ionic diffusion and buffering, active transport, second messengers, and use−dependent synaptic plasticity. At the cellular level, NEURON has been used to investigate topics that include pre− and post−synaptic mechanisms involved in synaptic transmission, the roles of dendritic architecture and active membrane properties in synaptic integration, spike initiation and propagation in dendrites and axons, the effects of developmental changes of anatomy and biophysics, functional genomics of ion channels, and extracellular stimulation and recording. Network models implemented with NEURON have been used to address issues such as the origin of cortical and thalamic oscillations, the role of gap junctions in neuronal synchrony, information encoding in biological networks, visual orientation selectivity, mechanisms of epilepsy, and the actions of anticonvulsant drugs.

NEURON is also being used in neuroscience education at the undergraduate and graduate level at numerous universities across the United States and around the world. Many of these courses are completely home−grown, but one lab manual with exercises has already appeared in print (Moore and Stuart 2000), and the authors of this review are involved in a collaboration to develop another set of laboratory exercises for publication. NEURON is particularly well−suited to educational applications, since special expertise in numerical methods or programming are not required for its productive use. Furthermore, NEURON runs under MacOS, MSWindows, and UNIX/Linux, and can execute research−quality simulations with reasonable runtimes on entry−level hardware.

HOW DOES NEURON WORK?

Historically, NEURON's primary domain of application was in simulating empirically−based models of biological neurons with extended geometry and biophysical mechanisms that are spatially nonuniform and kinetically complex. In the past decade its functionality has been enhanced to include extracellular fields, linear circuits to emulate the effects of nonideal instrumentation, models of artificial (integrate and fire) neurons, and networks that can involve

any combination of artificial and biological neuron models. The following paragraphs outline how these capabilities have been implemented so as to achieve computational efficiency while maintaining conceptual clarity, i.e. the knowledge that what has been instantiated in the computer model is an accurate representation of the user's conceptual model.

Representing biological neurons

Information processing in the nervous system involves the spread and interaction of electrical and chemical signals within and between neurons and glia. These signals are continuous functions of time and space and are described by the diffusion equation and the closely−related cable equation (Crank 1979, Rall 1977). To simulate the operation of biological neurons, NEURON uses the tactic of discretizing time and space, approximating these partial differential equations by a set of algebraic difference equations that can be solved numerically (numerical integration) (Hines and Carnevale 1997).

Discretization is often couched in terms of "compartmentalization," but it is perhaps better to regard it as an approximation of the original continuous system by another system that is discontinuous in time and space. Simulating a discretized model results in computation of the values of spatiotemporally continuous variables over a set of discrete points in space ("nodes") for a finite number of instants in time. If NEURON's second order correct integration method is used, these values are a piecewise linear approximation to the continuous system, so that second order accurate estimates of continuous variables at intermediate locations can be found by linear interpolation (Hines and Carnevale 2001).

In one form or another, spatial discretization lies at the core of all simulators used to model biological neurons (e.g. THE GENESIS SIMULATION SYSTEM). Unlike other simulators, however, NEURON does not force users to deal with compartments. Instead NEURON's basic building block is the "section," an unbranched, continuous cable whose anatomical and biophysical properties can vary continuously along its length. The branched architecture of a cell is reconstructed by connecting sections together, each section having its own anatomical dimensions, biophysical properties, and discretization parameter `nseg`, which specifies the number of nodes at which solutions are computed. This strategy makes it easier to manage anatomically detailed models, since each section in the model is a direct counterpart to a branch of the original cell (Fig. 1). Furthermore, neuroscientists naturally tend to think in terms of axonal or dendritic branches rather than compartments.

But even in topologically simple cases, there is still the problem of how to treat variables that are continuous functions of space. Thinking in terms of compartments leads to representations that require users to keep track of which compartments correspond to which anatomical locations. If we change the size or number of compartments, e.g. in order to see whether spatial discretization is adequate for numerical accuracy, we must also abandon the old mapping between compartments and locations in favor of a completely new one.

This is the motivation for another strategy that helps NEURON users maintain conceptual clarity: *range* and *range variables*. Range variables are continuous functions of position along a branch of a cell, e.g. diameter, membrane potential, ion channel density. NEURON deals with range variables in terms of arc length (normalized distance) along the centroid of each section. This normalized distance, which is called range, is a continuous parameter that varies from 0 at

one end to 1 at the other. In NEURON's programming language `hoc`, the membrane potential at a point 700 μm down the length of a 1000 μm long axon would be called `axon.v(0.7)`, regardless of the value of the axon's discretization parameter `nseg`. Range and range variables allow NEURON itself to take care of the correspondence between nodes and anatomical location. This avoids the tendency of compartmental approaches to confound representation of the physical properties of neurons, which are biologically relevant, with implementational details such as compartment size, which are mere artifacts of having to use a digital computer to emulate the behavior of a distributed physical system that is continuous in time and space.

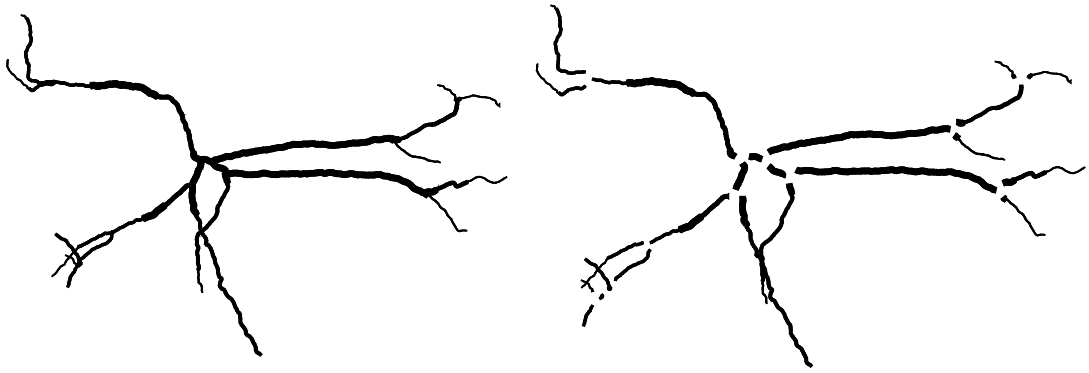

Figure 1. Left: Morphometric reconstruction of a hippocampal interneuron (data from A.I. Gulyás). Right: An "exploded" view, in which individual, unbranched neurites have been separated from each other at branch points.

Representing artificial neurons

In NEURON the basic difference between biological and artificial neuron models is that the former may have arbitrarily complex anatomical and biophysical complexity, while the latter have no spatial extent and employ highly simplified kinetics. Indeed, the three built−in classes of artificial spiking neurons are so simple that they are simulated using a discrete−event method, which executes hundreds of times faster than numerical integration methods. If an event occurs at time t_1, all state variables are computed from the state values and time t_0 of the previous event. Since computations are performed only when an event occurs, total computation time is proportional to the number of events delivered and independent of the number of cells, number of connections, or problem time. Thus handling 100,000 spikes in one hour for 100 cells requires the same time as handling 100,000 spikes in 1 second for 1 cell. This takes advantage of NEURON's event delivery system, which was originally implemented to facilitate efficient network simulations of biological neurons (see following section).

Three different classes of integrate and fire models are built into NEURON. The simplest is IntFire1, a leaky integrator that treats input events as weighted delta functions. When an IntFire1 cell receives an input event of weight *w*, its "membrane potential" state *m* jumps instantaneously by an amount equal to *w* and thereafter resumes its decay toward 0 with time constant τ_m.

A step closer toward the behavior of a biological neuron is the IntFire2 mechanism, which differs in that *m* integrates a net synaptic current *i*. An input to an IntFire2 cell makes the synaptic current jump by an amount equal to the synaptic weight, after which *i* continues to decay toward a steady level i_b with its own time constant τ_s, where $\tau_s > \tau_m$. Thus a single input

event produces a gradual change in *m* with a delayed peak, and cell firing does not obliterate all traces of prior synaptic activation. The firing rate is ~ i / τ_m if $i >> 1$ and $\tau_s >> \tau_m$.

While IntFire2 can emulate a wide range of relationships between input pattern and firing rate, all inputs produce responses with the same kinetics regardless of whether they are excitatory or inhibitory. The fact that synaptic excitation in biological neurons is generally faster than inhibition inspired the design of IntFire4. IntFire4 integrates two synaptic current components that have different dynamics depending on whether the input event is excitatory or inhibitory (Fig. 2). Excitatory inputs add instantaneously to an excitatory synaptic current *e*, which otherwise decays toward 0 with a single time constant τ_e; this is analogous to IntFire2 with $i_b = 0$. However, the inhibitory synaptic current i_2 is described by the reaction scheme

$$i_1 \xrightarrow{1/\tau_{i1}} i_2 \xrightarrow{1/\tau_{i2}} bath \qquad (1)$$

where an inhibitory input (weight $w < 0$) adds instantaneously to i_1, so that i_2 follows a biexponential course (a slow rise followed by an even slower decay).

These are not the only kinds of artificial neurons that can be simulated using discrete events. The only prerequisite for discrete event simulations is that all state variables of a model cell can be computed analytically from a new set of initial conditions. Users who have special needs can add other kinds of artificial neuron classes to NEURON with the NMODL language (see below).

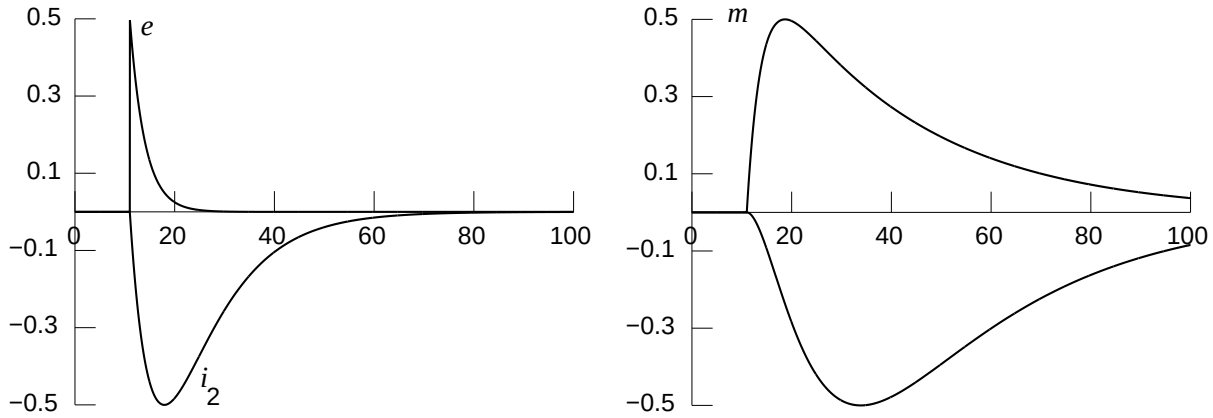

Figure 2. Left: Current generated by a single input event to an IntFire4 cell with weight 0.5 (*e*) or −0.5 (i_2). Right: The corresponding response of *m*. Parameters are $\tau_e = 3$, $\tau_{i1} = 5$, $\tau_{i2} = 10$, and $\tau_m = 30$ ms.

Representing networks

NEURON can handle networks that involve gap junctions or graded transmitter release, but because of space limitations this discussion is restricted to spiking networks. NEURON's NetCon class and event delivery system are used to manage synaptic communication between any combination of artificial and biological model neurons in a network. The event delivery system can also serve other purposes such parameter changes on the fly, and it is a key part of

the implementation of the built−in integrate and fire models, but these topics are beyond the scope of this paper.

NEURON's strategy for dealing with synaptic connections emerged from techniques initially developed by Destexhe et al. (1994) and Lytton (1996). This strategy is based on a very simple conceptual model of synaptic transmission: arrival of a spike at the presynaptic terminal causes transmitter release, which in turn perturbs some mechanism in the postsynaptic cell (e.g. a membrane conductance or second messenger) that is described by a differential equation or kinetic scheme. All that matters is whether or not a spike has occurred in the presynaptic cell; mechanistic details in the presynaptic and postsynaptic cells do not affect transmitter release. This conceptual model separates the specification of the connections between cells from the specification of the postsynaptic mechanisms that the connections activate.

A presynaptic spike triggers an event which, after a delay to account for conduction along the axon, transmitter release, and diffusion time, is delivered to the postsynaptic mechanism where it causes a change in some variable (e.g. a conductance). Event delivery is computationally efficient because of how synaptic divergence ("fan out") and convergence ("fan in") are handled. Synaptic divergence from model biological neurons is efficient because threshold detection is performed on a "per source" basis, rather than a "per connection" basis. That is, if a neuron projects to multiple targets, the presynaptic variable is checked only once per time step, and when it crosses threshold an event is generated for each of the targets. Fan−out from artificial neurons is also very efficient, since their discrete event mechanisms do not have to be checked at each time step. However, the greatest computational savings are offered by synaptic convergence onto model biological neurons. Suppose a neuron receives multiple synaptic inputs that are close to each other and of the same type, i.e. each synapse has the same kind of postsynaptic mechanism. Then the total effect of all these synapses can be represented by a single kinetic scheme or set of equations driven by multiple input streams, each of which has its own weight.

The event delivery service implemented in NEURON takes into account the fact that the delay between initiation and delivery of events is different for different streams. Consequently the order in which events are generated by a set of sources is rarely the order in which they are received by a set of targets. Furthermore the delay may be 0 or 10^9 or anything in between.

As noted above, NEURON separates specification of the connections between cells from specification of the postsynaptic mechanisms that the connections activate. This separation means that NEURON models are compatible with other event delivery systems, such as the parallel discrete event delivery system used by NeoSim (Goddard et al. 2001).

Integration methods

Users can choose among several different integration methods, but which is "best" for any given problem depends on many factors and may require empirical testing. Every choice involves tradeoffs between accuracy, on the one hand, and stability and/or run time, on the other. For more extensive treatments of this topic see (Hines and Carnevale 1995, 1997, 2001).

Two methods use fixed time steps: backward Euler, and a Crank−Nicholson variant. NEURON's default integrator is backward Euler, which is first order accurate in time, inherently stable, and generally produces good qualitative results even with large time steps. Used with

extremely large Δt, it will find the steady−state solution of a linear ("passive") model in one step, and quickly converge to a steady−state for nonlinear models. The Crank−Nicholson (CN) variant employs a staggered time step in order to provide second order accuracy without having to iterate nonlinear equations. The computational cost of a single time step is practically the same as for backward Euler, but much shorter run times are possible with CN because it can use a larger Δt for a given degree of accuracy. However CN cannot be used with models that involve purely algebraic relations between states, and it can produce spurious oscillations if Δt is too large or the model includes a fast voltage clamp.

Models that work with CN are generally also amenable to CVODE (Cohen and Hindmarsh 1984), the variable order, variable time step method offered by NEURON. For a given run time, CVODE often yields greater accuracy than CN. CVODE is usually the best choice for network models that involve artificial neurons. NEURON also offers a local variable order, variable time step method, in which each cell has its own time step. This can be advantageous for network models in which most cells are silent most of the time.

DEVELOPMENT ENVIRONMENT

Constructing and managing models and controlling simulations can be accomplished with an object−oriented interpreter, a set of GUI tools, or a combination of both. Most common tasks can be performed with the GUI tools, which are especially convenient for exploratory simulations during model development. Where the GUI is inadequate, users can resort to the interpreter, which is based on `hoc` (Kernighan and Pike 1984). The interpreter is also appropriate for noninteractive simulations, such as "production" runs that generate large amounts of data for later analysis. Even so, several of the GUI tools are quite powerful in their own right, offering functionality that would require significant effort for users to recreate in `hoc`. This is particularly true of the optimization and electrotonic analysis tools. Thus the most flexible and productive use of NEURON is to combine the GUI and `hoc` programming, taking advantage of the strengths of both.

Because of the ever growing number and diversity of ligand− and voltage−gated ionic currents, pumps, buffers, etc., NEURON has a special facility for expanding its library of biophysical mechanisms (Hines and Carnevale 2000). A user can write a text file that contains a description of the mechanism in NMODL, a programming language whose syntax for expressing nonlinear algebraic equations, differential equations, and kinetic reaction schemes closely resembles familiar notation. This file is then converted into C by a translator that automatically generates code for handling details such as mass balance for each ionic species. The translator output, which includes code that is suitable for each of NEURON's integration methods, is then compiled for computational efficiency. This achieves tremendous conceptual leverage and savings of effort because the high−level mechanism specification in NMODL is much easier to understand and far more compact than the equivalent C code, and the user is not bothered with low−level programming issues like how to interface the code with other mechanisms and with NEURON itself.

NEURON runs under MacOS, MSWindows, and UNIX/Linux, with a similar X11−based look and feel on all platforms. Furthermore, the same `hoc` and NMODL code works under all

these operating systems without modification. This facilitates collaborations in a heterogeneous computing environment.

DISTRIBUTION, DOCUMENTATION AND SUPPORT

NEURON is available free of charge from http://www.neuron.yale.edu/, along with extensive documentation and tutorials. The UNIX/Linux distribution includes full source code; the MSWin and MacOS distributions employ an identical computational engine, and come with the hoc code that implements the GUI and the NMODL definitions of the built−in biophysical mechanisms and artificial neuron classes.

The WWW site also has a signup page for joining the NEURON Users' Group, a moderated mailing list for questions and answers, and pertinent announcements such as program updates and courses on NEURON. For the past several years we have presented "executive summary" and intensive hands−on courses at sites in the USA and Europe, and we plan to continue this in the future. NEURON is actively supported by a development team that responds to bug reports and questions about program usage. Indeed, much of the program's current functionality has been stimulated by requests and suggestions from users, and we are grateful to them for their continued interest and encouragement.

DISCUSSION

The level of detail included in NEURON models can extend from a single compartment with linear membrane, to intricate extended architectures with membrane and cytoplasm that have complex biophysical properties. There are also several classes of artificial spiking neurons. Networks can involve biological neurons, artificial neurons, or both. Models can be simulated with fixed time step or with variable order, variable time step methods. With the variable time step integrator, the built−in artificial neurons are simulated as discrete event models, executing orders of magnitude faster than models of biological neurons do. Users can add new kinds of biophysical mechanisms and artificial neuron classes to NEURON's built−in library. NEURON runs on a wide variety of platforms, is actively supported, and is under continuous development with revisions and updates that address the evolving needs of users. Because of these features, NEURON is employed in research on topics that range from the biophysical basis of neuronal function at the subcellular level, to the operation of large scale networks involved in consciousness, perception, learning, and memory, and it is also increasingly being adopted for neuroscience education.

REFERENCES

Cohen, S.D. and Hindmarsh, A.C.. CVODE User Guide. Livermore, CA: Lawrence Livermore National Laboratory, 1984.

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

Destexhe, A., Mainen, Z.F., and Sejnowski, T.J., 1994, An efficient method for computing synaptic conductances based on a kinetic model of receptor binding. Neural Computation, 6:14−18.

*Goddard, N., Hood, G., Howell, F., Hines, M., and De Schutter, E., 2001, NEOSIM: portable large−scale plug and play modelling, Neurocomputing, 38:1657−1661.

*Hines, M. and Carnevale, N.T., 1995, Computer modeling methods for neurons, in The Handbook of Brain Theory and Neural Networks, edition 1, (M.A. Arbib, Ed.), Cambridge, MA: MIT Press, pp. 226−230.

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

*Hines, M.L. and Carnevale, N.T., 2000, Expanding NEURON's repertoire of mechanisms with NMODL, Neural Computation, 12:839−851.

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

*Kernighan, B.W. and Pike, R., 1984, Appendix 2: Hoc manual, in The UNIX Programming Environment, Englewood Cliffs, NJ: Prentice−Hall, pp. 329−333.

Lytton, W.W. 1996, Optimizing synaptic conductance calculation for network simulations, Neural Computation, 8:501−509.

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

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