Hines, Michael (1993). NEURON - A program for simulation of nerve equations. In: *Neural Systems: Analysis and Modeling.* pp.127–136. F. Eeckman (ed), Kluwer Academic Publishers.

NEURON – A Program for Simulation of Nerve Equations

Michael Hines
Dept. of Neurobiology
Duke University Medical Center
Durham, NC 27710

Introduction

Programs designed specifically to simulate nerve equations compare favorably with general purpose simulation programs in three areas. 1) The user deals directly with concepts that are familiar at the neuroscience level and is not required to translate the problem into another domain. 2) The program contains functions better suited for controlling the simulation and graphing the results of real neurophysiological problems. 3) Special methods and tricks can be used to take advantage of the structure of nerve equations to solve them much more quickly, e.g. Hines (1984) and Mascagni (1991).

However, the general domain of nerve simulation is still too large for any single program to optimally deal with every problem. In practice, programs have their genesis in the attempt to solve a much restricted class of problems — although sometimes provision is made in the underlying implementation to incrementally extend the domain of applicability. Programs do one kind of problem best but degrade, as the problem changes, both in allowing the user to maintain conceptual control and in speed of simulation.

This article describes a program, NEURON, developed in collaboration with John W. Moore, written in C, and with source code freely available to any interested person. With NEURON, nerve properties are specified, the simulation controlled, and the results graphed by writing procedural statements to an interpreter. This has the advantage of generality but the disadvantage that it is more difficult to learn than a menu driven interface. The usefulness of NEURON degrades very slowly with increased complexity of morphology and

membrane mechanisms. NEURON is best suited, in terms of efficiency, for problems ranging from parts of single cells to small numbers of cells in which cable properties play a crucial role. It is best suited, in terms of conceptual control, for stylized morphologies represented as connected cable sections and where membrane channel parameters are conveniently represented as piece-wise linear functions of position within each section. Two special classes of problems for which it is well suited are those in which it is important to calculate ionic concentrations and those where one needs to compute the extracellular potential just next to the nerve membrane. It is well suited for investigating new kinds of membrane channels since they are described using a high level model description language which allows the expression of models in terms of kinetic schemes or sets of simultaneous equations.

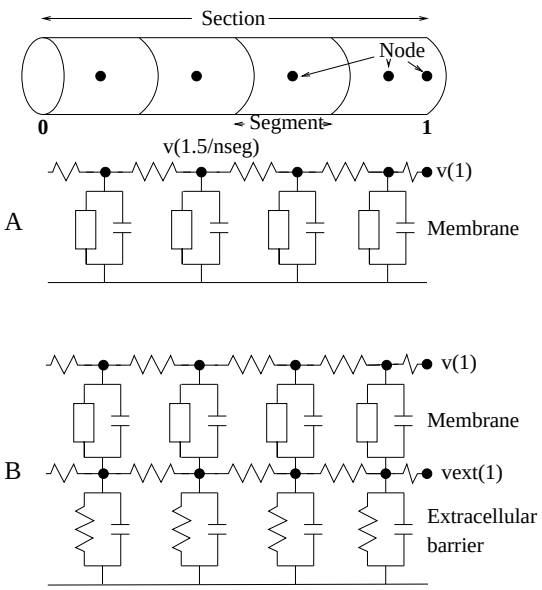

Figure 1: Section representation as electrical equivalent circuit. B shows the representation when the "extracellular" mechanism is used.

Every nerve simulation program solves for the longitudinal spread of voltage and current by approximating the cable (parabolic partial differential) equation as a series of compartments connected by resistors (Fig. 1). The compartments represent a membrane area that contains all the channels and other mechanisms that govern the total transmembrane current. The sum of all the compartment areas is the total membrane area of the whole nerve. Unfortunately, it is usually not clear at the outset how many compartments should be used. The accuracy of the approximation as well as the computation time increases as the number of compartments used to represent the cable increases. When the cable is "short",

a single compartment can be made to adequately represent the entire cable. For long cables or highly branched structures, it may be necessary to use a very large number of compartments.

The problem of "conceptual control" is how best to manage all the parameters which exist within these compartments. For example, consider the membrane capacitance. The values are different in every compartment but rather than specify them all individually it is better to deal in terms of a single specific membrane capacitance that is constant over the entire cell and have the program compute the values of the individual capacitances within the compartments based on the area of the compartments.

NEURON is a program designed around the notion of continuous cable "sections" which are connected together to form any kind of branched tree structure. The value of this notion lies in the ability to specify the physical properties of a neuron without regard for the purely numerical issue of how many compartments will be used to represent each of the cable sections. This means that one can easily trade off between accuracy and speed and makes for convenient verification of the numerical correctness of the simulation.

Numerical Methods

Spatial discretization

Figure 1 showed how NEURON represents a cable section in terms of a discrete set of nodes connected by resistors. The section is divided into a number (denoted by the section parameter, `nseg`) of segments of equal length. A node (location where the internal voltage is defined) is placed at the center of each segment along with one extra node (representing 0 area) at the distal end of the section. A node at the center of a segment represents the entire area of the segment. The distal node represents 0 area and facilitates numerical computation of membrane potential when multiple branches are attached to the end of a section. NEURON takes care to calculate the value of the circuit parameters based on the position of these nodes and it can be shown that the spatial accuracy is proportional to the square of the number of segments.

Sections which have the "extracellular" membrane mechanism inserted are represented by an extra layer of of nodes representing the extracellular potential just next to the membrane (bottom of figure 1).

Time integration

NEURON uses a fully implicit (backward euler) integration method to compute the membrane voltage (cf. Hines, 1989). This method has the advantage of being numerically stable for large time steps and, in fact, a single large time step suffices to determine the steady state for passive neurons. Although the implicit method gives good qualitative results for large time steps the simulation error is proportional to the integration time step. The results of this method for two

time steps can be seen in the left panel of figure 2 which shows the membrane

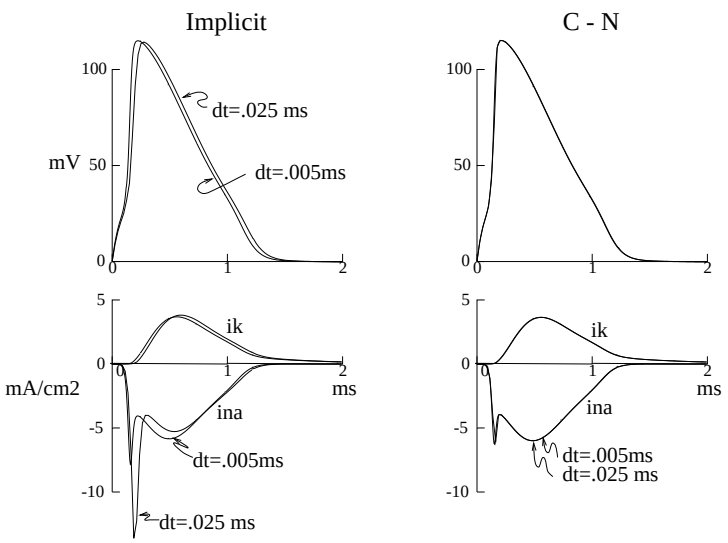

Figure 2: Frankenhaeuser - Huxley action potential: Comparison of implicit and Crank-Nicholson methods with two time steps.

voltage and ionic currents for an action potential using the Frankenhaeuser-Huxley (1964) model of the frog node. It is clear that the accurate simulation of fast changing ionic currents requires smaller time steps than that needed for the membrane potential.

A more accurate integration method (analogous to that of Crank and Nicholson, 1947) can be selected which has numerical errors proportional to the square of the time step. In fact, the simulation using the .025 ms time step in the right panel of figure 2 is more accurate than the .005 ms time step simulation with the fully implicit method. The reason the implicit method is used by default is that the C-N method doesn't work with fast voltage clamps or when the resistance coupling adjacent segments is very small. The C-N method may produce membrane potential artifacts that oscillate with large amplitude when the time step becomes too large.

Both integration methods use several efficiency tricks (Hines, 1984) that make simulations run faster. First, the geometry of the neuron is limited to a tree like structure which allows a fast direct method for solving the simultaneous equations representing charge conservation at each voltage node. Simulation time is proportional to the total number of nodes plus the total number of branches. Second, iterative solution of the nonlinear simultaneous equations is avoided by calculating channel states at the midpoint of each time step. This saves a lot of time but makes results a bit more complicated since for the C-

N method the user must be aware that voltage and concentration are most accurate at the indicated value of the time variable, `dt`, but channel states are second order correct at time, `t + dt/2`, and currents are second order correct at time, `t - dt/2`. Third, integration of channel states can be done analytically (actually second order since the voltage dependent rates use the voltage at the midpoint of the integration step) in just a single addition and multiplication operation and two table lookup operations. The tradeoff here is that the tables depend on the value of the time step and must be recomputed whenever the time step changes.

User Interface

The user interface uses broadly the same style as our previous nerve simulation program, CABLE (Hines, 1989). That is, simulation control at the user level is via a C-like interpreter called HOC (Kernighan and Pike, 1984). The built in editor for writing interpreted functions is microemacs. The integration of the nerve equations is accomplished by repeatedly calling a function which advances all the variables through a short time interval, dt.

The process of setting up and running a simulation consists in writing procedures to: 1) Define the physical properties of the neuron. This includes declaring the names of various components (continuous cable sections which may be short or long) of the cell, connecting these sections together, inserting membrane mechanisms (channels, synapses, stimulus electrodes, ionic concentrations, etc) into the sections, and setting the values of parameters for these membrane mechanisms. 2) Control the time course of the simulation. This is generally divided into at least two procedures. One initializes the membrane potential and the states of the inserted mechanisms (channel states, ionic concentrations, or extracellular potential next to the membrane). The second procedure repeatedly calls the built-in single step integration function and saves, plots, or computes functions of the desired variables at each step. In this procedure it is possible to change the value of membrane parameters during a run.

The flexibility of an interpreter makes it convenient to write procedures to handle families of related simulations, calculate new variables such as impulse propagation velocity, optimize parameters, search for thresholds, etc.

Cable Sections

The greatest problem with our previous program was that the user was responsible for knowing the correspondence between segment number and position on the nerve. All nerve properties were assigned via vector variables in which the index of the vector was the segment number. Changing the number of segments that describe the nerve was an error prone and laborious process.

This problem is overcome by the notion of a named section which can be thought of as a one-dimensional cable of length, `L`, (in μm) with a continuous

position parameter, $0 \leq x \leq 1$, in which $x = 1$ corresponds to absolute position `L`. Most properties are functions of the position parameter and are called "range variables". Examples are the diameter in μm, `diam`, the membrane potential in mV, `v`, and, if the Hodgkin - Huxley mechanism is inserted, the maximum HH sodium conductance in mho/cm^2, `gnabar`. Specifying the values of range variables does not depend on the number of segments (parameter `nseg`) used to represent the section.

Because the same names for parameters are used in all sections it is necessary for the user to specify which section is intended. The interpreter provides several methods to determine which is the currently specified section. For example, `forall {` *statements* `}`, executes the statements within the brackets once with every section name set to the currently specified section. Special functions are provided to determine the name of the currently specified section and whether a particular membrane mechanism is inserted in that section.

The simplest use of a range variable is to assign a constant value to it as in:

```
diam=10
```

This statement places the value, 10, in every segment of the currently specified section. For values which vary along the length of a section, the assignment syntax

`rangevar(` *xmin* `:` *xmax* `) =` *e1* `:` *e2*

with $0 \leq xmin \leq xmax \leq 1$ is used to store any piecewise linear function into the segments of the current section. The value of a range variable can appear in any expression using the syntax, "`rangevar(`x`)`", where $0 \leq x \leq 1$. The value is the value at the center of the segment containing point x.

Membrane Mechanisms — Model Description Language

Our previous program (CABLE) contained several built-in membrane mechanisms such as radial calcium diffusion, calcium channel, etc. However, in practice, only the Hodgkin-Huxley squid channels were enough of a standard to be used "as is" across more than one series of simulations. The other channels all required some type of modification to be useful as new situations arose. Sometimes the modifications were minor, such as changing the coordinate system for radial calcium diffusion so that there were more compartments near the membrane, but often we were forced to add an entirely new mechanism from scratch such as Frankenhaeuser-Huxley channels for Xenopus node. The problem was greatly compounded for other users of CABLE who needed to add new channels but were not familiar with the numerical issues or the detailed interface requirements.

The NEURON system overcomes this problem by allowing the incorporation of new membrane mechanisms that are defined using a high level model

description language, NMODL[1] (Kohn, et. al, 1989). NMODL with NEURON is a significant improvement over CABLE with regard to adding new membrane mechanisms:

- Interface details are handled automatically. Function tables are automatically generated.
- Consistency of units is ensured.
- Mechanisms described by a kinetic scheme are written with a syntax in which the reactions are clearly apparent.
- There is often a great increase in clarity since statements are at the model level instead of the C programming level and are independent of the numerical method.

At the same time, since the model description is translated into C, the computation speed remains the same or better than a user programmed mechanism in CABLE.

Types of mechanisms

The kinds of mechanisms that can be translated by NMODL and linked into NEURON are:

- Channels in which the model consists of current-voltage relationships. This includes HH, calcium channels, etc.
- Calculation of internal and external ionic concentration changes due to currents carried by specific ions. This includes radial calcium diffusion, extracellular potassium accumulation, etc.
- Point processes which are inserted at a particular point in a section instead of being distributed throughout a section. This includes current stimuli, voltage clamps, and synapses.

More than one user defined mechanism can be simultaneously inserted into sections and NEURON will keep track of the total current for each ionic species used and the effect of that current on the membrane potential.

Types of equations

Models consisting of a mixed set of nonlinear algebraic and differential equations are written using an expression syntax of the form

[1] The model description language was originally developed at the National Biomedical Simulation Resource to specify models for simulation with their "Simulation Control Program". Only modest extensions to the syntax were necessary to allow it to translate model descriptions into a form suitable for compiling and linking with NEURON.

```
x' = f(x, y, t)
~ g(x, y) = h(x, y)
```

where the prime refers to the derivative with respect to time (multiple primes such as `x''` refer to higher derivatives) and the tilde introduces an algebraic equation. The algebraic portion of such systems of equations is solved by Newton's method and a variety of methods are available for solving the differential equations, e.g. variable step Runge-Kutta or backward euler.

Chemical reactions and kinetic schemes are expressed using a reaction style syntax illustrated with

```
~ A + 2B <-> C + D   (k1, k2)
```

Where the `k1` and `k2` are the forward and reverse rate constants. As an example of the power and convenience of the kinetic scheme approach consider a stylized model of transmitter release at the nerve terminal in response to inward calcium current. Figure 3 illustrates a model which focuses on the diffusion of calcium

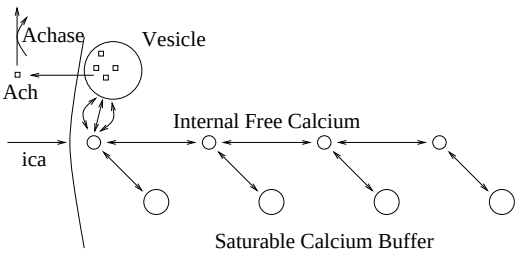

Figure 3: Calcium evoked transmitter release.

toward the interior of the terminal and its reaction with an immobile calcium buffer. At the same time, near the cell surface, three molecules of free calcium can react with a vesicle filled with neurotransmitter in such a way as to cause the vesicle to release its contents extracellularly. This transmitter is then destroyed by an enzyme.

The description of the kinetic process in the NMODL language follows. To save space the blocks which declare the parameters and variables have been left out as well as the procedure which computes factors that depend on the specific coordinate system. Ca[0] refers to the internal compartment closest to the membrane (there are N compartments indexed from 0 to N-1).

```
KINETIC calcium_evoked_release {

   : Release
 ~  Vesicle + 3Ca[0] <-> VRel   (Agen, Arev)
 ~  VRel <-> Ach (krel, 0)
 ~  Ach + Achase <-> Ach2ase    (Aasef, Aaseb)
```

```
    ~  Ach2ase <-> X + Achase      (Aase2, 0)

      : Buffering
      FROM i = 0 TO N-1 {
       ~  Ca[i] + Buffer[i] <-> CaBuffer[i]  (kCaB, kmCaB)
      }

      : Diffusion
      FROM i = 1 TO N-1 {
       ~  Ca[i-1] <-> Ca[i]      (Dca*f[i], Dca*f[i])
      }

      : Inward flux
    ~  Ca[0] <<     (ica*fac)

      COMPARTMENT i, vol[i] {Ca CaBuffer Buffer}
      COMPARTMENT    vol[0] {Vesicle VRel}
      COMPARTMENT    ext    {Ach Achase Ach2ase X}
  }
```

Especially note the elegance in the way calcium diffusion is specified. The vector parameters `vol[]` and `f[]` take into account the size and surface area of adjacent compartments. More importantly, modifications to the model (such as increasing the stoichiometry of release to four calcium ions) can be easily made without introducing errors.

The above KINETIC block is translated into a C function 150 lines long which explicitly sets up the associated matrix equation — clearly the leverage is tremendous. The generated code is very efficient and the solution method is stable for all time steps (Steady state solutions are found in one step when the time step is very large; equilibrium reactions are simulated with extremely large rate constants). Furthermore the simultaneous equations are solved using a direct sparse matrix method using minimum degree ordering resulting in a tremendous decrease in computation time over the standard full matrix method.

Units

The units checking feature of NMODL has proved invaluable. Its usefulness can be illustrated by the following example, especially if the reader takes the trouble to calculate the conversion factor on her/his own.

Consider the concentration change of sodium (μM) in a sphere of radius (μm) in response to a uniform inward current density (mA/cm^2) with time measured in (ms). The equation for the concentration change (the prime refers to the derivative with respect to time, the Faraday is measured in coulombs) is written in NMODL as

```
nai' = ina/FARADAY * (4*PI*radius^2)/(4/3*PI*radius^3)
```

but is missing a conversion factor to make the units consistent.

The units checker not only generates an error message stating that the above equation is inconsistent but suggests the proper conversion factor.

References

Crank, J. and Nicholson, P.N. (1947) A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type. *Proc. Cambridge Phil. Soc.* **43:** 50–67.

Frankenhaeuser, B. and Huxley, A.F. (1964). The action potential in the myelinated nerve fibre of Xenopus laevis as computed on the basis of voltage clamp data. *J. Physiol. (Lond.)* **171:** 302–315.

Hines, Michael. (1984). Efficient computation of branched nerve equations. *Int. J. of Biomed. Computing.* **15:** 69–76.

Hines, Michael. (1989). A program for simulation of nerve equations with branching geometries. *Int. J. of Biomed. Computing.* **24.** 55–68.

Kernighan, B.W. and Pike, R. (1984). *The Unix Programming Environment.* Prentice hall, Englewood Cliffs, New Jersey.

Kohn, M. C., Hines, M., Kootsey, J. M., and Feezor, M. D. (1989). A block organized model builder. *Proceedings of the symposium on physiological modeling at the 7th ICCM, Chicago.* In press.

Mascagni, Michael. (1991). A parallelizing algorithm for computing solutions to arbitrarily branched cable neuron models. *J. Neurosci. Methods.* **36:** 105–114.