The finer points of modeling (with NEURON)

Practical aspects of constructing and using models of cells and networks

Ted Carnevale
Yale University School of Medicine

Bill Lytton
SUNY Downstate Medical Center
Selected topics

1. Spatial discretization in NEURON
2. Some essential idioms
3. How to discover section names and other things
4. Combine hoc and the GUI
5. Discovering NEURON's hoc library
6. Customizing simulation execution
7. Customizing initialization

Other sources of information
1. Spatial discretization in NEURON

- The discretized cable equation
- Sections, segments, range, and range variables
- The discretization parameter nseg, and how to decide what value to select
- Implications for iterating over segments
The discretized cable equation

Conservation of charge

\[ C_m \frac{d V_m}{dt} + i_{ion} = \sum i_a \]
The model equations

\[ c_j \frac{dv_j}{dt} + i_{ion_j} = \sum_k \frac{v_k - v_j}{r_{jk}} \]

- \( v_j \): membrane potential in compartment j
- \( i_{ion_j} \): net transmembrane ionic current in compartment j
- \( c_j \): membrane capacitance of compartment j
- \( r_{jk} \): axial resistance between the centers of compartment j and adjacent compartments k
Separating anatomy and biophysics from purely numerical issues

section

a continuous length of unbranched cable

Anatomical data from A.I. Gulyás
### Range Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Meaning</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>diam</td>
<td>diameter</td>
<td>[µm]</td>
</tr>
<tr>
<td>cm</td>
<td>specific membrane capacitance</td>
<td>[µf/cm²]</td>
</tr>
<tr>
<td>g_pas</td>
<td>specific conductance of the pas mechanism</td>
<td>[siemens/cm²]</td>
</tr>
<tr>
<td>v</td>
<td>membrane potential</td>
<td>[mV]</td>
</tr>
</tbody>
</table>
range

normalized position along the length of a section

\[ 0 \leq \text{range} \leq 1 \]

any variable name can be used for range, e.g. \( x \)
**nseg**

the number of points in a section at which the discretized cable equation is integrated

- nseg=1
- nseg=2
- nseg=3

Example: axon nseg = 3

To evaluate spatial resolution

forall nseg = nseg*3

and repeat the simulation
nseg = ?

The exact value is always an empirical issue. It should be

- odd
- as small as possible, but not too small

How to decide:

- make first guess
- run a simulation
- REPEAT
  - forall nseg *= 3
  - check by simulation
- UNTIL no significant change in results

Question: how to make the first guess?
The \texttt{d\_lambda} rule

Membrane current is almost entirely capacitive when $f \geq 5 / (2 \pi \tau_m)$

For $\tau_m > 10 \text{ ms}$, this happens at frequencies $> \sim 80 \text{ Hz}$

for each section
  calculate $\lambda_{100}$
    make \texttt{nseg} an odd number just big enough
    that $L/\texttt{nseg} \leq \texttt{d\_lambda} \cdot \lambda_{100}$

$\texttt{d\_lambda} = 0.1$ usually works well
Applying the d_lambda rule

With model specifications created with the CellBuilder (or by hoc code exported from CB):
   specify "d_lambda rule" for the "all" subset
    (on Geometry Strategy page)

With model specifications in user-written hoc code:
load_file("nrngui.hoc") // or load_file("stdgui.hoc")
   // defines lambda_f()

        . . . model specification code . . .

    d_lambda = 0.1
    forsec all {
       nseg = 1 + 2*int((0.999 + L/(d_lambda*lambda_f(100)))/2)
2. Some essential idioms

- Iterating over sections
- Iterating over segments: when to "for (x)" and when to "for (x,0)"
- Special examples: testing for bottlenecks
  - stylized models
  - pt3d geometry
Iterating over sections

```python
oc>tally = 0
oc>forall tally += 1
oc>print tally  // total # sections
```

Generalizations:
To get total # segments, substitute nseg for 1.
```
// sections with names that match ".*string.*"
forsec string statement

// sections that were appended to sectionlist
forsec string statement
```
Iterating over segments

```
oc>create axon
oc>access axon
oc>nseg = 3

oc>for (x) print x
0
0.16666667
0.5
0.83333333
1

oc>for (x) diam(x)=x
oc>for (x) print x, diam(x)
0 0.16666667
0.16666667 0.16666667
0.5 0.5
0.83333333 1
1 1
```
Iterating over segments \textit{cont.}

\begin{verbatim}
  oc>create axon
  oc>access axon
  oc>nseg = 3

  oc>for (x,0) print x
  0.16666667
  0.5
  0.83333333

  oc>for (x,0) diam(x)=x
  oc>for (x,0) print x, diam(x)
  0.16666667 0.16666667
  0.5 0.5
  0.83333333 0.83333333
\end{verbatim}

\textbf{Summary:}

for \((x,0)\) \textit{statement} \textit{\// to assign values to range vars}
for \((x)\) \textit{statement} \textit{\// ok for "browsing" range var values}
\textit{\// but not for assigning values to range vars}
Testing for bottlenecks

Stylized \((L, \text{diam})\) models

\[
\text{forall } \text{for } (x) \text{ if } (\text{diam}(x) < 1) \text{ \textbackslash }
\text{print secname()}, " ", x, \text{diam}(x)
\]
Stylized (L, diam) models

forall for (x) if (diam(x) < 1) \
    print secname(), " ", x, diam(x)

3-D models

forall for i=0,n3d()-1 \
    if (diam3d(i) < 1) \
    print secname(), " ", i, diam3d(i)
3. How to discover section names

... and other things of interest

Tools / Distributed Mechanisms / Viewers / Shape Name
Note scrollable list of section names.
Click on a section in shape plot →
        turns section red, highlights name.
Click on a section name → turns section red.
Double click on a section name →
displays section parameters.
Use "Type" button to specify other action
e.g. show assigned variables, states, or all.
4. Combine hoc and the GUI

CellBuilder
    manage anatomically detailed model cells

Network Builder
    specify cell classes,
      create basic code for network management

LinearCircuit Builder
    electronic instrumentation

Channel Builder
    fast HH-style or kinetic scheme models of voltage
      and/or ligand-gated channels, without writing NMODL code

ModelViewer
    discover what's really in a model

VariableTimeStep control's AToI Scale Tool
Mine reusable code from GUI-generated ses and hoc files
Mining code from a session file

Example: given a GUI-created graph, mine its session file for reusable code.
The session file

objectvar save_window_, rvp_
objectvar scene_vector_[5]
objectvar ocbox_, ocbox_list_, scene_, scene_list_
{ocbox_list_ = new List() scene_list_ = new List()}
{
    save_window_ = new Graph(0)
save_window_.size(0, 5, -80, 40)
scene_vector_[4] = save_window_
{save_window_.view(0, -80, 5, 120, 697, 31, 303.36, 203.2)}
graphList[0].append(save_window_)
save_window_.save_name("graphList[0].")
save_window_.addvar("soma.v(.5)", 2, 1, 2.55696, 41.6063, 1)
save_window_.addvar("dendrite_1[5].v( 0.5 )", 1, 1, 0.512025, \
    0.914173, 2)
}
objectvar scene_vector_[1]
{doNotify()}
Mining code from a session file cont.

Identify the useful stuff

```python
objectvar save_window_, rvp_
objectvar scene_vector_[5]
objectvar ocbox_, ocbox_list_, scene_, scene_list_
{ocbox_list_ = new List() scene_list_ = new List()}
{
    save_window_ = new Graph(0)
save_window_.size(0,5,-80,40)
scene_vector_[4] = save_window_
{save_window_.view(0, -80, 5, 120, 697, 31, 303.36, 203.2)}
graphList[0].append(save_window_)
save_window_.save_name("graphList[0].")
save_window_.addvar("soma.v(.5)", 2, 1, 2.55696, 41.6063, 1)
save_window_.addvar("dendrite_1[5].v( 0.5 )", 1, 1, 0.512025, \
                      0.914173, 2)
}
objectvar scene_vector_[1]
{doNotify()}
```
Mining code from a session file \textit{cont.}

Reuse with own graph

```java
objref g
g = new Graph(0)
g.size(0,5,-80,40)
g.view(0, -80, 5, 120, 697, 31, 303.36, 203.2)
graphList[0].append(g)
g.addvar("soma.v(.5)", 2, 1, 2.55696, 41.6063, 1)
g.addvar("dendrite_1[5].v( 0.5 )", 1, 1, 0.512025, 0.914173, 2)
```
5. Discovering NEURON's hoc library

The heart of the GUI and standard run system

UNIX   /usr/local/src/nrn/share/lib/hoc/
MSWin c:\nrn\lib\hoc\n
load_file("stdgui.hoc") gets what you need
without bringing up NEURONMainMenu

stdlib.hoc  commonly used procs and funcs
stdrun.hoc  simulation initialization and execution code
Snooping in stdlib.hoc

Right at the top:

String class
iterator case()
func lambda_f()
Using the String class

```java
objref sobj[5]
for i=0,4 sobj[i] = new String()
for i=0,4 sprint(sobj[i].s, \\
    "Number %d", i)

for i=0,4 print sobj[i].s
```
Using iterator case()

x=0
for case (&x, 1,2,4,7,-25) {
    print x
}

for case (&IClamp[0].amp, -0.1, -0.07, 0.07, 0.1) run()
6. Customizing simulation execution

Standard run system code is in stdrun.hoc

How to:

• launch batch runs
• automate preprocessing or postprocessing
• attach graphs to the standard run system
• attach objects that need updating at every fadvance()
• force code execution before/after each time step
Batch runs / automated preprocessing or postprocessing

proc batchrun() { local i
  for i = 0,$1 {
    perturb some parameter(s)
    run()
    do something with results
  }
}
Attach a graph to the standard run system by appending it to a graphlist.

x coordinates are

graphlist[0]  t
graphlist[1]  t-0.5*dt
graphlist[2]  t+0.5*dt
graphlist[3]  arbitrary function of t

If an object needs to be updated at every fadvance(), append it to graphlist[0].
To force code execution before/after each time step, use a custom proc `advance()`

```
// load after standard library
// so it supercedes the built-in advance()
proc advance() {
    ...precalc code...
    fadvance()
    ...postcalc code...
}
```

If pre- or postcalc changes a parameter or state, it must also call `cvode.re_init()`.
7. Customizing initialization

Minimum prerequisite: absent any change of model structure or parameters, each run produces the same result, regardless of what was done before.

**Typical custom initializations**
- to steady state (of a system that has a resting state, i.e. lacks spontaneous endogenous activity and external perturbations)
- to a defined starting point on a trajectory of an oscillating or chaotic system
- to a state that satisfies some criterion

**How?** A custom init() procedure (load after standard library).
stdrun.hoc's built-in proc init():

proc init() {
    finitialize(v_init)
    // Extra initialization should normally go here.
    // If you change any states or parameters after
    // an finitialize, then you should complete
    // the initialization with
    /*
    if (cvode.active()) {
        cvode.re_init()
    } else {
        fcurrent()
    }
    */
    frecord_init()
    */
}
Initializing to steady state

"Jump into the past," advance with implicit Euler, then return to the present.

```plaintext
proc init() {
  local dtsav, temp
  finitialize(v_init)
  t = -1e10
  dtsav = dt
  dt = 1e9 // can be very large if model allows
  // if cvode is on, turn it off to do large fixed step
  temp = cvode.active()
  if (temp!=0) { cvode.active(0) }
  while (t<-1e9) {
    fadvance()
  }
  // restore cvode if necessary
  if (temp!=0) { cvode.active(1) }
  dt = dtsav
  t = 0
  if (cvode.active()) {
    cvode.re_init()
  } else {
    fcurrent()
  }
  frecord_init()
}
```
Initializing to a desired state
Especially useful for oscillating or chaotic models.

Run a "warmup simulation," then save all states.

```java
objref svstate, f
svstate = new SaveState()
svstate.save()
```

If desired, write state info to a file for future use

```java
f = new File("states.dat")
svstate.fwrite(f)
```

To read state info from a file

```java
objref svstate, f
svstate = new SaveState()
f = new File("states.dat")
svstate.fread(f)
```

Then use a custom `init()` to restore the saved states.
Initializing to a desired state *continued*

A custom init() to restore the saved states:

```plaintext
proc init() {
   finitialize(v_init)
   svstate.restore()
   t = 0 // t is one of the "states"
   if (cvode.active()) {
      cvode.re_init()
   } else {
      fcurrent()
   }
   frecord_init()
}
```
Initializing to a particular resting potential

One approach: adjust the leak equilibrium potential so that leak current balances the other ionic currents when the cell is at the desired resting potential.

Example: for a single compartment model with hh,

```plaintext
proc init() {
  finitialize(v_init)
  el_hh = (ina + ik + gl_hh*v)/gl_hh
  if (cvode.active()) {
    cvode.re_init()
  } else {
    fcurrent()
  }
  frecord_init()
}
```
Initializing to a particular resting potential continued

Alternative strategy: add a mechanism that injects a constant current to balance the other currents.

Example:

```plaintext
NEURON {
    SUFFIX constant
    NONSPECIFIC_CURRENT i
    RANGE i, ic
}
UNITS {
    (mA) = (milliamp)
}
PARAMETER {
    ic = 0 (mA/cm2)
}
ASSIGNED {
    i (mA/cm2)
}
BREAKPOINT {
    i = ic
}
```

This needs a different custom `init()`
Initializing to a particular resting potential continued

Custom init() to use with the constant current mechanism:

```plaintext
proc init() {
    finitialize(-65)
    ic_constant = -(ina + ik + il_hh)
    if (cvode.active()) {
        cvode.re_init()
    } else {
        fcurrent()
    }
    frecord_init()
}
```
Other sources of information

The **NEURON hacks** and **Hot tips** pages
at the NEURON Forum
http://www.neuron.yale.edu/phpBB/

The Programmer's Reference, FAQ list,
and tutorial links at NEURON's Documentation page
http://www.neuron.yale.edu/neuron/docs

And don't forget NEURON's hoc library . . .