

Table of Contents and Schedule of Presentations

NTC	Ted Carnevale
MLH	Michael Hines
PJ	Peter Jonas
GT	Gašper Tkačik

Hands-on exercises are indicated by an asterisk * in the Page column.
Times shown are approximate, except for lunch.

Monday, 8/20 Morning session

Time	Speaker	Title	Page
8:30 AM	MLH	Welcome to the NEURON summer course	
8:35 AM	PJ	Fast-spiking, parvalbumin-expressing interneurons: from subcellular experimental analysis to modeling	
9:15	NTC	Introduction to modeling with NEURON	7
9:30	NTC	Example: building and using a simple model with the GUI	11 *
10:45	Coffee Break		
11:00	NTC	Fundamental concepts: neurites, cables, and sections	13
11:15	MLH	Interactive modeling: Hodgkin-Huxley axon	15 *
12:15	End of morning session		
12:30	Lunch		

Afternoon session

1:30 PM	NTC	Fundamental concepts: range, range variables, nodes, and nseg	17
1:45	NTC	Example: constructing branched model cells with the CellBuilder	21 *
3:15	Coffee Break		
3:30	NTC	Working with morphometric data	23 *
5:00	End of afternoon session		

Tuesday, 8/21 Morning session

Time	Speaker	Title	Page
9:00 AM	Q & A		
9:15	NTC	Inhomogeneous channel distributions	27 *
10:30	Coffee Break		
10:45	MLH	The Channel Builder	29
12:15	End of morning session		
12:30	Lunch		

Afternoon session

1:30 PM	MLH	Numerical methods: accuracy, stability, speed	37
2:30	MLH	NMODL: the NEURON Model Description Language	45 *
3:30	Coffee Break		
3:45	NTC	The hoc programming language	51
5:00	End of afternoon session		

Wednesday, 8/22 Morning session

Time	Speaker	Title	Page
8:30 AM	Q & A		
8:45	GT	Inferring the functional connectivity in networks of neurons from simultaneously recorded spiking activity	
9:30	NTC	ModelDB and Model View	79 *
10:45	Coffee Break		
11:00	MLH	Ion accumulation mechanisms: a calcium pump	85 *
12:15	End of morning session		
12:30	Lunch		

Afternoon session

1:30 PM	MLH	NEURON + threads	89 *
3:15	Coffee Break		
3:30	MLH	Families of simulations in parallel	101 *
5:00	End of afternoon session		

Thursday, 8/23 Morning session

Time	Speaker	Title	Page
9:00 AM	Q & A		
9:15	MLH	Python + NEURON	107
10:30	Coffee Break		
10:45	NTC	Networks: synapses, events, and artificial spiking cells	115 *
12:15	End of morning session		

12:30 Lunch

Afternoon session

1:30 PM	MLH	Networks: inhibitory synchronizing network	127 *
3:00	Coffee Break		
3:15	MLH	Variable time steps and parameter discontinuities	133 *
5:00	End of afternoon session		

Friday, 8/24 Morning session

Time	Speaker	Title	Page
9:00 AM	Q & A		
9:15	NTC	Initialization	157 *
10:30	Coffee Break		
10:45	MLH	Parallel computation: distributed network models	165
12:15	End of morning session		
12:30	Lunch		

Afternoon session

1:30 PM	NTC	NEURON's tools for analyzing electrotonus	183 *
3:00	Coffee Break		
3:15	NTC	The Linear Circuit Builder	191 *
4:15	Review discussion		
4:45	Evaluation form (see last page in this booklet)		
5:00	End of afternoon session		

Appendix	Translating network models to parallel hardware in NEURON (Hines & Carnevale 2008). Parallel tutorial from 2010 NEURON Users' Meeting. NEURON and Python (Hines et al. 2009).	before survey
Survey		last page

The What and the Why of Neural Modeling

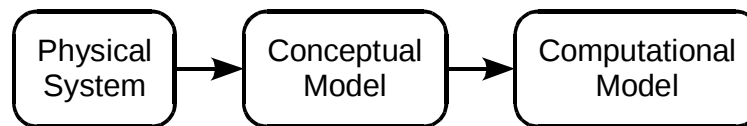
The moment-to-moment processing of information in the nervous system involves the propagation and interaction of electrical and chemical signals that are distributed in space and time.

Empirically-based modeling is needed to test hypotheses about the mechanisms that govern these signals and how nervous system function emerges from the operation of these mechanisms.

Topics

1. How to create and use models of neurons and networks of neurons
 - How to specify anatomical and biophysical properties
 - How to control, display, and analyze models and simulation results
2. How NEURON works
3. How to add user-defined biophysical mechanisms

From Physical System to Computational Model



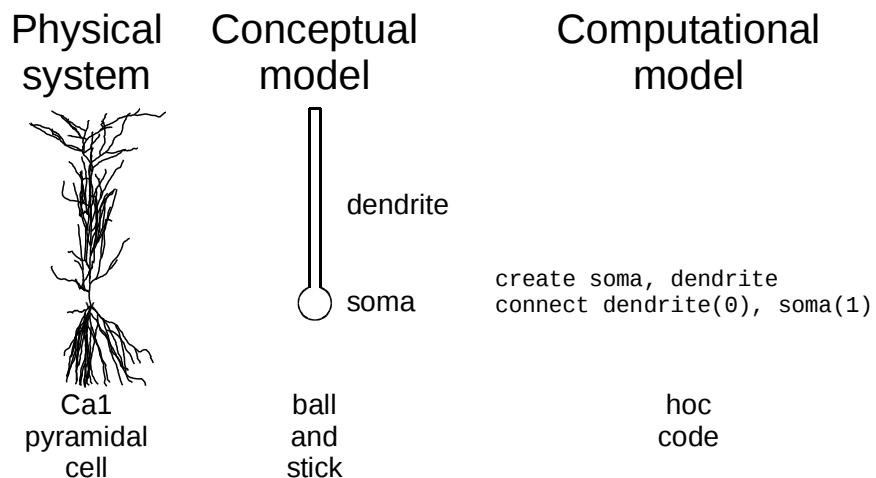
Conceptual model

a simplified representation of the physical system

Computational model

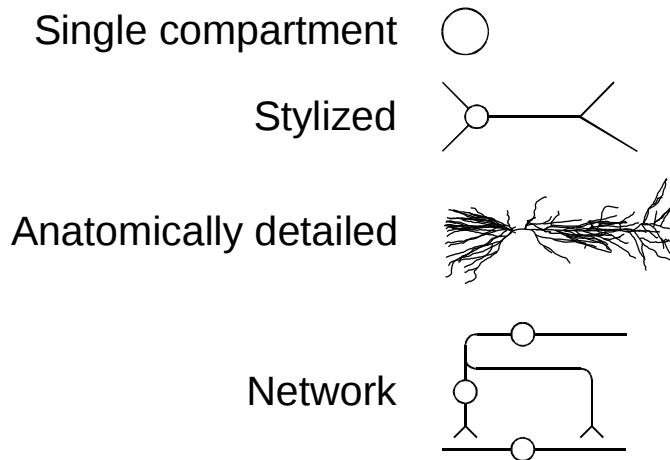
an accurate representation of the conceptual model

From Physical System to Computational Model



Hierarchies of Complexity

Structure



Hierarchies of Complexity

Mechanism

Passive and Active currents

HH-style
kinetic scheme

Synaptic transmission

continuous
spike-triggered

Gap junctions

Extracellular fields, Linear circuits

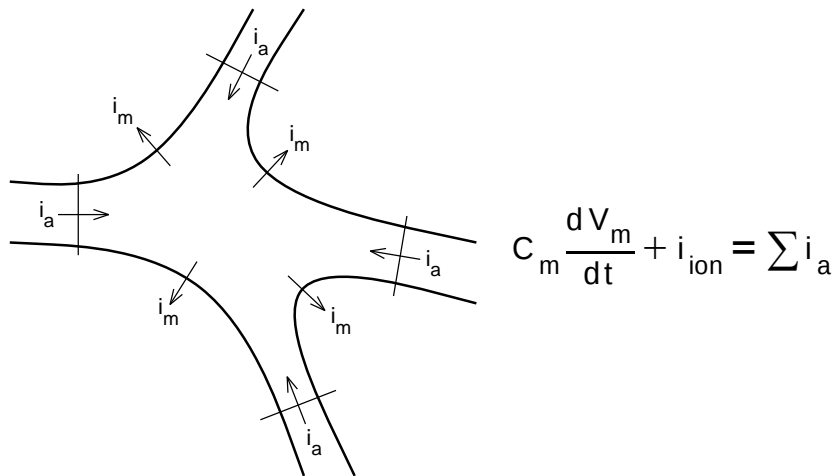
Diffusion, buffers, transport & exchange

Artificial spiking cells ("integrate & fire")

Fundamental Concepts in NEURON

Signals	What moves	Driving force	What is conserved
Electrical	charge carriers	voltage gradient	charge
Chemical	solute	concentration gradient	mass

Conservation of Charge



Example: Single Compartment

Lipid bilayer (no channels)

Membrane with linear ion channels (passive leak)

Project goals:

- Use the GUI to build the model and custom interface for using it
- Run simulations and analyze results
- Change stimulus intensity and duration
- Adjust graphical displays of simulation results
- Adjust dt and Points Plotted / ms

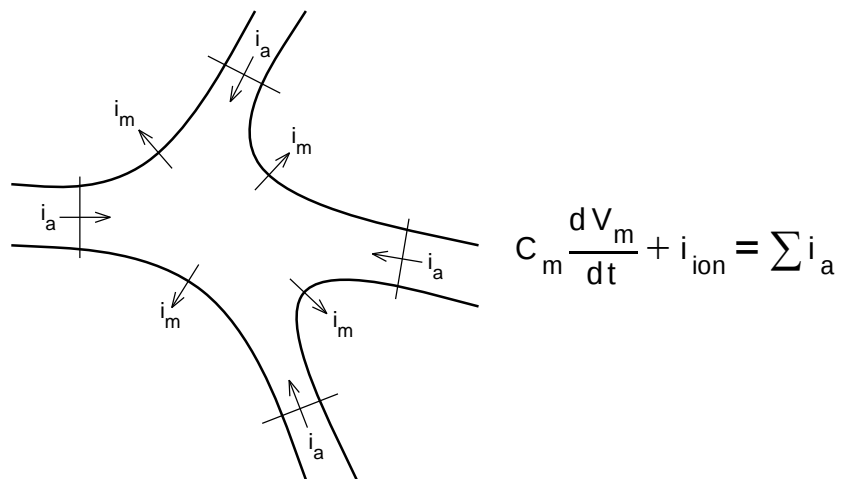
Fundamental Concepts in NEURON

What equations are being solved?

How to separate the biology
from computational details?

It's all about conceptual control . . .

Conservation of Charge



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

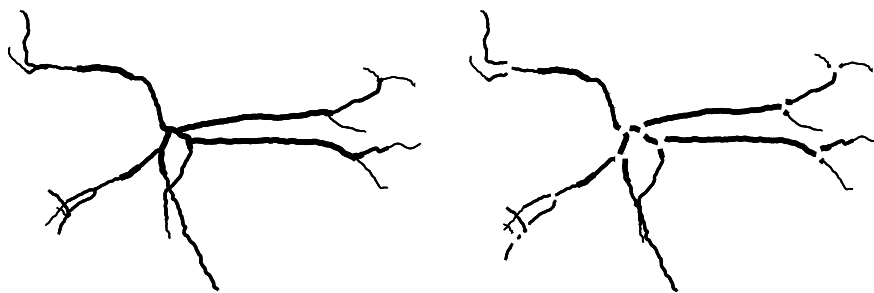
c_j membrane capacitance of compartment j

r_{jk} axial resistance between the centers of
compartment j
and
adjacent compartment k

Separating Anatomy and Biophysics from Purely Numerical Issues

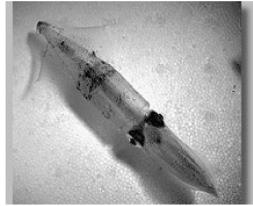
section

a continuous length of unbranched cable



Anatomical data from A.I. Gulyás

Physical System



From <http://www.mbl.edu>

Model

Hodgkin-Huxley cable equations

$$\frac{1}{4D} \frac{\partial}{\partial x} \left(\frac{D^2}{R_a} \frac{\partial V}{\partial x} \right) = C_m \frac{\partial V}{\partial t} + \bar{g} m^3 h \cdot (V - E_{na}) + \bar{g}_k n^4 \cdot (V - E_k) + g_l \cdot (V - E_l)$$

$$\begin{aligned} \frac{dm}{dt} &= -\alpha_m m + \beta_m (1-m) & \alpha_m &= \frac{0.1(V+40)}{1-e^{-0.1(V+40)}} & \beta_m &= 4e^{-(V+65)/18} \\ \frac{dh}{dt} &= -\alpha_h h + \beta_h (1-h) & \alpha_h &= 0.07e^{-0.05(V+65)} & \beta_h &= \frac{1}{1+e^{-0.1(V+35)}} \\ \frac{dn}{dt} &= -\alpha_n n + \beta_n (1-n) & \alpha_n &= \frac{0.01(V+55)}{1-e^{-0.1(V+55)}} & \beta_n &= 0.125e^{-(V+65)/80} \end{aligned}$$

Model

Hodgkin-Huxley cable equations

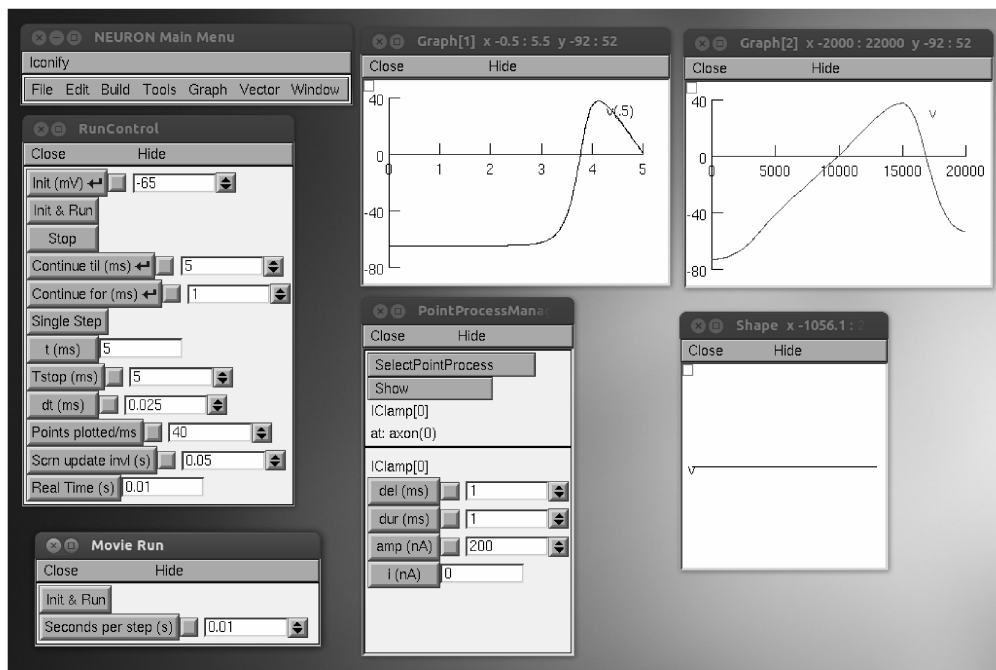
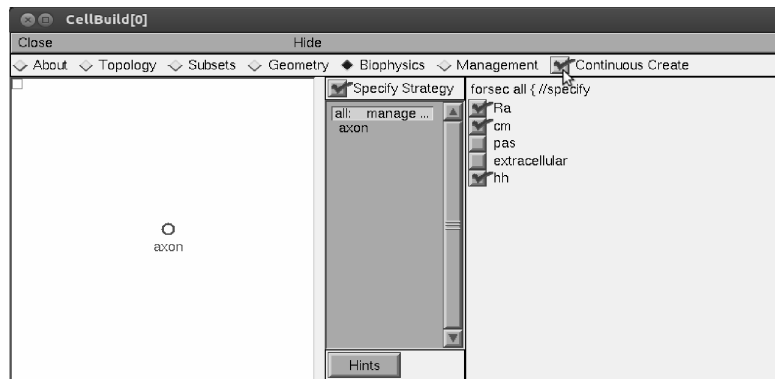
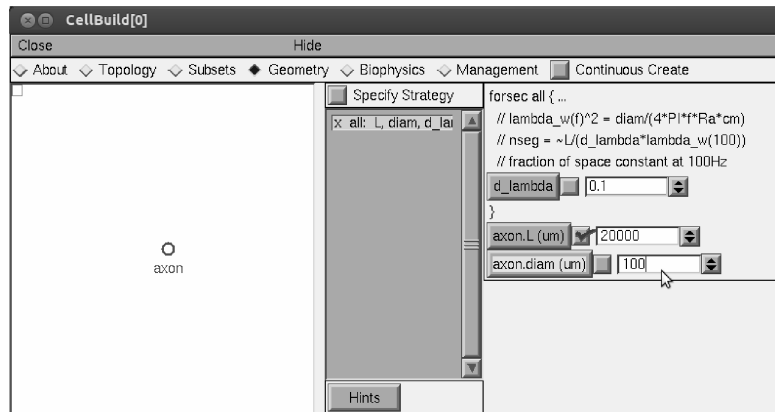
$$\frac{1}{4D} \frac{\partial}{\partial x} \left(\frac{D^2}{R_a} \frac{\partial V}{\partial x} \right) = C_m \frac{\partial V}{\partial t} + \bar{g} m^3 h \cdot (V - E_{na}) + \bar{g}_k n^4 \cdot (V - E_k) + g_l \cdot (V - E_l)$$

$$\begin{aligned} \frac{dm}{dt} &= -\alpha_m m + \beta_m (1-m) & \alpha_m &= \frac{0.1(V+40)}{1-e^{-0.1(V+40)}} & \beta_m &= 4e^{-(V+65)/18} \\ \frac{dh}{dt} &= -\alpha_h h + \beta_h (1-h) & \alpha_h &= 0.07e^{-0.05(V+65)} & \beta_h &= \frac{1}{1+e^{-0.1(V+35)}} \\ \frac{dn}{dt} &= -\alpha_n n + \beta_n (1-n) & \alpha_n &= \frac{0.01(V+55)}{1-e^{-0.1(V+55)}} & \beta_n &= 0.125e^{-(V+65)/80} \end{aligned}$$

Simulation

Representation

```
create axon
axon {
    nseg = 43
    diam = 100
    L = 20000
    insert hh
}
```



Syntax: create sectionname

Example: create soma, dend[3]
 creates one section called soma
 and three sections called dend[0], dend[1], and dend[2]

Assigning anatomical and biophysical attributes:

```
soma {
  L = 50      // [um] length
  diam = 50   // [um] diameter
  insert hh   // Hodgkin-Huxley mechanism
}
for i=0,2 dend[i] {
  L = 200
  diam = 2
  insert pas  // passive channels
}
```

Range Variables

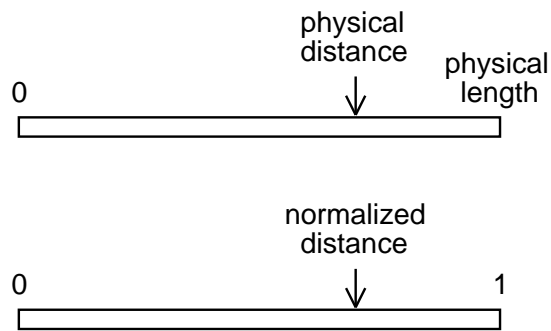
Name	Meaning	Units
diam	diameter	[μm]
cm	specific membrane capacitance	[$\mu\text{f}/\text{cm}^2$]
g_pas	specific conductance of the pas mechanism	[siemens/ cm^2]
v	membrane potential	[mV]

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



Syntax:

```
sectionname.rangevar(range)
```

returns or sets the value of rangevar
at the location corresponding to range

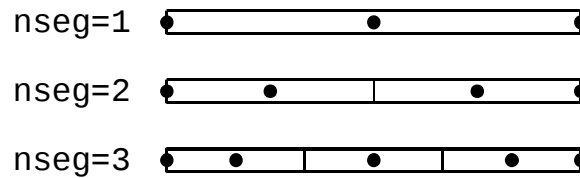
Examples:

```
dend.v(0.5)
```

returns membrane potential at the middle of dend
Shortcut: `dend.v`
`dend for (x) print x*L, v(x)`
prints physical distance and v
at each point in dend where v was calculated

nseg

the number of points in a section where
membrane current and potential are computed



Example: axon nseg = 3

To test spatial resolution
forall nseg = nseg*3
and repeat the simulation

Category	Variable	Units
Time	t	[ms]
Voltage	v	[mV]
Current		
specific	i	[mA/cm ²] (distributed)
absolute		[nA] (point process)
Capacitance		
specific	cm	[μf/cm ²]
absolute		[nf] (point process)
Length	diam, L	[μm]
Conductance		
specific	g	[S/cm ²] (distributed)
absolute		[μS] (point process)
Cytoplasmic resistivity	Ra	[Ω cm]
Resistance	ri()	[10 ⁶ Ω]
Concentration	nai etc.	[mM]

Example: Branched Model Cells

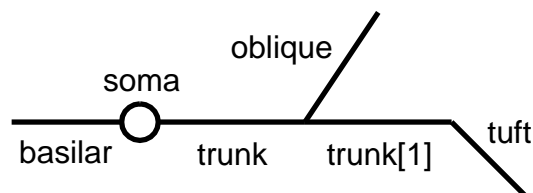
Physical system: anatomically complex cell

Conceptual model: "stick figure"

Computational model: soma + dendritic cylinder(s)
(and maybe an axon . . .)

Project goals:

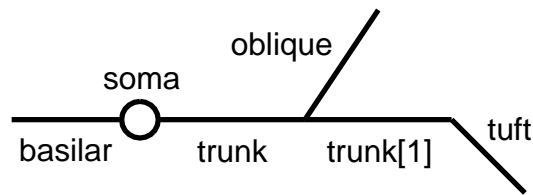
- Learn how to use CellBuilder
- Use session files to save and retrieve user interface (elementary project management)
- Test model and simulation:
structural integrity
discretization of space and time



From hoc file generated by CellBuilder:

```
create soma, trunk[2], oblique, tuft, basilar
```

```
proc topol() { local i
  connect trunk(0), soma(1)
  connect trunk[1](0), trunk(1)
  connect oblique(0), trunk(1)
  connect tuft(0), trunk[1](1)
  connect basilar(0), soma(0)
  . . .
```



From hoc file generated by CellBuilder:

```
proc geom() {
  forsec all { }
  soma { L = 30  diam = 30 }
  trunk { L = 400  diam = 3 }
  trunk[1] { L = 400  diam = 2 }
  oblique { L = 300  diam = 1.5 }
  tuft { L = 300  diam = 1 }
  basilar { L = 300  diam = 3 }
}
```

```
proc biophys() {
  forsec all {
    Ra = 160
    cm = 1
  }
  forsec dendrites {
    insert pas
    g_pas = 3e-05
    e_pas = -70
  }
  forsec apicals {
    insert hh
    gnabar_hh = 0.012
    gkbar_hh = 0.0036
    gl_hh = 0
    el_hh = -54.3
  }
  soma {
    insert hh
    gnabar_hh = 0.12
    gkbar_hh = 0.036
    gl_hh = 0.0003
    el_hh = -54.3
  }
}
```

Anatomy and Empirically-based Models

Quality of data

- histology
- staining, amputation, shrinkage
- diameter
- spines

Data formats

Detailed Morphometric Data

Where to get it?

- DIY
- the kindness of strangers
- ModelDB
- NeuroMorpho.org

How to get it into NEURON?

- standalone conversion programs
- import into CellBuilder
- Import3D tool
- "already in hoc"

Trust but verify . . .

Qualitative tests

Orphan sections and bottlenecks?

insert pas, set Ra and g_pas low

inject large depol current at soma

examine shape plot of v

Z-axis drift and backlash?

examine side view of shape plot

for abrupt jumps

. . . and verify some more

Quantitative tests

Diameter

Too large?

```
_dmin=10
```

```
forall for i=0, n3d()-1 \
```

```
  if (diam3d(i)<_dmin) _dmin=diam3d(i)
```

```
print _dmin
```

Too small?

```
forall for i=0, n3d()-1 \
```

```
  if (diam3d(i)<0.1) \
```

```
    print secname(), " ", i, diam3d(i)
```


When it really matters . . .

Test for systematic errors

Favorite numbers?

 histogram of diameter measurements

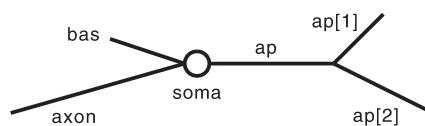
Other tests

Spatially inhomogeneous parameters

Rules

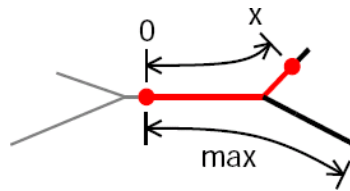
- None (arbitrary values)
- Constant over sets of sections
 use SectionLists (CellBuilder Subsets)
- A function of position
 hoc or ?

Example: model with hh in apical dendrites



Suppose `gnabar_hh` in the apical tree
decreases linearly with distance from the soma.

Details: 100% at tree origin, 0% at most distant termination.



This example:

$$gnabar_hh = 0.12 * (1 - p) \text{ where } p = L_{0x}/L_{max}$$

(normalized path distance from location x
to origin 0 of apical tree)

The general problem: $param = f(p)$, where f can be any function
and p is a "distance metric" such as:

- path length from a reference point
- radial distance from a reference point
- distance from a plane ("3D projection onto a line")

An equivalent hoc idiom:

forsec subset for (x,0) { rangevar_suffix(x) = f(p(x)) }

Conceptualize the task

1. Specify the

subset s

distance metric p

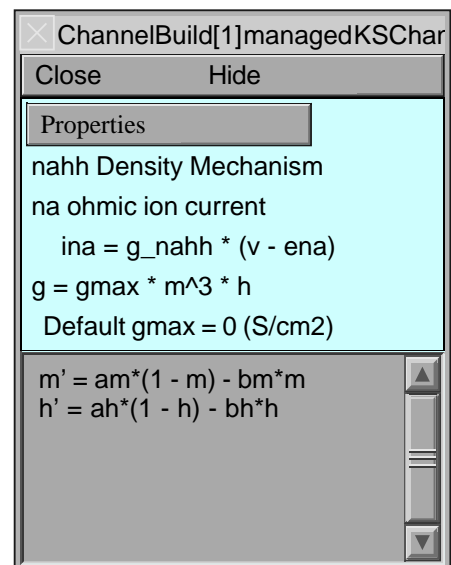
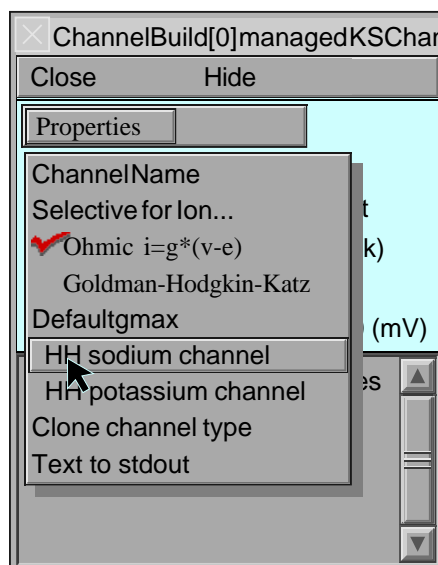
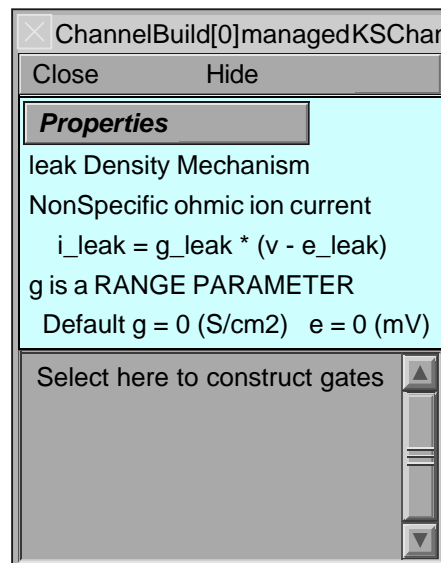
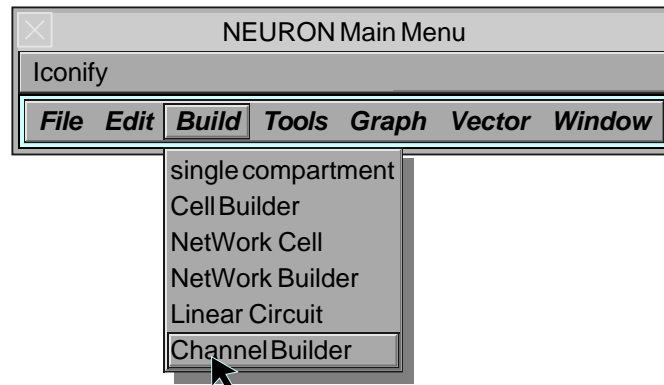
parameter that depends on distance

function f that governs the relationship

between the parameter and p

2. Verify the implementation

How? hoc or GUI (CellBuilder, Model View)



Default gmax = 0 (S/cm2)

$m' = am*(1 - m) - bm*m$
 $h' = ah*(1 - h) - bh*h$

ChannelBuildGateGUI[0]forChannelBuild[0]

Close Hide

States Transitions Properties

Select hh state or ks transition to change properties

m

h

m^3

$m' = am*(1 - m) - bm*m$

Power 3

Fractional Conductance

m fraction 1

Adjust Run

$m \leftrightarrow m(a, b)$ (KSTrans[0])

☒ Display inf, tau

$am = A*x/(1 - \exp(-x))$ where $x = k*(v - d)$

A 1

k 0.1

d -40

$bm = A*\exp(k*(v - d))$

infm

taum

nahh Density Mechanism

na ohmic ion current

$$ina = g_nahh * (v - ena)$$

$$g = gmax * m^3 * h$$

Default gmax = 0 (S/cm2)

$$m' = am*(1 - m) - bm*m$$

$$am = 1*x/(1 - \exp(-x)) \text{ where } x = 0.1*(v + 40) \quad (\text{Vector}[7])$$

$$bm = 4*\exp(-0.05556*(v + 65)) \quad (\text{Vector}[8])$$

$$h' = ah*(1 - h) - bh*h \quad (\text{KSTrans}[1])$$

$$ah = 0.07*\exp(-0.05*(v + 65)) \quad (\text{Vector}[11])$$

$$bh = 1/(1 + \exp(-0.1*(-35 - v))) \quad (\text{Vector}[12])$$

ChannelBuild[0]managedKSChar

Close Hide

Properties

ChannelName

Selective for Ion...

☒ Ohmic $i = g*(v - e)$

Goldman-Hodgkin-Katz

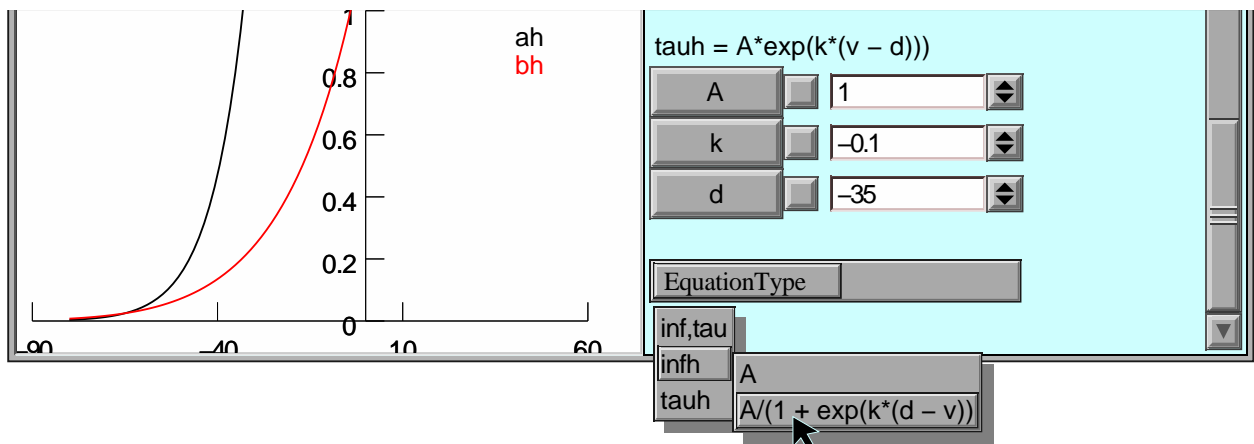
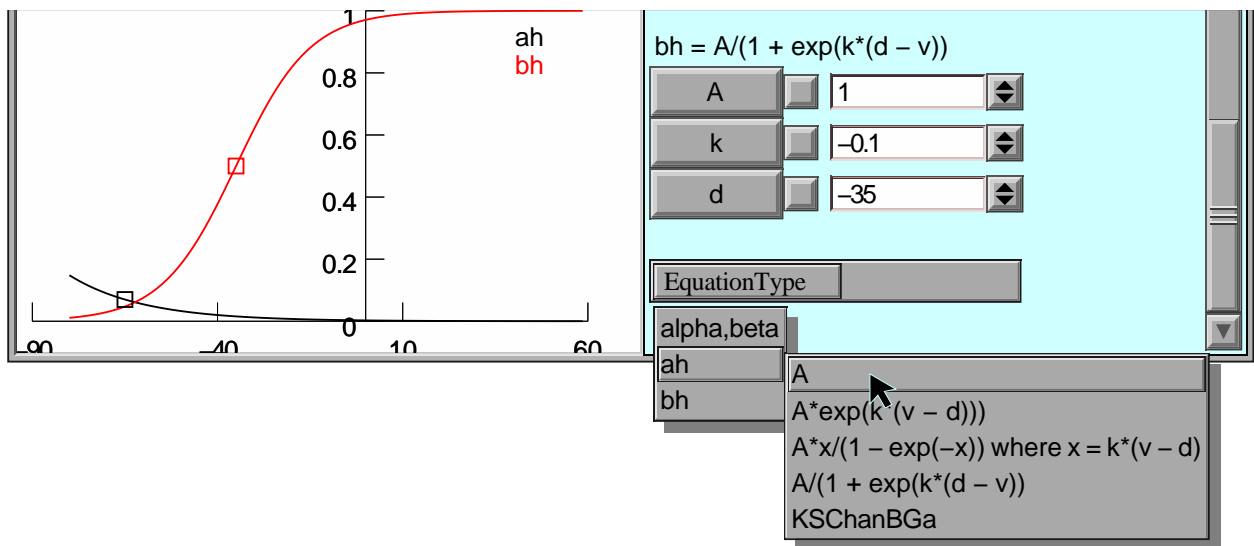
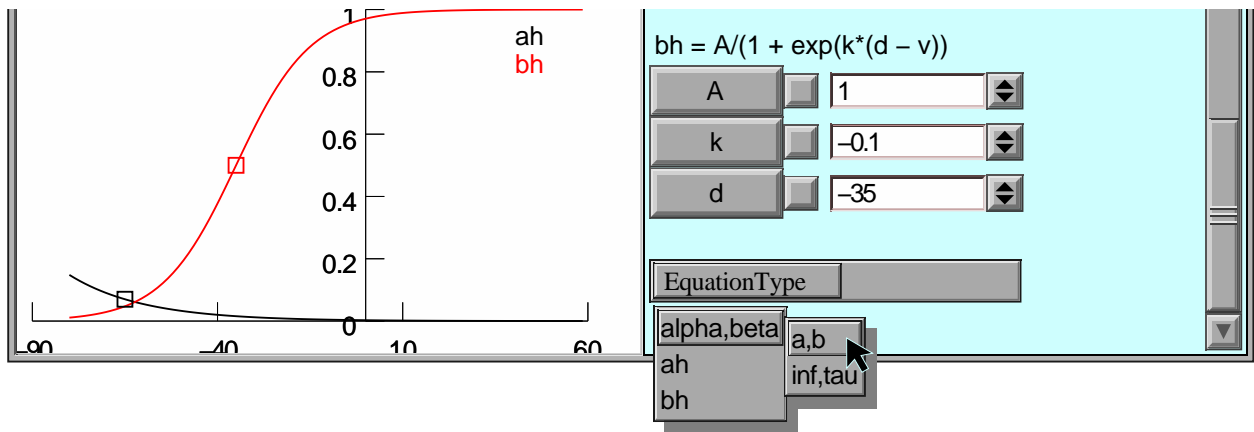
Defaultgmax

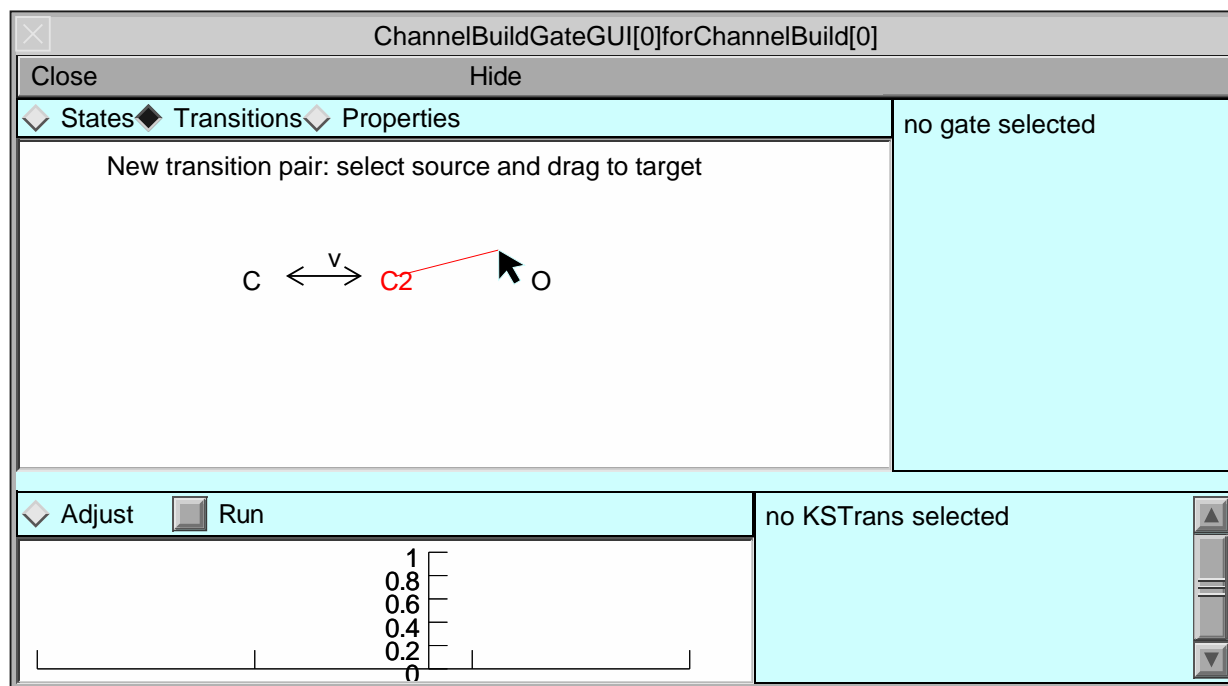
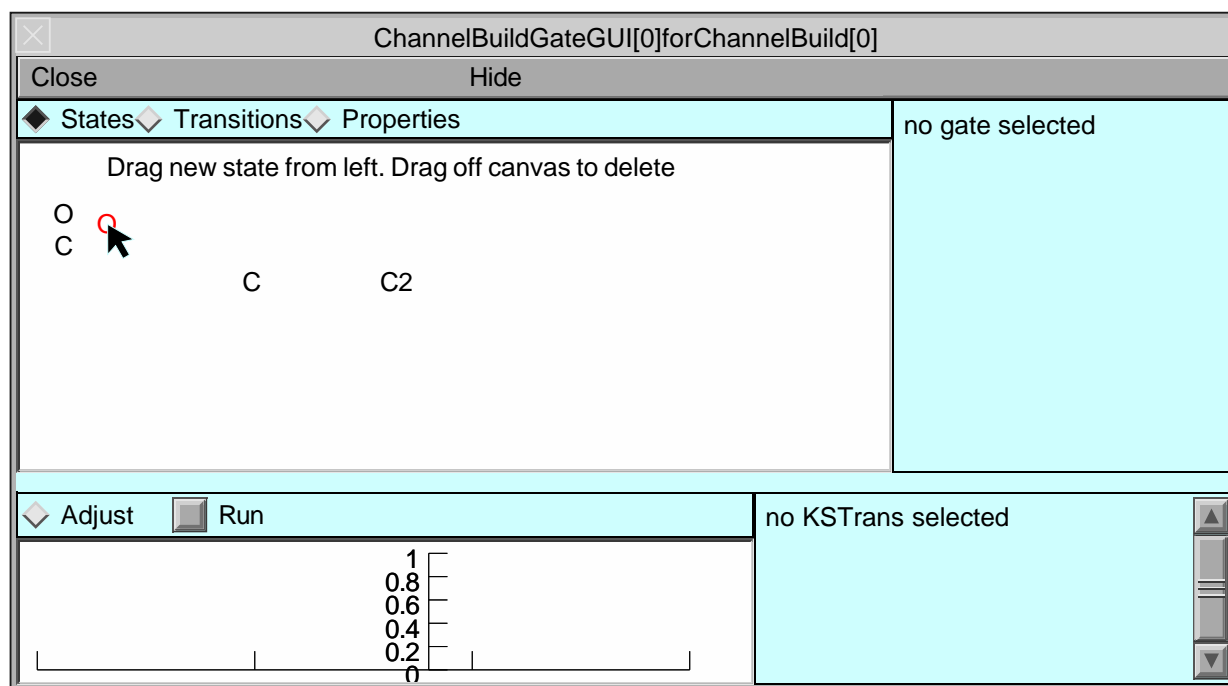
HH sodium channel

HH potassium channel

Clone channel type

Text to stdout





ChannelBuildGateGUI[0]forChannelBuild[0]

Close Hide

States Transitions **Properties**

Select hh state or ks transition to change properties

$C \xrightleftharpoons{v} C2 \xrightleftharpoons{v} O$

O
O: 3 state, 2 transitions

Power 1

Fractional Conductance

C2 fraction 0

O fraction 1

Adjust Run

1
0.8
0.6
0.4
0.2
0

aC2O
bC2O

bC2O = A

A 0

EquationType

alpha,beta
aC2O
bC2O
a,b
inf,tau
nai
nao
ki
ko
cai
cao

Close

Hide

States

Transitions

Properties

Select hh state or ks transition to change properties

$$C \xrightleftharpoons{v} C2 \xrightleftharpoons{cai} O$$

O

O: 3 state, 2 transitions

Power

1

Fractional Conductance

C2 fraction

0

O fraction

1

Adjust

Run

1

0.8

0.6

0.4

0.2

0

aC2O

bC2O

C2 + cai <--> O (a, b) (KSTrans[9])

Display inf, tau

aC2O = A

ChannelBuild[0]managedKSChar

Close

Hide

Properties

kca Density Mechanism

k ohmic ion current

ik = g_kca * (v - ek)

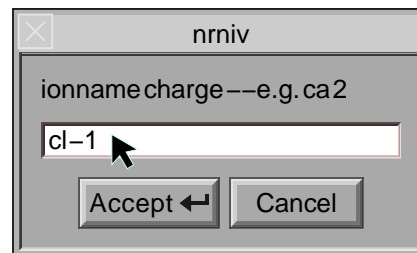
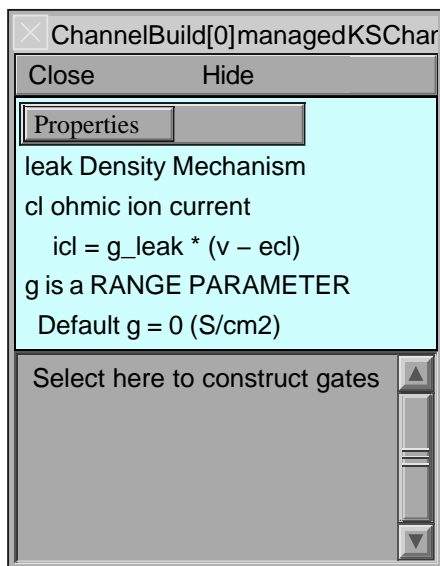
g = gmax * O

Default gmax = 0 (S/cm2)

O: 3 state, 2 transitions

The screenshot shows the NEURON ChannelBuild GUI. The main window is titled "ChannelBuildGateGUI[0]forChannelBuild[0]". It has a "Close" and "Hide" button at the top. Below the title bar, there are tabs for "States", "Transitions", and "Properties". The "States" tab is active, showing a state transition diagram with states O, C, C2, O2, and C3. The transitions are labeled with voltage (V) and arrows. The diagram shows: C \xleftrightarrow{V} C2 \xleftrightarrow{V} O, and C3 \xleftrightarrow{V} O2. The state O3 is highlighted in red. A dialog box titled "nrniv" is open, showing "Change state name" with a text field containing "O3" and "Accept" and "Cancel" buttons. The "Properties" tab shows the following text: "leak Density Mechanism", "NonSpecific ohmic ion current", $i_{leak} = g_{leak} * (v - e_{leak})$, $g = g_{max} * O * O2 * O3$, "Default gmax = 0 (S/cm2) e = 0 (mV)", "O: 3 state, 2 transitions", "O2: 2 state, 1 transitions", and $O3' = aO3*(1 - O3) - bO3*O3$. At the bottom, there are "Adjust" and "Run" buttons, and a "no KSTrans selected" message.

The screenshot shows the NEURON ChannelBuild GUI. The main window is titled "ChannelBuild[0]managedKSChar". It has a "Close" and "Hide" button at the top. Below the title bar, there are tabs for "Properties", "ChannelName", "Selective for Ion...", "Defaultgmax", "HH sodium channel", "HH potassium channel", "Clone channel type", and "Text to stdout". The "Properties" tab is active, showing a list of channel types: "NonSpecific", "na", "k", and "Create new type". The "Create new type" button is highlighted. The "ChannelName" tab shows "ChannelName". The "Selective for Ion..." tab shows a list of ion types: "Ohmic i=g*(v-e)", "Goldman-Hodgkin-Katz", "HH sodium channel", "HH potassium channel", "Clone channel type", and "Text to stdout". The "Ohmic i=g*(v-e)" option is checked with a red checkmark.

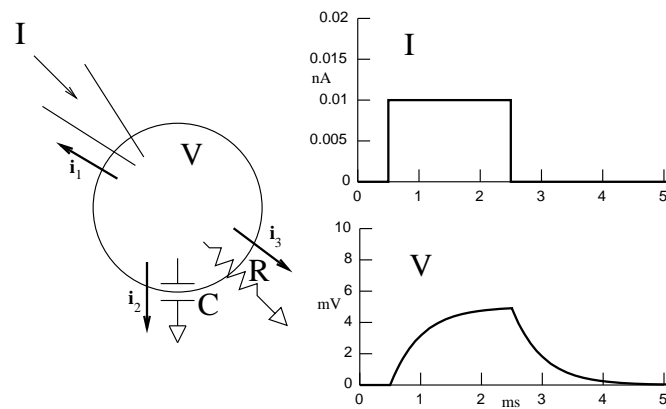


Compartmental Modeling

Not much mathematics required.

Good judgment essential!

1



$$i_1 + i_2 + i_3 = 0$$

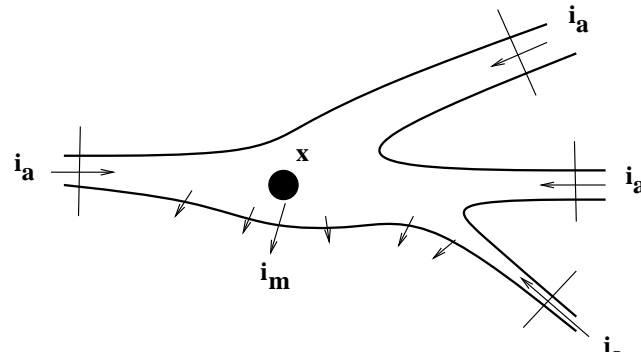
$$i_1 = -I$$

$$i_2 = C \, dV / dt$$

$$i_3 = V / R$$

$$C \, dV / dt + g \, V = I$$

2

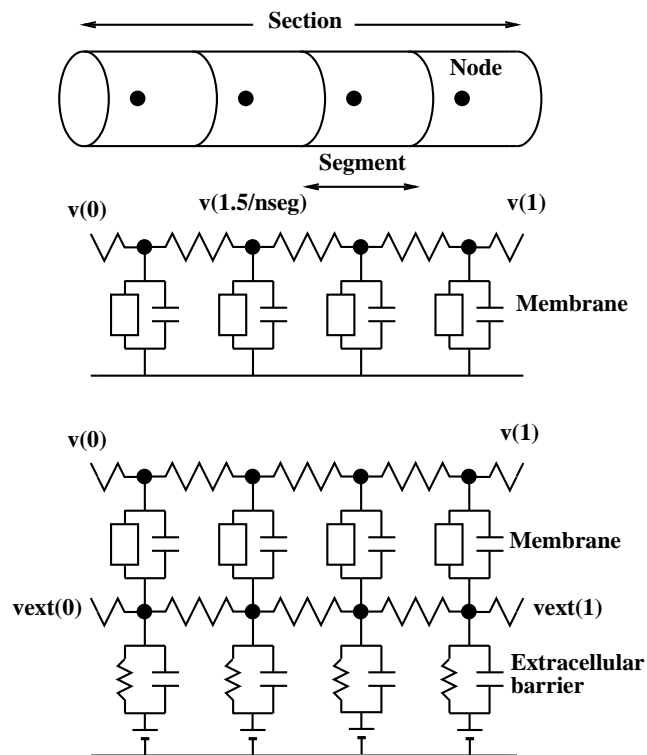


The diagram shows a neuron segment with a central node 'x'. Axial currents i_a flow into and out of the segment at its boundaries. Membrane currents i_m flow out of the segment across its surface. Below the diagram are two equations:

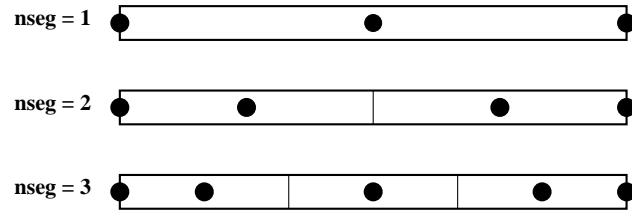
$$\int i_m = \sum i_a$$

$$c_j \frac{dv_j}{dt} + i_j = \sum_k \frac{v_k - v_j}{r_{jk}}$$

3



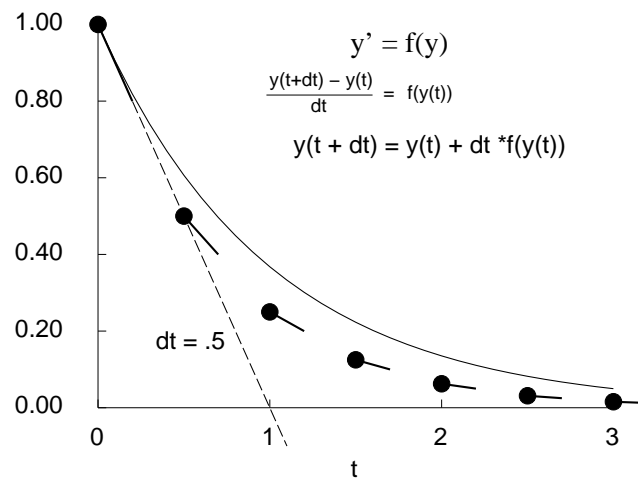
4



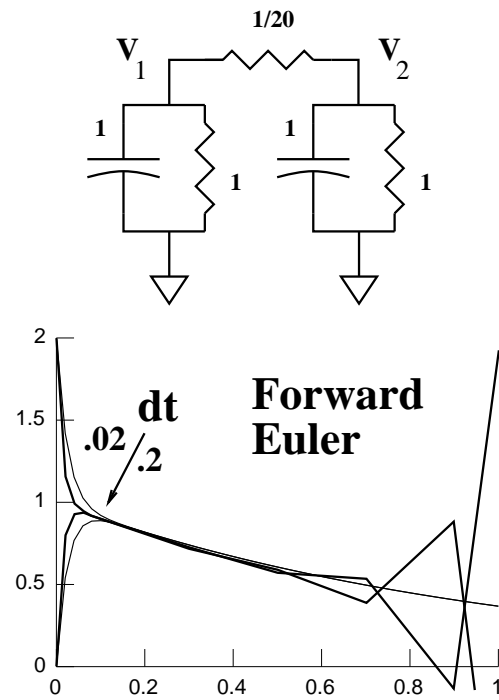
forall nseg *= 3

5

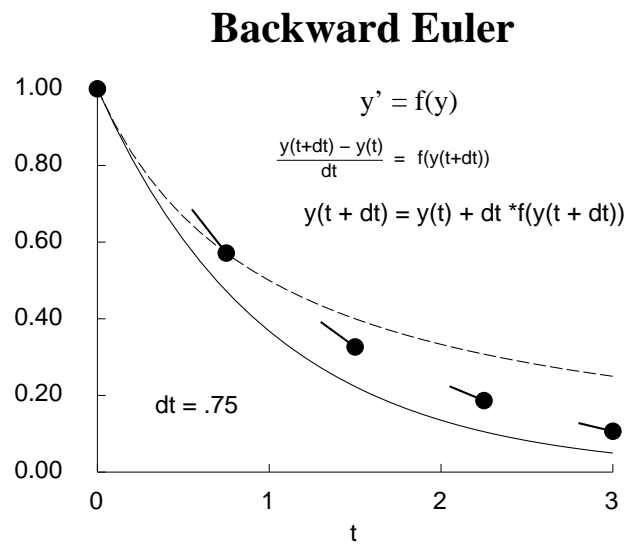
Forward Euler



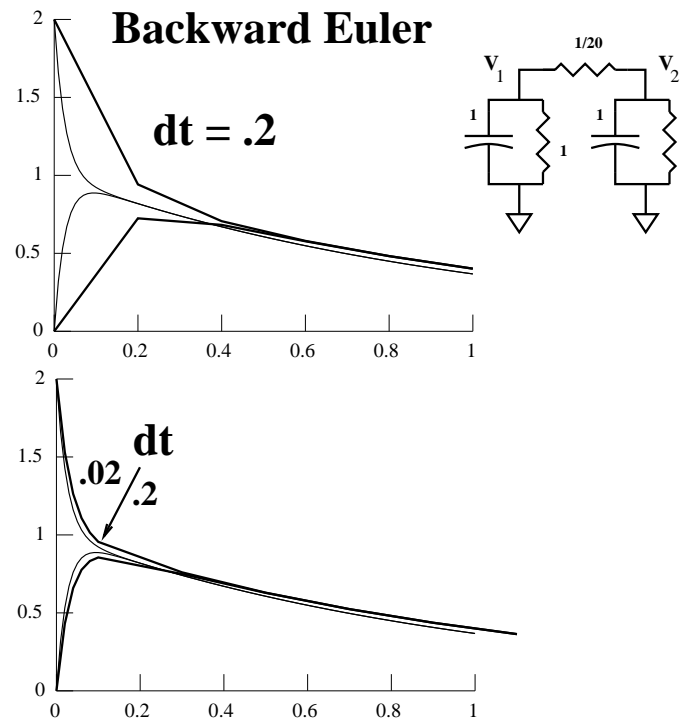
6



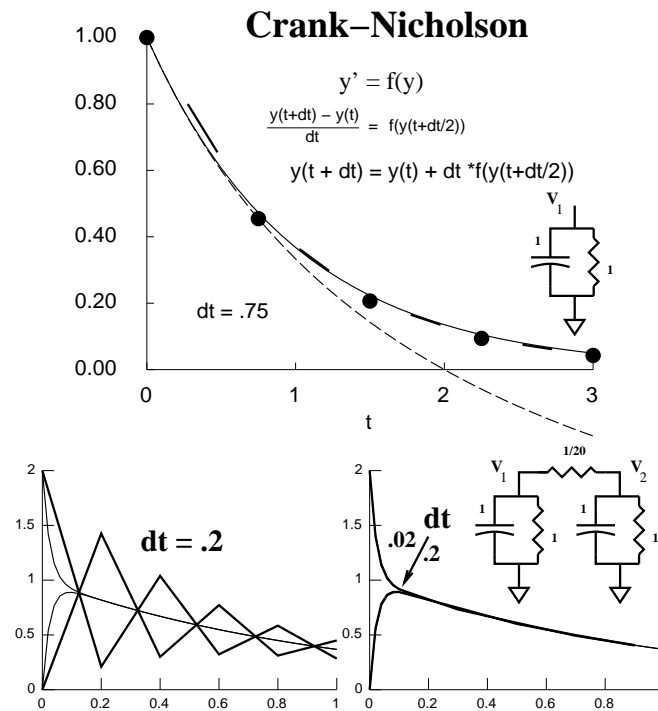
7



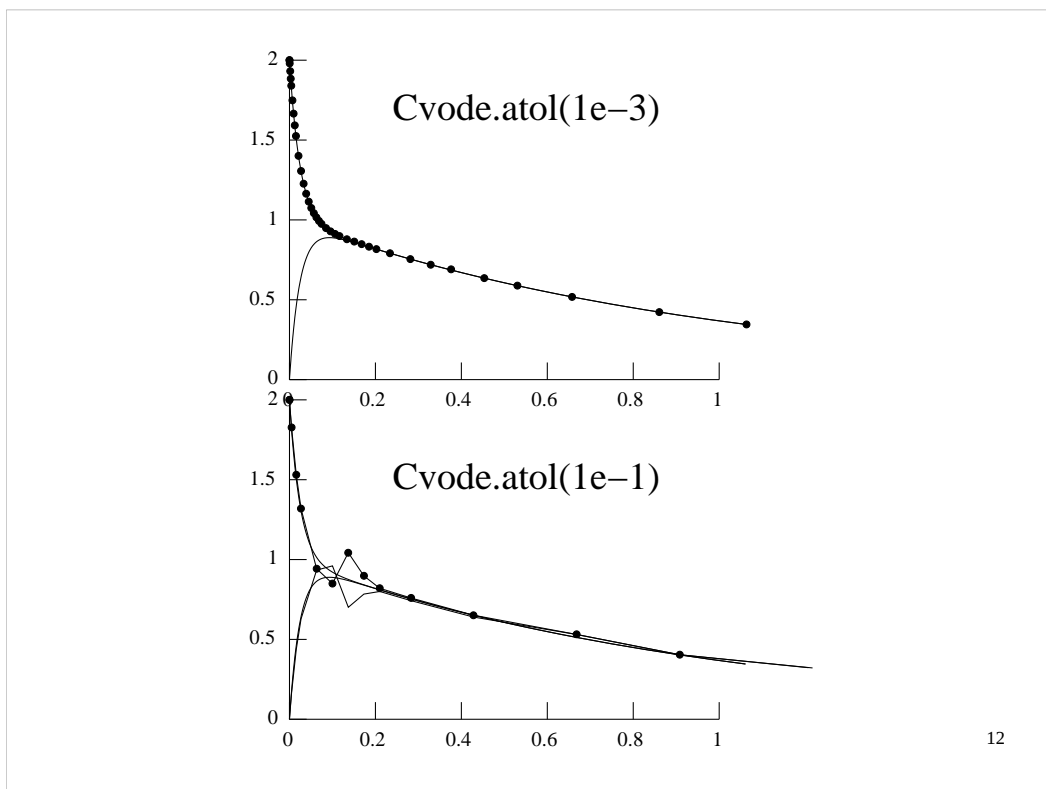
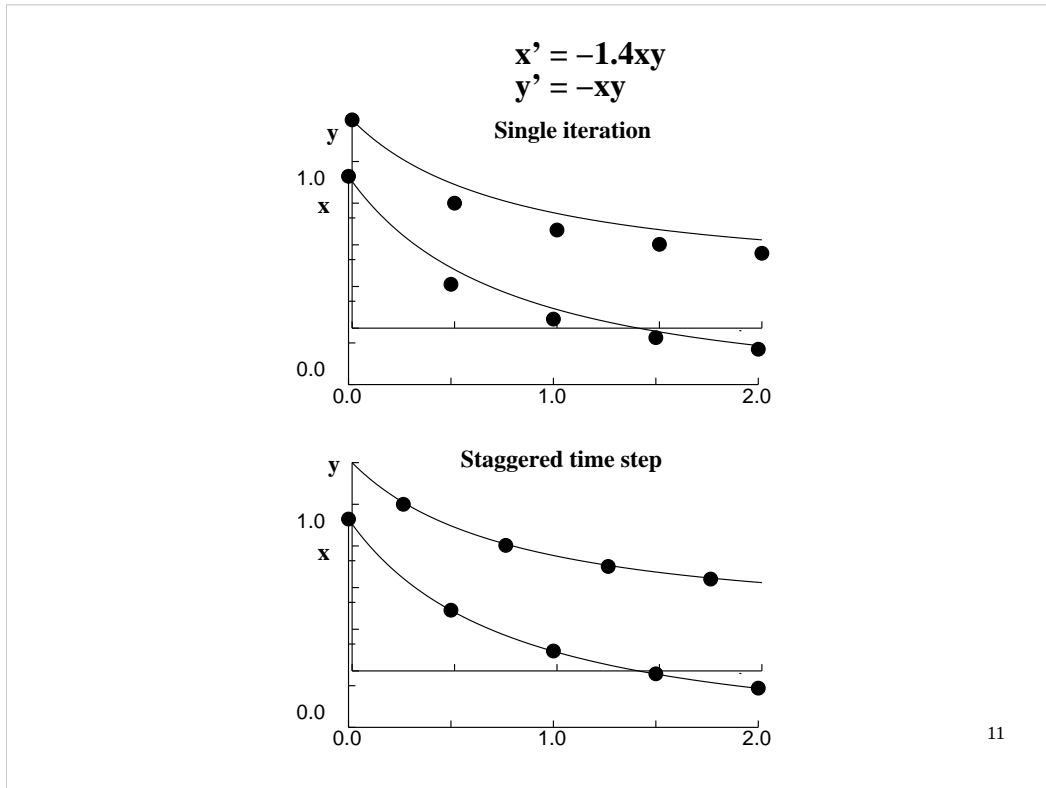
8

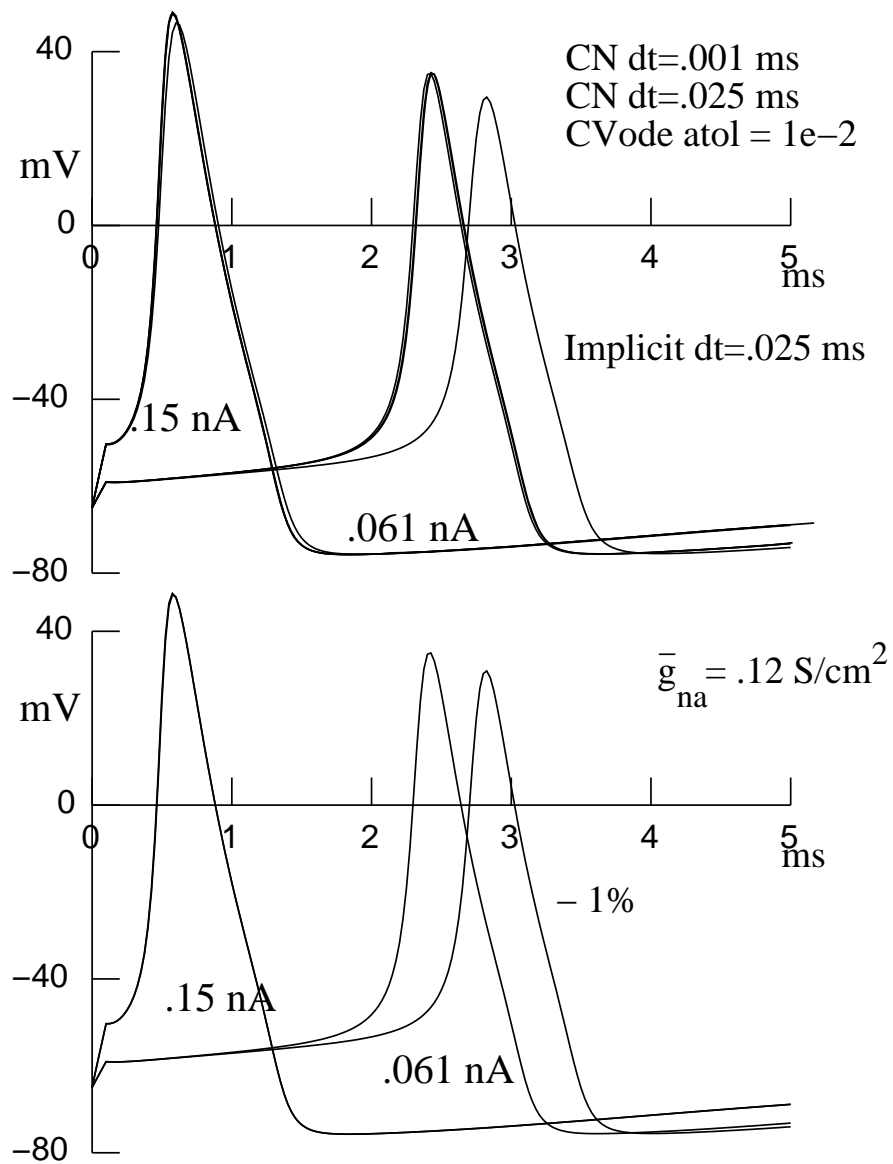


9



10





NMODL

NEURON Model Description Language

Add new membrane mechanisms to NEURON

Density mechanisms

- Distributed Channels
- Ion accumulation

Point Processes

- Electrodes
- Synapses

Described by

- Differential equations
- Kinetic schemes
- Algebraic equations

Benefits

- Specification only — independent of solution method.
- Efficient — translated into C.
- Compact
 - One NMODL statement → many C statements.
 - Interface code automatically generated.
- Consistent ion current/concentration interactions.
- Consistent Units

NMODL general block structure

What the model looks like from outside

```
NEURON {
    SUFFIX kchan
    USEION k READ ek WRITE ik
    RANGE gbar, ...
}
```

What names are manipulated by this model

```
UNITS { (mV) = (millivolt) ... }
PARAMETER { gbar = .036 (mho/cm2) <0, 1e9>... }
STATE { n ... }
ASSIGNED { ik (mA/cm2) ... }
```

Initial default values for states

```
INITIAL {
    rates(v)
    n = ninf
}
```

Calculate currents (if any) as function of v, t, states

(and specify how states are to be integrated)

```
BREAKPOINT {
    SOLVE deriv METHOD cnexp
    ik = gbar * n^4 * (v - ek)
}
```

State equations

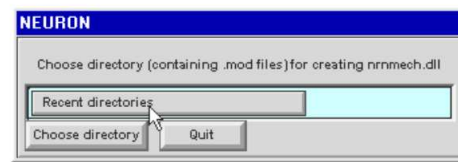
```
DERIVATIVE deriv {
    rates(v)
    n' = (ninf - n)/ntau
}
```

Functions and procedures

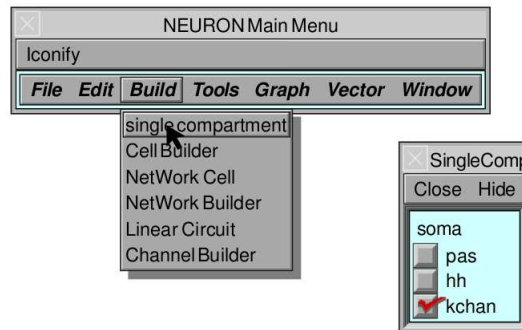
```
PROCEDURE rates(v(mV)) {
    ...
}
```

UNIX

nrnivmodl
nrngui

MSWIN

Select NEURON Main Menu / Build / single compartment



Density mechanism

```

NEURON {
    SUFFIX leak
    NONSPECIFIC_CURRENT i
    RANGE i, e, g
}

PARAMETER {
    g = .001 (mho/cm2) <0, 1e9>
    e = -65 (millivolt)
}

ASSIGNED {
    i (milliamp/cm2)
    v (millivolt)
}

BREAKPOINT {
    i = g*(v - e)
}

```

Point Process

```

NEURON {
    POINT_PROCESS Shunt
    NONSPECIFIC_CURRENT i
    RANGE i, e, r
}

PARAMETER {
    r = 1 (gigaohm) <1e-9,1e9>
    e = 0 (millivolt)
}

ASSIGNED {
    i (nanoamp)
    v (millivolt)
}

BREAKPOINT {
    i = (.001)*(v - e)/r
}

```

Density mechanism**Point Process****NMODL**

```

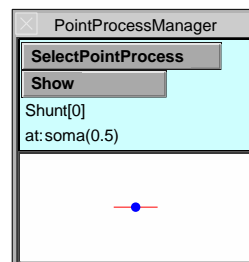
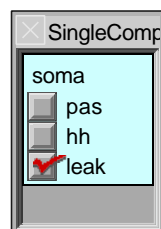
NEURON {
    SUFFIX leak
    NONSPECIFIC_CURRENT i
    RANGE i, e, g
}

```

```

NEURON {
    POINT_PROCESS Shunt
    NONSPECIFIC_CURRENT i
    RANGE i, e, r
}

```

GUI**Interpreter**

```

soma {
    insert leak
    g_leak = .0001
}
print soma.i_leak(.5)

```

```

objref s
soma s = new Shunt(.5)
s.r = 2

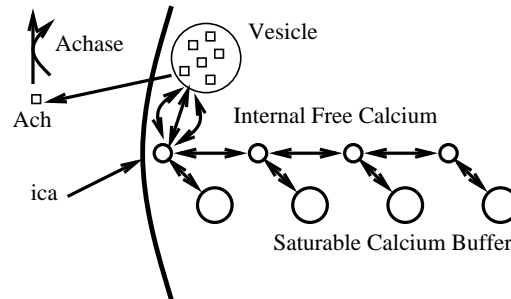
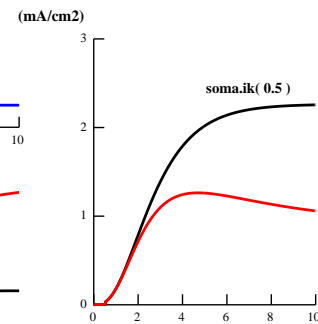
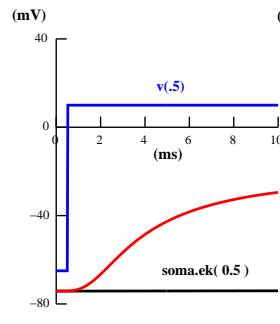
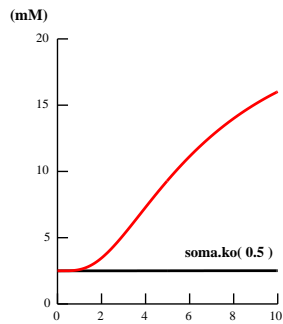
```


Ion Channel

```

NEURON {
    USEION k READ ek WRITE ik
}
BREAKPOINT {
    SOLVE states METHOD cnexp
    ik = gbar*n*n*n*(v - ek)
}
DERIVATIVE states {
    rate(v*1/(mV))
    n' = (inf - n)/tau
}

```



```

STATE {
    Vesicle Ach Achase Ach2ase X Buffer[N] CaBuffer[N] Ca[N]
}
KINETIC calcium_evoked_release {
    : release
    ~ Vesicle + 3Ca[0] <-> Ach (Agen, Arev)
    ~ Ach + Achase <-> Ach2ase (Aase2, 0) : idiom for enzyme reaction
    ~ Ach2ase <-> X + Achase (Aase2, 0) : requires two reactions
    : Buffering
    FROM i = 0 TO N-1 {
        ~ Ca[i] + Buffer[i] <-> CaBuffer[i] (kCaBuffer, kmCaBuffer)
    }
    : Diffusion
    FROM i = 1 TO N-1 {
        ~ Ca[i-1] <-> Ca[i] (Dca*a[i-1], Dca*b[i])
    }
    : inward flux
    ~ Ca[0] << (ica)
}

```

UNITS Checking

```

NEURON { POINT_PROCESS Shunt ... }
PARAMETER {
    e = 0 (millivolt)
    r = 1 (gigaohm) <1e-9,1e9>
}
ASSIGNED {
    i (nanoamp)
    v (millivolt)
}
BREAKPOINT {
    i = (v - e)/r
}

```

Units are incorrect in the "i = ..." current assignment.

```

BREAKPOINT {
    i = (v - e)/r
}

```

**The output from
modlunit shunt
is:**

```

Checking units of shunt.mod
The previous primary expression with units: 1-12 coul/sec
is missing a conversion factor and should read:
(0.001)*()
at line 14 in file shunt.mod
    i = (v - e)/r<>

```

To fix the problem replace the line with:

```
i = (.001)*(v - e)/r
```

What conversion factor will make the following consistent?

$$\begin{array}{ccccccc}
 \text{nai}' & = & \text{ina} & / & \text{FARADAY} & * & (\text{c/radius}) \\
 (\text{uM/ms}) & & (\text{mA/cm}^2) & / & (\text{coulomb/mole}) & & / (\text{um})
 \end{array}$$

The hoc programming language

Based on Kernighan and Pike's "high-order calculator"

Used in

- most published NEURON models
- most NEURON models currently under development
- almost all NEURON models currently available from ModelDB
- NEURON's standard run system and GUI tools
 UNIX / Linux / OS X: **nrnxx/share/nrn/lib/hoc**
 MSWin: **nrnxx\lib\hoc**

Where to learn more:

- Programmer's Reference link at
 <http://www.neuron.yale.edu/neuron/>
- The NEURON Book (especially chapters 12 and 13),
 papers about NEURON
- standard run system and GUI tools
- session files, hoc code exported from CellBuilder, Network Builder
- your own programming experiments

1

Kernighan and Pike's hoc: scalars, arrays, strings, procedures, functions

NEURON adds:

- **Domain-specific features**
 - specification of model properties (geometry, topology, biophysics)
 - event delivery system for synaptic connections,
 artificial spiking cells, and simulation flow control
 - selection of numerical integration method
 - elementary initialization and simulation execution
 - parallel simulation of cell and/or network models
- **Objects**
 - built-in and user-specified classes
- **Graphics**
 - *interactive* plots of variables vs. time, distance ("space plot")
 or another variable (phase plane)
 - shape plots (2D renderings of 3D shapes,
 may show a variable in false color)
 - customizable GUI for building, analyzing, and using models

And it's

- extendable via NMODL, Channel Builder
- Interoperable with Python

2

Starting

UNIX / Linux / OS X / MSWin via the rxvt terminal
nrniv at the system prompt

OS X / MSWin
double click on the nrngui icon

Result:

```
bash-4.1$ nrniv
NEURON -- VERSION 7.3 (725:87d07a86a67e) 2012-08-03
Duke, Yale, and the BlueBrain Project -- Copyright 1984-2012
See http://www.neuron.yale.edu/credits.html

oc>
```

3

Stopping and exiting

Stopping runaway code

^C halts execution "safely"

^C^C halts "immediately"

Exiting hoc

^D or quit() at the oc> prompt

If NEURONMainMenu toolbar ("NMM") exists
NMM / File / Quit

4

Getting code into NEURON

Built-in microemacs

oc>em starts it

See em in the Programmer's Reference, and

<http://www.neuron.yale.edu/neuron/static/docs/help/emacs.txt>

Program files

Plain text (ASCII)

OS X: drag & drop hoc file onto nrngui icon

MSWin: double click on hoc file in Windows Explorer

If NEURONMainMenu toolbar ("NMM") exists

NMM / File / load hoc or NMM / File / load session

xopen("*filename*") reads *filename* on every call

load_file("*filename*") reads *filename* only once per session

Interactive code entry

Type commands at the oc> prompt

5

Interactive code entry

Particularly useful for

- revising and debugging small chunks of code
- using "toy examples" to understand hoc syntax

Keyboard arrow keys:

↑ goes back in command history ("recalls commands"), ↓ goes forward

←, → moves cursor by 1 character, ^←, ^→ by 1 word

EMACS-style cursor control:

^P goes back in command history, ^N forward

^A moves cursor to line start, ^E to end

^B moves back 1 character, ^F forward

esc B moves back 1 word, esc F forward

^K kills to end of line and copies to buffer, ^Y pastes from buffer

6

Basic hoc syntax

Similar to C but no semicolons

Numbers

Numeric values are double precision floating point.

Interpreter

(user input **bold**): Remarks:

oc>**1+0.2** note immediate evaluation

1.2

oc>**1e-3** scientific notation

0.001

oc>**1E-3** 0.001

0.001

oc>**PI** a built-in variable
3.1415927 (treat it like a constant)

7

User-created names

May refer to

- a number ("ordinary" variable or "scalar")
- an array of numbers
- a string
- a function or procedure
- a class ("template")
- an object reference

8

Naming rules

- start with an alpha character [A-Za-z]
- contain alphanumeric characters or underscore _ [A-Za-z0-9_]
- < 100 characters long
- must not conflict with keywords or built-in functions
 - see Programmer's Reference entries on keywords-general, functions-general, keywords-neuron, and functions-neuron
- scope is global except for
 - variables declared local in a procedure or function
 - variables declared in a template
 - "visibility" (public / private)

9

Create a scalar by assigning a value to a new name.

```
oc>x
/usr/local/nrn/i686/bin/nrniv: undefined variable x
near line 14
x
^
oc>

oc>x=2
first instance of x
oc>x
2
oc>
```

10

Anything that isn't a scalar must be defined before use.

```
oc>double y[3]
oc>for i=0,2 y[i]=sqrt(i)
oc>for i=0,2 print y[i]
0
1
1.4142136
```

```
oc>strdef hello
oc>hello="hi"
oc>hello
hi
```

Notes:

- indices start at 0
- if *name* is an array, *name* is a shortcut for *name*[0]

```
oc>print y
0
```

11

A defined name cannot be redefined as something else.

```
oc>strdef y
/usr/local/nrn/i686/bin/nrniv: y already declared
near line 31
strdef y
      ^
```

```
oc>proc hello() { print "hi" }
/usr/local/nrn/i686/bin/nrniv: syntax error
near line 32
proc hello() { print "hi" }
      ^
```

However, the body of a proc or func can be changed.

```
oc>proc foo() { print x^3 }
oc>foo()
8
oc>proc foo() { print x, exp(-x) }
oc>foo()
2 0.13533528
```

12

Expression: a combination of numbers, variables, operators, and functions that, when "executed," produces ("returns") a value of some type (number, string, object class).

```
1
x+2
sin(PI*0.4)
```

Statement: contains one or more expressions

Simple statements

```
x=3 // assignment statement
xopen("cell.hoc") // function call
```

Compound statements

```
{ a=5 b=sqrt(a) } // works, hard to read
{ // one statement/line is better
  a=5
  b=sqrt(a)
}
if (x<1) print "tiny" else print "big"
```

Program: a sequence of one or more statements

13

Statements that span multiple lines

Interactive code entry:

```
oc>proc foo() {
>    oc>print "hello, \
oc>world"
>    oc>}
oc>foo()
hello,
world
```

In a program:

```
i = p*z^2*(V*F^2/(R*T))*(si-so*exp(-z*V*F/(R*T))) \
    /(1 - exp(-z*V*F/(R*T)))
```

14

Comments

```
// a one-line comment
/* another way to comment */
/* a comment
that spans two
or more lines */
```

Indentation

Do whatever you like, but make it readable.

```
{
  x = 5.1
  y = sqrt(3)
  print x*y
}
is better than
{ x = 5.1 y = sqrt(3) print x*y }
```

15

Operator precedence

operator	example	comment
()	2*(x+0.1)	grouping
^	x^2	exponentiation
- !	-3, !x	unaryminus, not
* / %	5%3	multiply, divide, remainder
+ -		add, subtract
> >= < <= != ==	x==2	logical comparison
&&	(x>1) && (x<2)	AND
	(x<0) (x>1e6)	OR
=	x=2	assignment

16

Assignment statements

$x = \text{expression}$

Two shortcuts:

$x += a$ same as $x = x + a$

$x *= a$ same as $x = x * a$

Note: no space between + and =, or * and =

17

Logical expressions and comparisons

```
oc>x=2
```

```
oc>x==2 // does x equal 2?
```

```
1
```

```
oc>x==2 // does x equal 3?
```

```
0
```

Problem: roundoff error.

```
oc>x=sqrt(2)
```

Does x^2 equal 2?

```
oc>x^2-2
```

```
4.4408921e-16
```

How to deal with this?

18

`float_epsilon` sets the threshold
for deciding equality/inequality

```
oc>float_epsilon
1e-11
```

```
oc>y=1+0.9*float_epsilon
oc>y==1
1
```

```
oc>y=1+float_epsilon
oc>y==1
0
```

So even though roundoff error makes
 $\text{sqrt}(2)^2 - 2$ nonzero,

```
oc>sqrt(2)^2==2
1
```

19

Flow control

```
if (expr) stmt
if (expr) stmt1 else stmt2
while (expr) stmt
for (stmt1; expr2; stmt3) stmt
for var = expr1,expr2 stmt
for iterator_name ( . . . ) stmt
```

Examples:

```
i=0
while (i<10) { i+=1 print i }
for (i=0; i<10; i+=1) print i
for i=0,9 print i
```

20

```
oc>x=3
oc>if (x==3) print "x is 3"
      x is 3
```

What if you typed = when you meant == ?

```
oc>x=4
oc>if (x=3) print "x is 3"
      x is 3
```

... and it really will be 3.

Why? Assignment inside () returns a value

```
oc>(x=3)
      3
```

if (number) treats a nonzero *number* as "true"

21

Flow control: iterator

```
// this is included in stdlib.hoc
iterator case() { local i
  for i=2,numarg() {
    $&1 = $1
    iterator_statement
  }
}
// next line requires scalar x to exist
for case(&x, 1, -1, E, R) print x
produces this output:
1
-1
2.7182818
8.31441
oc>
```

22

Functions and procedures

```
func name() { stmt }
```

where *stmt* includes a "return statement"

```
    return expr
```

that returns a value. Example:

```
oc>func three() { return 3 }
oc>three()
    3
```

```
proc name() { stmt }
```

where *stmt* does not include a return statement

23

Arguments to funcs and procs

- scalars, strings or objects
- retrieved by position

```
oc>func quotient() { return $1/$2 }
oc>quotient(1,3)
    0.33333333
```

<u>Variable type</u>	<u>Name of <i>n</i>th argument</u>	<u>Call by</u>
scalar	$\$n$	value
scalar pointer	$\$&n$	reference
string	$\$sn$	reference
object reference	$\$on$	reference

24

Example: scalar and string as arguments

```
oc>proc printerr() { print "Error ", $1, "-- ", $s2 }
oc>printerr(3, "file not open")
Error -- file not open
```

Example: updating and reporting a count. Note call by reference.

```
tally=0
proc count() {
  $&2+=1 // affects arg 2
  print $s1, $&2
}
for i=0,2 count("updated count--", &tally)
```

produces this result:

```
updated count--1
updated count--2
updated count--3
```

25

Local variables

- temporary--exist only while the proc or func is executed
- scope is local to the proc or func,
no conflict with globals that have the same names

Declare as part of the proc *name* { line.

```
i=10
proc squares() { local i
  for i=1,$1 print i*i
}
squares(3)
print "i is ", i
```

produces this result:

```
1
4
9
i is 10
```

26

Classes, objects, and object references in hoc

Class: a type or category.

Object: a specific instance of a type.

Object reference: a label or alias for an object,
not the object itself. Similar to pointer.

In hoc there are no "free-standing" objects.

- If an object exists, there must also be an objref that refers to it.
- If an object's reference count drops to 0, the object is destroyed.

27

Creating and destroying an object

```
oc>objref cells          // make new objref
oc>cells
    NULLObject

oc>cells = new List()    // make new List object
oc>          // and associate cells with it
oc>cells          // verify
    List[8]

oc>List[8]             // just making sure . . .
    List[8]

oc>objref b             // make another objref
oc>b
    NULLObject

oc>b = cells            // associate b with the same List object
oc>b          // verify the association
    List[8]
```

28


```

oc>objref b          // break link between b and List[8]
oc>b                 // verify
      NULLObject

oc>cells             // make sure cells is still associated
      List[8]

oc>objref cells      // break link between cells and List[8]
oc>cells             // verify
      NULLObject

oc>// the List's reference count should now be 0

oc>List[8]           // does the object still exist?
/usr/local/nrn/i686/bin/nrniv: Object ID doesn't exist: List[8]

near line 15
List[8]
  ^

```

29

Using an objref as an argument

```

func totalarea() { local tmp
  tmp = 0 // clear any leftover value
  for $o1.all for (x,0) tmp += area(x)
  return tmp
}
print "total area of ", cell, "is ", totalarea(cell)

```

Using call by reference to modify an object

```

proc scalediam() {
  for $o1.all diam *= $2
}
scalediam(cell, 2) // doubles diam of cell's sections

```

30

obfunc: a function that returns an object

```
// creates customized Graph that plots user-specified variable vs. time
// $s1      name of variable to be plotted
// $2 and $3 y axis min and max
// $4 and $5 screen coordinates of left upper corner
// $6 and $7 graph width and height
// assumes standard run system, so tstop and graphList[0] exist

GWIDTH=300.48 // default width and height of entire graph
GHEIGHT=200.32

obfunc makegraph() { localobj gtmp
  gtmp = new Graph(0) // creates but does not display
  gtmp.size(0,tstop,$2,$3) // axis scaling
  gtmp.view(0, $2, tstop, $3-$2, $4, $5, $6, $7) // draws on the screen
  // with user-specified location and size
  graphList[0].append(gtmp) // so it updates at integer multiples of dt
  gtmp.addexpr($s1, 1, 1, 0.8, 0.9, 2)
  return gtmp
}

objref g
g = makegraph("soma.v(0.5)", -80, 40, 300, 150, GWIDTH, GHEIGHT)

Comment: note use of localobj
```

31

Analyzing a program

Understanding how existing programs work is key to

- debugging and maintenance
- modifying them to handle other tasks
- learning by example

Case study: a hoc file generated by the CellBuilder**Aims:**

- discover how the program works
- identify the programming tactics and strategies that are used in it

32

cell.hoc part 1 of 2

```

proc celldef() {
    topol()
    subsets()
    geom()
    biophys()
    geom_nseg()
}

create soma, dend

proc topol() { local i
    connect dend(0), soma(1)
    basic_shape()
}
proc basic_shape() {
    soma {pt3dclear() pt3dadd(0, 0, 0, 1) pt3dadd(15, 0, 0, 1)}
    dend {pt3dclear() pt3dadd(15, 0, 0, 1) pt3dadd(90, 0, 0, 1)}
}

objref all
proc subsets() { local i
    objref all
    all = new SectionList()
    soma all.append()
    dend all.append()
}

```

33

cell.hoc part 2 of 2

```

proc geom() {
    forsec all { }
    soma { L = 10 diam = 10 }
    dend { L = 1000 diam = 1 }
}
proc geom_nseg() {
    forsec all { nseg = int((L/(0.1*lambda_f(100))+.999)/2)*2 + 1 }
}
proc biophys() {
    forsec all {
        cm = 1
    }
    soma {
        insert hh
        gnabar_hh = 0.12
        gkbar_hh = 0.036
        gl_hh = 0.0003
        el_hh = -54.3
    }
    dend {
        insert pas
        g_pas = 0.0001
        e_pas = -65
    }
}
access soma

celldef()

```

34

```

proc celldef() {
  topol()
  subsets()
  geom()
  biophys()
  geom_nseg()
}

create soma, dend

proc topol() { local i
  connect dend(0), soma(1)
  basic_shape()
}
proc basic_shape() {
  soma {pt3dclear() pt3dadd(0, 0, 0, 1) pt3dadd(15, 0, 0, 1)}
  dend {pt3dclear() pt3dadd(15, 0, 0, 1) pt3dadd(90, 0, 0, 1)}
}

objref all
proc subsets() { local i
  objref all
  all = new SectionList()
  soma all.append()
  dend all.append()
}

```

35

```

proc celldef() {
  topol()
  subsets()
  geom()
  biophys()
  geom_nseg()
}

create soma, dend

proc topol() { local i
  connect dend(0), soma(1)
  basic_shape()
}
proc basic_shape() {
  soma {pt3dclear() pt3dadd(0, 0, 0, 1) pt3dadd(15, 0, 0, 1)}
  dend {pt3dclear() pt3dadd(15, 0, 0, 1) pt3dadd(90, 0, 0, 1)}
}

objref all
proc subsets() { local i
  objref all
  all = new SectionList()
  soma all.append()
  dend all.append()
}

```

The first statement in cell.hoc
that is actually executed.
Time to start building an outline.

36

We need a way to record our discoveries that is quick and easy to set up, and quick and easy to understand.

An outline that summarizes the sequence of program execution is sufficient for most cases.

create soma, dend

37

```

proc celldef() {
  topol()
  subsets()
  geom()
  biophys()
  geom_nseg()
}

create soma, dend

proc topol() { local i
  connect dend(0), soma(1)
  basic_shape()
}
proc basic_shape() {
  soma {pt3dclear() pt3dadd(0, 0, 0, 1) pt3dadd(15, 0, 0, 1)}
  dend {pt3dclear() pt3dadd(15, 0, 0, 1) pt3dadd(90, 0, 0, 1)}
}

objref all
proc subsets() { local i
  objref all
  all = new SectionList()
  soma all.append()
  dend all.append()
}

```

More procedure definitions.
Skip them for now--we're looking for
the next statement that is executed.

38

```

proc celldef() {
    topol()
    subsets()
    geom()
    biophys()
    geom_nseg()
}

create soma, dend

proc topol() { local i
    connect dend(0), soma(1)
    basic_shape()
}
proc basic_shape() {
    soma {pt3dclear() pt3dadd(0, 0, 0, 1) pt3dadd(15, 0, 0, 1)}
    dend {pt3dclear() pt3dadd(15, 0, 0, 1) pt3dadd(90, 0, 0, 1)}
}

objref all
proc subsets() { local i
    objref all
    all = new SectionList()
    soma all.append()
    dend all.append()
}

```

Not very exciting--just a declaration of what kind of variable `all` is--but let's add it to the outline.

And skip the definition of `proc subsets()`; we're still looking for the next instruction that is executed.

39

The updated outline.

```

create soma, dend
objref all

```

40

```

proc geom() {
  forsec all { }
  soma { L = 10 diam = 10 }
  dend { L = 1000 diam = 1 }
}
proc geom_nseg() {
  forsec all { nseg = int((L/(0.1*lambda_f(100))+.999)/2)*2 + 1 }
}
proc biophys() {
  forsec all {
    cm = 1
  }
  soma {
    insert hh
    gnabar_hh = 0.12
    gkbar_hh = 0.036
    gl_hh = 0.0003
    el_hh = -54.3
  }
  dend {
    insert pas
    g_pas = 0.0001
    e_pas = -65
  }
}

```

access soma to this bonanza--two executed statements in a row!

celldef() The pace quickens . . .

Three more procedure definitions.
For now, jump over these . . .

41

The latest version of the outline.

```

create soma, dend
objref all
access soma
celldef()

```

We have to examine `proc celldef()`
to find out what happens next.

42

```

proc celldef() {
  topol()      celldef() makes a lot of things happen.
  subsets()    Let's add these to our outline.
  geom()
  biophys()
  geom_nseg()
}

create soma, dend

proc topol() { local i
  connect dend(0), soma(1)
  basic_shape()
}
proc basic_shape() {
  soma {pt3dclear() pt3dadd(0, 0, 0, 1) pt3dadd(15, 0, 0, 1)}
  dend {pt3dclear() pt3dadd(15, 0, 0, 1) pt3dadd(90, 0, 0, 1)}
}

objref all
proc subsets() { local i
  objref all
  all = new SectionList()
  soma all.append()
  dend all.append()
}

```

43

celldef() really expands the outline--
look at all the procs it calls:

```

create soma, dend
objref all
access soma
celldef()
  topol()      These five
  subsets()    are indented
  geom()       because they
  biophys()    are called
  geom_nseg()  by celldef()

```

Next we examine each of these five procs
to see what they do, and discover if they
call any other procs or funcs.

44


```

proc celldef() {
    topol()
    subsets()
    geom()
    biophys()
    geom_nseg()
}

create soma, dend

proc topol() { local i
    connect dend(0), soma(1)
    basic_shape()
}

proc basic_shape() {
    soma {pt3dclear() pt3dadd(0, 0, 0, 1) pt3dadd(15, 0, 0, 1)}
    dend {pt3dclear() pt3dadd(15, 0, 0, 1) pt3dadd(90, 0, 0, 1)}
}

objref all
proc subsets() { local i
    objref all
    all = new SectionList()
    soma all.append()
    dend all.append()
}

```

Working our way through proc celldef() . . .

First it calls topol()

topol() sets up the topology of the model, then calls basic_shape().

Time to revise the execution outline again.

An aside: why is there an unused local i?

basic_shape() specifies how the model would look in a CellBuilder.

Note the use of "section stack" syntax to specify the currently accessed section.

Also note the compound statements.

Will soma length and diameter really be 15 um and 1 um, respectively?

45

The revised outline.

```

create soma, dend
objref all
access soma
celldef()
    topol()
        basic_shape()
    subsets()
    geom()
    biophys()
    geom_nseg()

```

basic_shape() indented because called by topol()

Next we examine subsets()

46

```

proc celldef() {
    topol()
    subsets()
    geom()
    biophys()
    geom_nseg()
}

create soma, dend

proc topol() { local i
    connect dend(0), soma(1)
    basic_shape()
}
proc basic_shape() {
    soma {pt3dclear() pt3dadd(0, 0, 0, 1) pt3dadd(15, 0, 0, 1)}
    dend {pt3dclear() pt3dadd(15, 0, 0, 1) pt3dadd(90, 0, 0, 1)}
}

objref all
proc subsets() { local i
    objref all
    all = new SectionList()
    soma all.append()
    dend all.append()
}

```

So this is where the various subsets (SectionLists, actually) are assembled. Why is it useful to execute `objref all` inside this proc? Why not just do `objref all` at the top level of the interpreter (i.e. outside of any proc or func or object), and be done with it?

47

```

proc celldef() {
    topol()
    subsets()
    geom()
    biophys()
    geom_nseg()
}

. . .

proc geom() {
    forsec all { }
    soma { L = 10 diam = 10 }
    dend { L = 1000 diam = 1 }
}
proc geom_nseg() {
    forsec all { nseg = int((L/(0.1*lambda_f(100))+.999)/2)*2 + 1 }
}
proc biophys() {
    forsec all {
        cm = 1
    }
    soma {
        . . .
    }
}

```

`geom()` specifies the lengths and diameters of the model's sections. Also, more examples of section stack syntax to specify the currently accessed section. Why is there a `forsec all { }` that doesn't do anything?

48

```

proc celldef() {
  topol()
  subsets()
  geom()
  biophys()
  geom_nseg()
}

. . .

proc biophys() {
  forsec all {
    cm = 1
  }
  soma {
    insert hh
    gnabar_hh = 0.12
    gkbar_hh = 0.036
    gl_hh = 0.0003
    el_hh = -54.3
  }
  dend {
    insert pas
    g_pas = 0.0001
    e_pas = -65
  }
}

. . .

```

Hmm. `geom_nseg()` is defined before `biophys()`,
 but `biophys()` is called before `geom_nseg()`.
 Why is that?

More section stack syntax.
 Applying it to a bunch of statements grouped by `{ }`
 saves a lot of typing.

49

```

proc celldef() {
  topol()
  subsets()
  geom()
  biophys()
  geom_nseg()
}

. . .

proc geom_nseg() {
  forsec all { nseg = int((L/(0.1*lambda_f(100))+.999)/2)*2 + 1 }
}

. . .

```

Last but not least, spatial discretization.
`geom_nseg()` calls `lambda_f()`, which is
 included in `stdlib.hoc`, but we'll count it
 as a call anyway.

50

The complete outline.

```
create soma, dend
objref all
access soma
celldef()
  topol()
    basic_shape()
  subsets()
  geom()
  biophys()
  geom_nseg()
  lambda_f()
```

51

So what can you do now?

Change

- number and names of sections
- model topology
- section geometry
- section biophysics
- discretization strategy

Analyze

- other code generated by the CellBuilder or other GUI tools, e.g. cell classes exported from the CellBuilder, network models exported from the Network Builder
- ses files saved from the GUI, e.g. the RunControl panel
- code mined from the hoc library (not just stdlib.hoc and stdrun.hoc)

to discover how to create special-purpose GUI tools and solve more complex problems.

52

Practical suggestions

Strategy: design programs to have a modular structure. Code that consists of short, relatively simple procedures or functions is easier to develop, debug, and understand.

Tactics: before writing any code, write an outline that breaks the task into manageable steps. Any step that requires more than a couple of statements is a candidate for implementing as a proc or func.

In retrospect, proc `celldef()` was pretty close to being the outline of `cell.hoc`

53

Computational Modeling and Neuroscience

Does computational modeling have a role
in neuroscience research?

1

Best Practices

Know the literature
Collaborate with experimentalists
Use Occam's razor
Adhere to scientific method

2

Scientific Method

Observation

Hypothesis

Prediction

Verification

Evaluation

3

Reproducibility

The ideal:

"Reproducibility is the cornerstone
of scientific method."

"Experiments should be fully described
so that anyone can reproduce them."

The harsh reality:

Velilind's Laws of Experimentation

"If reproducibility may be a problem,
conduct the test only once."

"If a straight line is required,
obtain only two data points."

4

<http://senselab.med.yale.edu/modeldb>



ModelDB provides an accessible location for storing and efficiently retrieving computational neuroscience models. ModelDB is tightly coupled with [NeuronDB](#). Models can be coded in any language for any environment. Model code can be viewed before downloading and browsers can be set to auto-launch the models. For further information, see also, [model sharing in general](#) and [ModelDB in particular](#).

[Submit a new model entry](#)  [Help](#)

Find models by **Find models for** **Find models of**

- * [Model name](#)
- * [First author](#)
- * [Each author](#)
- * [Region\(circuits\)](#)
- * [Cell type](#)
- * [Current](#)
- * [Receptor](#)
- * [Gene](#)
- * [Transmitters](#)
- * [Topic](#)
- * [Simulators](#)
- * [Methods](#)
- * [Networks](#)
- * [Neurons](#)
- * [Electrical synapses \(gap junctions\)](#)
- * [Chemical synapses](#)
- * [Ion channels](#)
- * [Neuromuscular junctions](#)
- * [Axons](#)

Search for models by author name or accession number

Search for SenseLab models using Google [Hints](#)

Combine ModelDB curated keywords with a Google search: [ModelSearch](#)

[Search for publications in ModelDB](#) or [in PubMed](#)

[Register](#) for an account

[Login](#) to access your models

Related [Resources](#)

Some models versions are available in a [mercurial repository](#)

5

Accommodates models from a wide range of simulation environments

721 entries as of 6/17/2012

1 BioPAX	2 KinNeSS	1 PSICS
8 Brian	1 Lua	2 PSpice
79 C / C++	155 MATLAB	3 Pascal/Delphi
1 CNrun	1 MCell	2 PyNN
4 CSIM	1 MOOSE / PyMOOSE	13 Python
8 CalC	1 MVASpike	2 Q/Quick/Turbo Basic
1 Catacomb	1 MadSim	1 QuB
1 CellExcite	1 Mathematica	1 R
1 CellML	1 MySQL	1 SABER
2 ChemoSis	1 NCS	1 SBML
1 Content	2 NEST	21 SNNAP
1 Dynamics Solver	2 NEURONPM	4 SciLab
1 ERNST	1 Nengo	9 Simulink
3 Emergent/PDP++	1 Network	1 Sspice
6 FORTRAN	1 NeuroRD	1 Topographica
1 GNU/Next/Openstep	345 Neuron	3 Virtual Cell
28 Genesis	1 NeuronExperimenter	4 XML
1 IDL	2 Octave	66 XPP
3 IGOR Pro	1 PCSIM	5 neuroConstruct
7 Java	1 PGENESIS	2 parplex

6

Search results

Models by Moore

1. [Frog second-order vestibular neuron models \(Rössert et al. 2011\)](#)
Rössert C, Straka H, Moore LE, Glasauer S (2011) Cellular and network contributions to vestibular signal processing: impact of ion conductances, synaptic inhibition and noise. *J Neurosci* **31**:8359-8372
2. [Engaging distinct oscillatory neocortical circuits \(Vierling-Claassen et al. 2010\)](#)
Vierling-Claassen D, Cardin JA, Moore CI, Jones SR (2010) Computational modeling of distinct neocortical oscillations driven by cell-type selective failure at axon branch points. *J Neurophysiol* **41**:1-8 [PubMed]
11. [Myelinated axon conduction velocity \(Brill et al 1977\)](#)
Brill MH, Waxman SG, Moore JW, Joyner RW (1977) Conduction velocity and spike configuration in myelinated fibres: computed dependence on internode distance. *J Neurol Neurosurg Psychiatry* **40**:769-74 [PubMed]

7

Site of impulse initiation in a neuron (Moore et al 1983)

Accession: 9852

Examines the effect of temperature, the taper of the axon hillock, and HH channel density on antidromic spike invasion into the soma and spike initiation under dendritic stimulation.
Reference: Moore JW, Stockbridge N, Westerfield M (1983) On the site of impulse initiation in a neurone. *J Physiol* **336**:301-311 [PubMed]

Citations [Citation Browser](#)

Model Information (Click on a link to find other models with that property)

Model Type: [Neuron or other electrically excitable cell](#);
Brain
Region(s)/Organism:
Cell Type(s): [Spinal motor neuron](#);
Channel(s): [I_{Na,t}](#); [I_K](#)
Gap Junctions:
Receptor(s):
Gene(s):
Transmitter(s):
Simulation
Environment: [Neuron](#);
Model Concept(s): [Action Potential Initiation](#); [Simplified Models](#);
Implementer(s): [Hines, Michael](#);

Search NeuronDB for information about: [Spinal motor neuron](#); [I_K](#); [I_{Na,t}](#)

Model files	Download zip file	Auto-launch	Help downloading and running models
<ul style="list-style-type: none"> moore83 README mosinit.hoc init.hoc start.ses 	<p>Moore, Stockbridge, and Westerfield. (1983) On the site of impulse initiation in a neurone. <i>J. Physiol.</i> 336: 301-311.</p> <p>This model qualitatively reproduces figures 1-5. Note that orthodromic stimulus amplitude is considerably different from that noted in the paper. IClamp[0].amp was chosen to give qualitative similarity. We attribute minor quantitative differences to the following:</p> <ol style="list-style-type: none"> 1) The precise site of axon v vs t curve is not specified. We plot axon.v(0.25). 2) The antidromic stimulus was unspecified. <p>The NEURON implementation of this model was prepared by Michael Hines. Questions about details of this implementation should be addressed to him at michael.hines@yale.edu.</p>		

8

How to proceed

Read abstract / paper

Download, extract zip, compile mod files,
run mosinit.hoc

Analyze model

ModelView

topology(), Shape plot

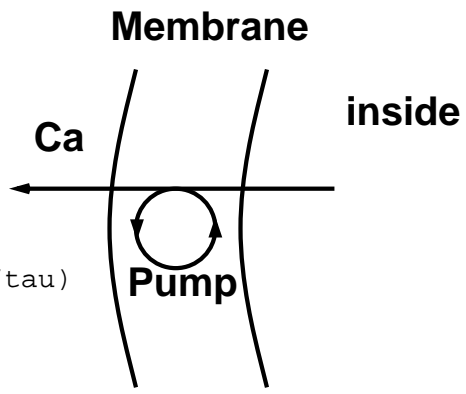
forall psection()

Read code . . .

Reusable components?

9

Rate limited active transport



```

KINETIC pmp {
  ~ cabulk <-> cam          (1/tau, 1/tau)
  ~ cam + pump <-> capump   (k1, k2)
  ~ capump <-> cao + pump   (k3, k4)
  ica_pmp = 2*FARADAY*(f_flux - b_flux)

  ~ cam << -(ica)           : there is a problem here

  COMPARTMENT width {cam}    : volume has dimensions of (um)
  COMPARTMENT 1 {pump capump} : area is dimensionless
  COMPARTMENT 1(m) {cao cabulk}

}

```

Declarations for capump.mod

```

NEURON {
  SUFFIX capmp
  USEION ca READ cao, ica, cai WRITE cai, ica
  RANGE tau, width, cabulk, ica, pump0
}

UNITS {
  (um) = (micron)
  (molar) = (1/liter)
  (mM) = (millimolar)
  (uM) = (micromolar)
  (mA) = (milliamp)
  (mol) = (1)
  FARADAY = (faraday) (coulomb)
}

```

Declarations for capump.mod

```

PARAMETER {
    width = 0.1 (um)
    tau = 1 (ms)
    k1 = 5e8 (/mM-s)
    k2 = 0.25e6 (/s)
    k3 = 0.5e3 (/s)
    k4 = 5e0 (/mM-s)
    cabulk = 0.1 (uM)
    pump0 = 3e-14 (mol/cm2)
}

STATE {
    cam (uM) <1e-6>
    pump (mol/cm2) <1e-16>
    capump (mol/cm2) <1e-16>
}

ASSIGNED {
    cao (mM) : 10
    cai (mM) : 1e-3
    ica (mA/cm2)
    ica_pmp (mA/cm2)
    ica_pmp_last (mA/cm2)
}

```

Equations for capump.mod

```

INITIAL {
    ica = 0  ica_pmp = 0
    ica_pmp_last = 0
    SOLVE pmp STEADYSTATE sparse
}

BREAKPOINT {
    SOLVE pmp METHOD sparse
    ica_pmp_last = ica_pmp
    ica = ica_pmp
}

KINETIC pmp {
    ~ cabulk <-> cam (width/tau, width/tau)
    ~ cam + pump <-> capump ((1e7)*k1, (1e10)*k2)
    ~ capump <-> cao + pump ((1e10)*k3, (1e10)*k4)
    ica_pmp = (1e-7)*2*FARADAY*(f_flux - b_flux)

    : ica_pmp_last vs ica_pmp needed because of STEADYSTATE
    ~ cam << (-(ica - ica_pmp_last)/(2*FARADAY)*(1e7))

    CONSERVE pump + capump = (1e13)*pump0
    COMPARTMENT width {cam} : volume has dimensions of um
    COMPARTMENT (1e13) {pump capump}: area is dimensionless
    COMPARTMENT 1(um) {cabulk}
    COMPARTMENT (1e3)*1(um) {cao}

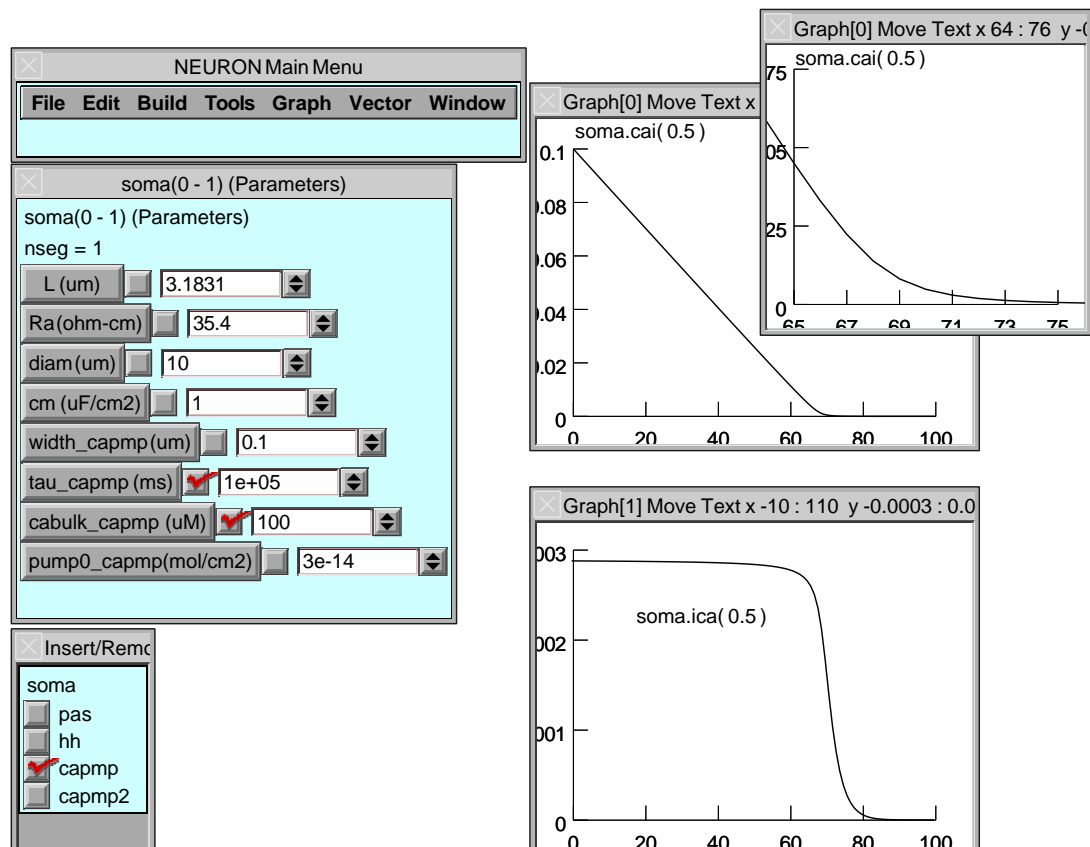
    cai = (0.001)*cam
}

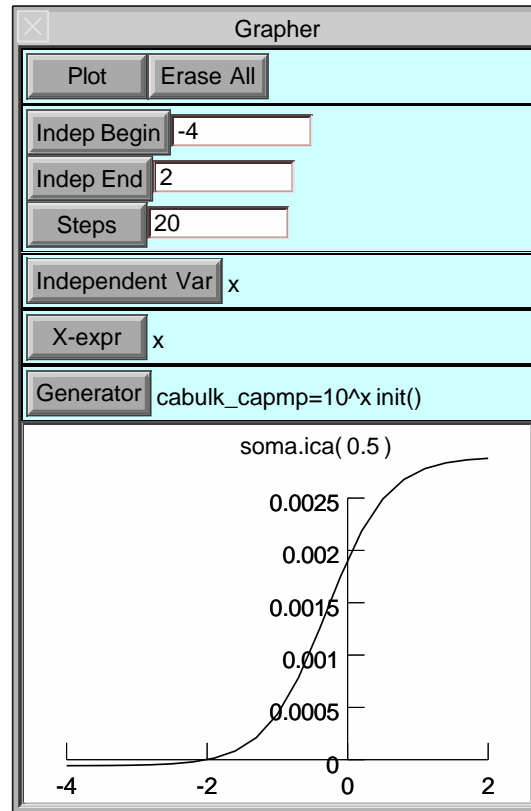
```

Testing capump.mod

```
load_file("nrngui.hoc")

// define a replacement for the stdrun.hoc version of
proc init() {
    finitialize(v_init)
    fcurrent()
}
// that lets you escape from the tyranny
// of the steady state initialization of cai.
proc init() { local savtau
    // will initialize cai to cabulk
    savtau = tau_capmp
    tau_capmp = 1e-6
    finitialize(v_init)
    tau_capmp = savtau
    fcurrent()
    if (cnode.active()) { cnode.re_init() }
}
```

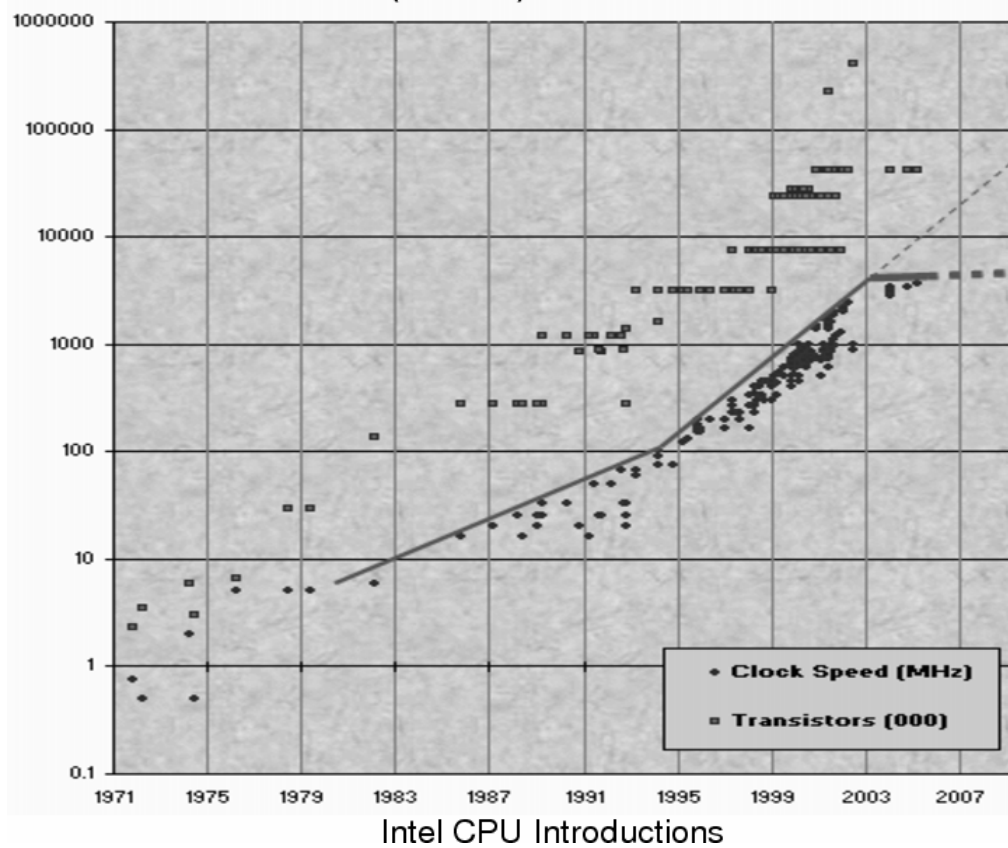




NEURON + Threads

Simulations on multicore desktops.

No (more) Free Lunch



Thread style in NEURON

Join

```
run() {
  while (t < tstop) {
    multithread_job(step)
    plot()
  }
}
```

```
void* step(NrnThread* nt) {
  ... nt->id ...
}
```

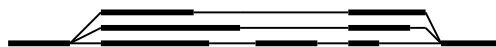


We never use.

Condition Wait

```
multithread_job(run)
```

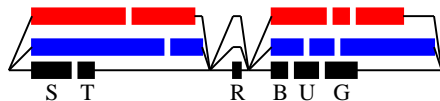
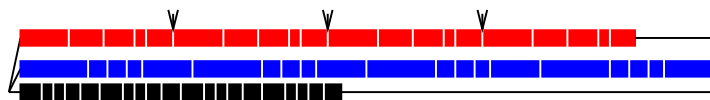
```
run(NrnThread* nt) {
  while(t < tstop) {
    step(nt)
    barrier()
    if (nt->id == 0) { plot() }
    barrier()
  }
}
```



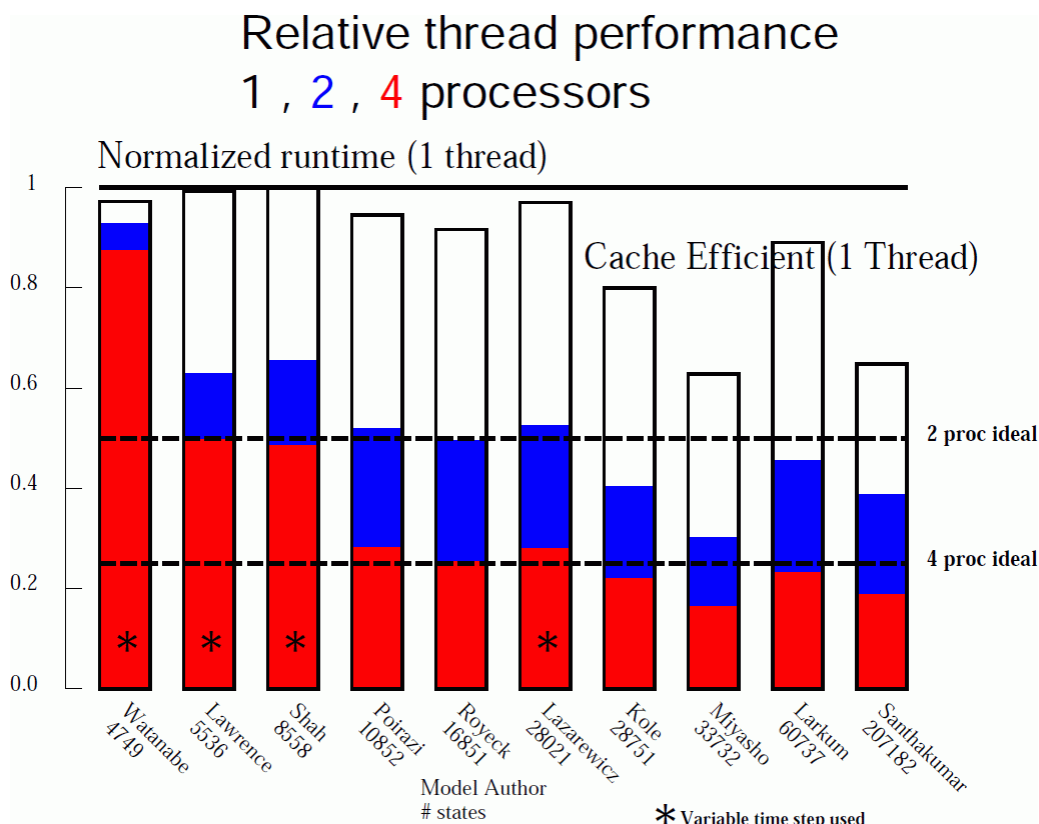
Reminiscent of MPI

Fixed step: $t \rightarrow t + dt$

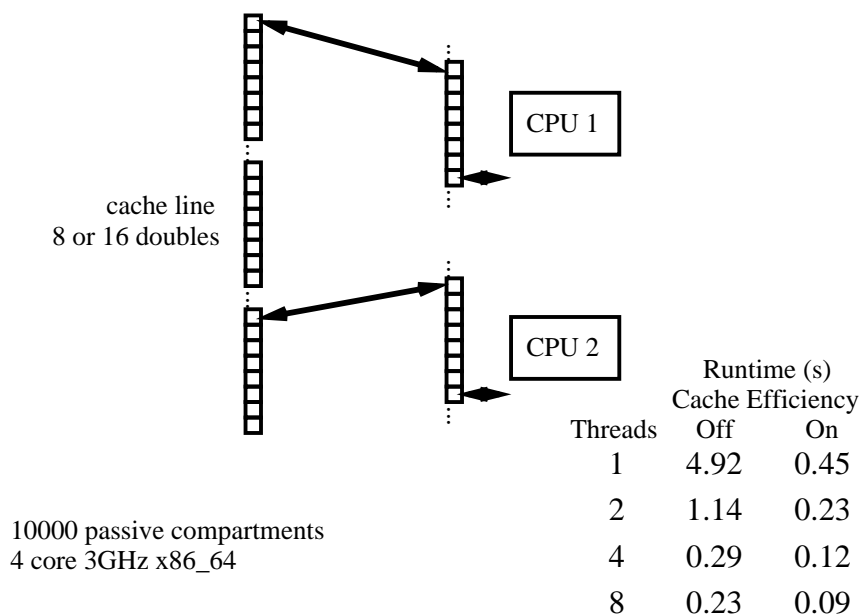
S	T	R	B	U	G
setup	triang	reduce solve	bksub	update	cond gates



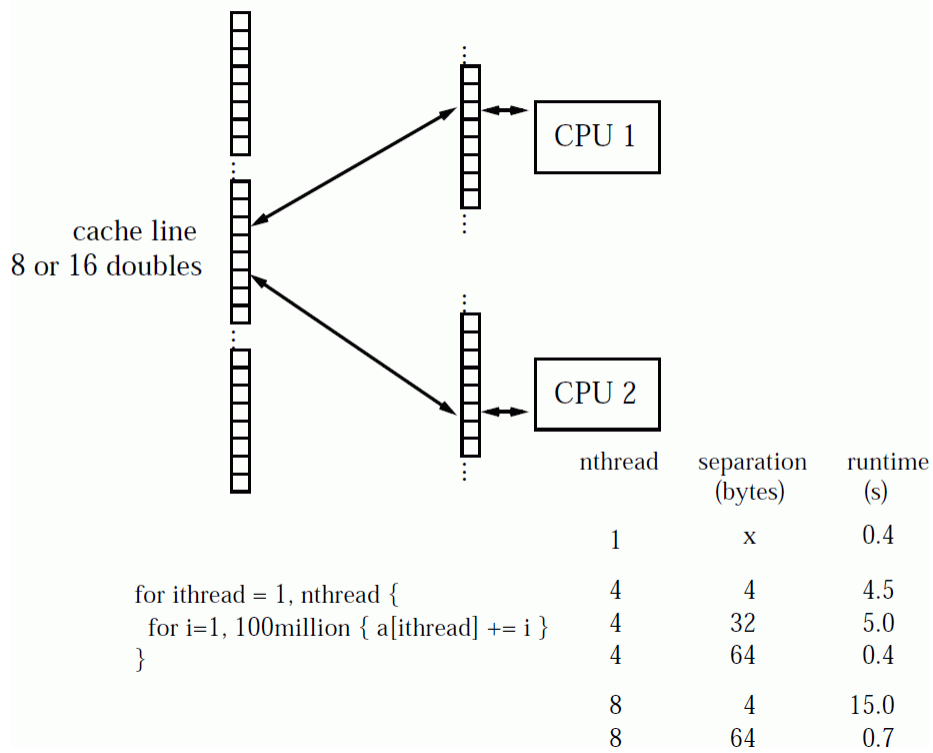
Global var dt $y' = f()$ dy'/dy
27 ||Vector operations



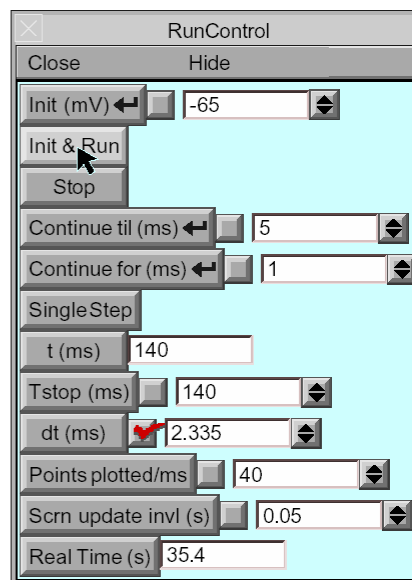
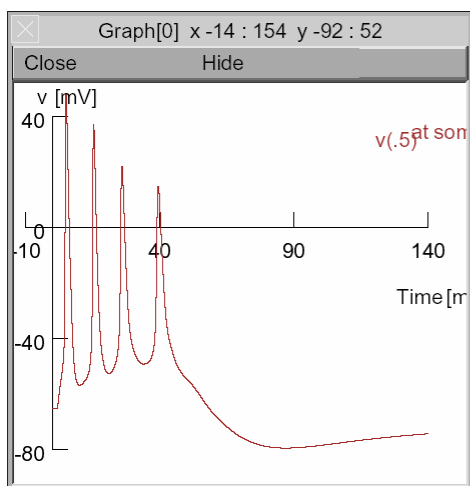
Ideal cache efficiency

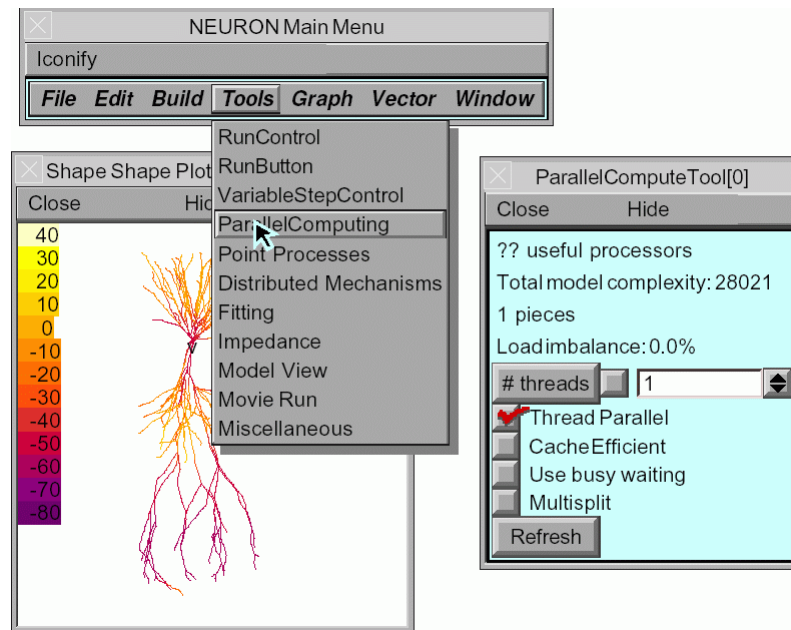


False cache line sharing



Lazarewicz 2002, CA3 Pyramidal Neuron





The first screenshot shows the 'ParallelComputeTool[0]' window with the following configuration:

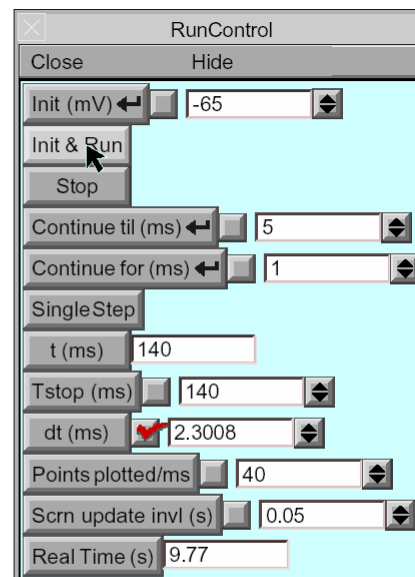
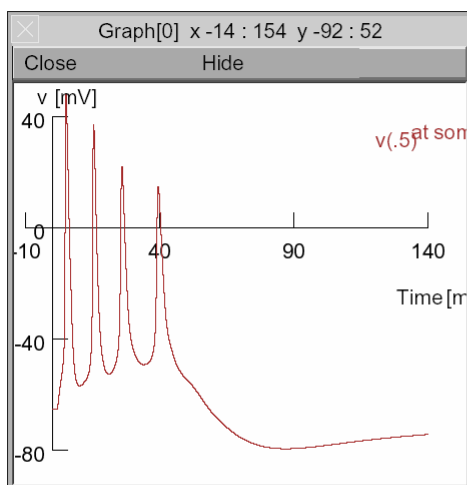
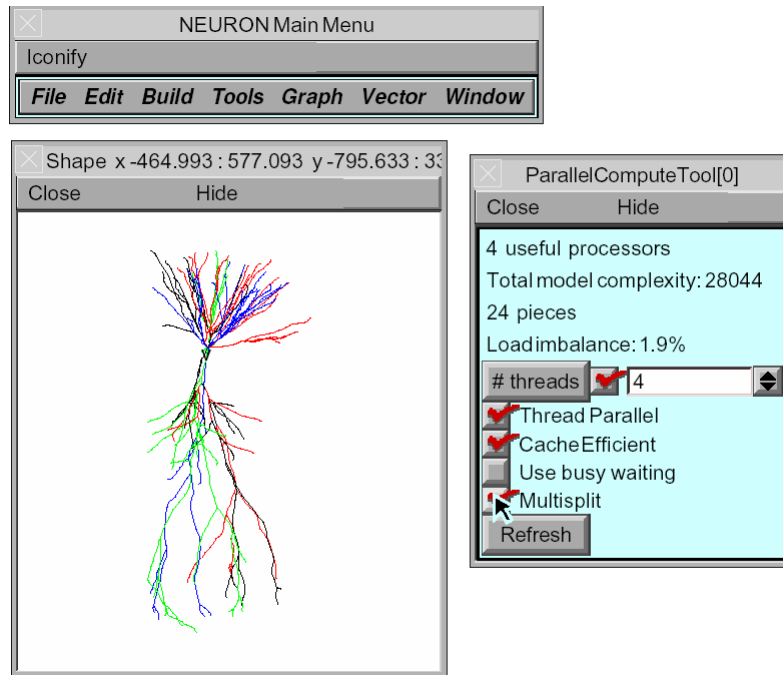
- Close Hide
- 4 useful processors
- Total model complexity: 28021
- 1 pieces
- Load imbalance: 0.0%
- # threads: 1
- ☒ Thread Parallel
- ☐ CacheEfficient
- ☐ Use busy waiting
- ☐ Multisplit
- Refresh

The second screenshot shows the same window with the following configuration:

- Close Hide
- 4 useful processors
- Total model complexity: 28021
- 1 pieces
- Load imbalance: 300.0%
- # threads: 4
- ☒ Thread Parallel
- ☐ CacheEfficient
- ☐ Use busy waiting
- ☐ Multisplit
- Refresh

oc>nthread walltime (count to 1e8 on each thread)

1	0.0500002
2	0.0599999
4	0.0599999
8	0.14



instead of 35.4s

\$ mkthreadsafe

```
NEURON {
    SUFFIX CAIM95
    USEION ca READ cai,cao WRITE ica
    RANGE gbar,ica
    GLOBAL minf,tau
}
```

Translating CAIM95.mod into CAIM95.c

Notice: Assignment to the GLOBAL variable, "minf", is not thread safe

Notice: Assignment to the GLOBAL variable, "tau", is not thread safe

Force THREADSAFE? [y][n]: n

```
DERIVATIVE state {
    rate(v)
    m' = (minf - m)/tau
}
```

```
PROCEDURE rate(v (mV)) {
    LOCAL a
    a = alp(v)
    tau = 1/(tfa*(a + bet(v)))
    minf = tfa*a*tau
}
```

Force THREADSAFE? [y][n]: n
y

```
NEURON {
    THREADSAFE
    SUFFIX CAIM95
    USEION ca READ cai,cao WRITE ica
    RANGE gbar,ica
    GLOBAL minf,tau
}
```

\$ mkthreadsafe

```

NEURON {
    POINT_PROCESS GABAA
    POINTER pre
    ...
}
    VERBATIM
    return 0;
    ENDVERBATIM

```

Translating gabaa.mod into gabaa.c

Notice: Use of POINTER is not thread safe.

Notice: VERBATIM blocks are not thread safe

Notice: Assignment to the GLOBAL variable, "Rtau", is not thread safe

Notice: Assignment to the GLOBAL variable, "Rinf", is not thread safe

Force THREADSAFE? [y][n]: n

\$ mkthreadsafe

```

NEURON {
    SUFFIX Kv
    USEION k READ ek WRITE ik
    RANGE n, gk, gbar
    RANGE ninf, ntau
    GLOBAL Ra, Rb
    GLOBAL q10, temp, tadj, vmin, vmax
}

```

Translating kv.mod into kv.c

Notice: This mechanism cannot be used with CVODE

Notice: Assignment to the GLOBAL variable, "tadj", is not thread safe

Force THREADSAFE? [y][n]: n


```

NEURON {
  GLOBAL q10, temp, tadj, vmin, vmax
INITIAL {
  trates(v)
  n = ninf
}
BREAKPOINT {
  SOLVE states
  gk = tadj*gbar*n
  ik = (1e-4) * gk * (v - ek)
}
PROCEDURE trates(v) {
  TABLE ninf, nexp
  tadj = q10^((celsius - temp)/10)

```

```

NEURON {  THREADSAFE
  GLOBAL q10, temp, tadj, vmin, vmax
INITIAL {
  trates(v)  tadj = q10^((celsius - temp)/10)
  n = ninf
}
BREAKPOINT {
  SOLVE states
  gk = tadj*gbar*n
  ik = (1e-4) * gk * (v - ek)
}
PROCEDURE trates(v) {
  TABLE ninf, nexp
  tadj = q10^((celsius - temp)/10)

```

... a case often seen in ca accumulation models

```

NEURON {
  GLOBAL vol, Buffer0

  ...

  INITIAL {

    if (coord_done == 0) {
      coord_done = 1
      coord()
    }

    ...

    vol[0] = 0
    FROM i=0 TO NANN-2 {
      vol[i] = vol[i] + PI*(r-dr2/2)*2*dr2
    }

    ...

    vol[i+1] = PI*(r+dr2/2)*2*dr2
  }

```

```

NEURON {
  GLOBAL vol, Buffer0
  THREADSAFE vol

  ...

  INITIAL {
    MUTEXLOCK
    if (coord_done == 0) {
      coord_done = 1
      coord()
    }
    MUTEXUNLOCK
  }

  ...

  vol[0] = 0
  FROM i=0 TO NANN-2 {
    vol[i] = vol[i] + PI*(r-dr2/2)*2*dr2
  }

  ...

  vol[i+1] = PI*(r+dr2/2)*2*dr2

```

**If thread results differ,
a good way to diagnose the
cause is to use prcellstate.hoc**

\$ nrngui mosinit.hoc ../prcellstate.hoc

```
// serial model
finitialize(-70)
prtop(0) // constructs cs0.0.1 (5MB)
```

```
//switch to 4 threads
finitialize(-70)
prtop(1) // constructs cs1.0.1
```

diff cs*|more
**notice differences in ik and ica
and in particular**

```
595,605c595,605
< 0 594 0.29053584721744774 gk_km(0.0454545)
< 0 595 0.29053584721744774 gk_km(0.136364)
---
> 0 594 0 gk_km(0.0454545)
> 0 595 0 gk_km(0.136364)
```

```
672,682c672,682
< 0 671 7.8321478840514193e-12 gca_sca(0.0454545)
< 0 672 7.8321478840514193e-12 gca_sca(0.136364)
---
> 0 671 0 gca_sca(0.0454545)
> 0 672 0 gca_sca(0.136364)
```


Two kinds of parallel problems

A simulation run takes about a second.

Want to do 1000's of them,
varying a dozen or so parameters.

A simulation of a large network takes hours.

Want to spread the problem over several machines,
each machine handling a subset of the neurons in the network

Serial

```
s = 0
for i = 1, 10 {
  s += f(i)
}
```

Parallel

```
s = 0
for i = 1, 10 {
  pc.submit("f", i)
}
while (pc.working) {
  s += pc.retval
}
```

Goals

Keep all the machines as busy as possible.

If there is only one machine the parallel program
should run as fast as the serial program.

Things asked for earlier tend to get done earlier.

Assumptions

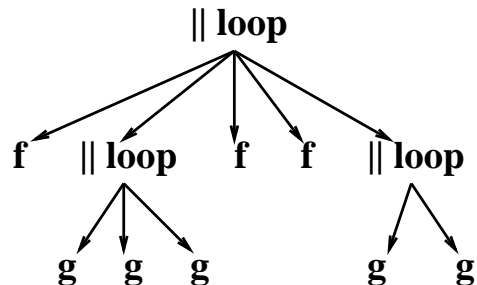
Workstation cluster – 1, 3, 15, 100 machines.

Wide variety of machine speeds.

Sending a byte is much slower than
executing a hoc statement.

Domain

Very coarse grain parallelization.



NEURON's style

A bulletin board
... on top of MPI.

Launching a parallel program

```

objref pc
pc = new ParallelContext()

    // setup which is exactly
    // the same on every machine
    // i.e. declaration of all
    // functions, procedures,
    // setup of neurons

pc.runworker

    // the master scatters tasks
    // onto the bulletin board
    // and gathers results

pc.done
  
```

Example.hoc

```
objref pc
pc = new ParallelContext()

func f() {local s, i
    s = 0
    for i=1,100000 {
        s += $1
    }
    return s
}

{pc.runworker()}

runtime = startsw()
s = 0
for i=1,10 {
    pc.submit("f", i)
}
while (pc.working()) {
    s += pc.retval
}
print "sum = ", s
print "runtime ", startsw() - runtime
{pc.done()}
quit()
```

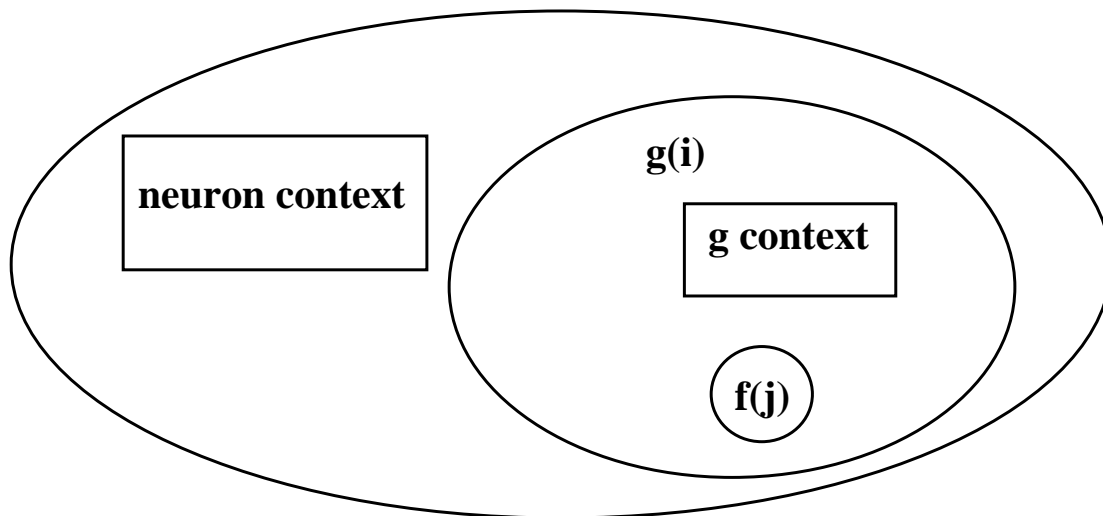
```
$ mpiexec -n 1 nrniv -mpi example.hoc
numprocs=1
NEURON -- VERSION 7.2 (428:986821b56b98) 2010-03-17
...
sum = 5500000
runtime 0.079999924
```

```
$ mpiexec -n 4 nrniv -mpi example.hoc
numprocs=4
NEURON -- VERSION 7.2 (428:986821b56b98) 2010-03-17
...
sum = 5500000
runtime 0.019999981
```

```
$
```

Context and Communication

NEURON



post \longrightarrow **Bulletin board**

take
look \longleftarrow **Bulletin board**
look_take

context("stmt") : stmt executed on every worker

Context and Communication

With Python

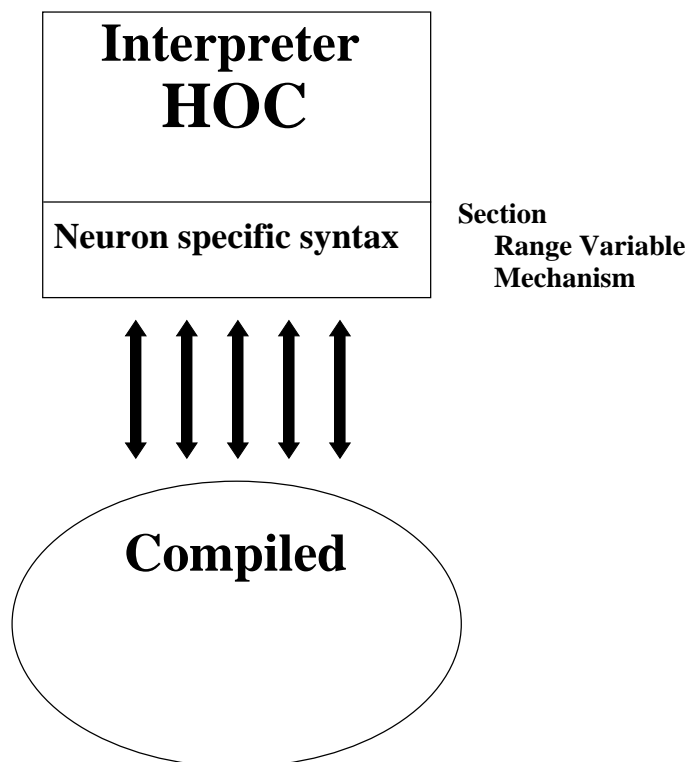
```
def f(arg1, arg2):  
    ...  
    return any_pickleable_object  
  
...  
pc.submit(f, (arg1, arg2))  
...  
while pc.working():  
    r = pc.pyret()
```

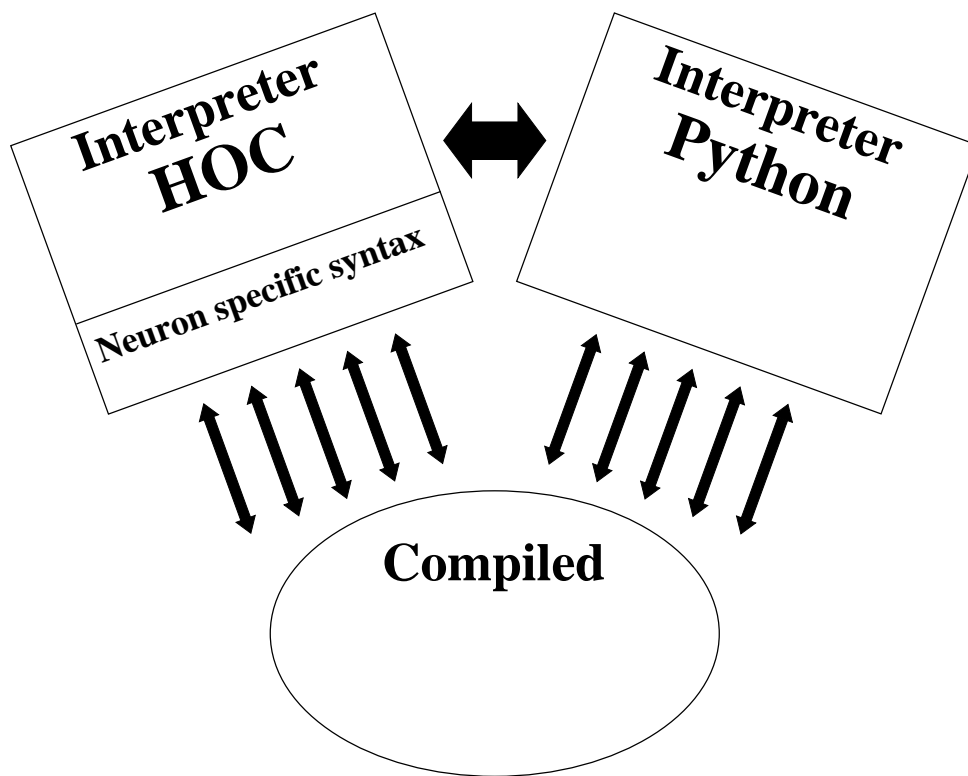

Python + NEURON

All legacy models must work.

Superior representation of
underlying concepts.

No extra installation difficulty.





Installation

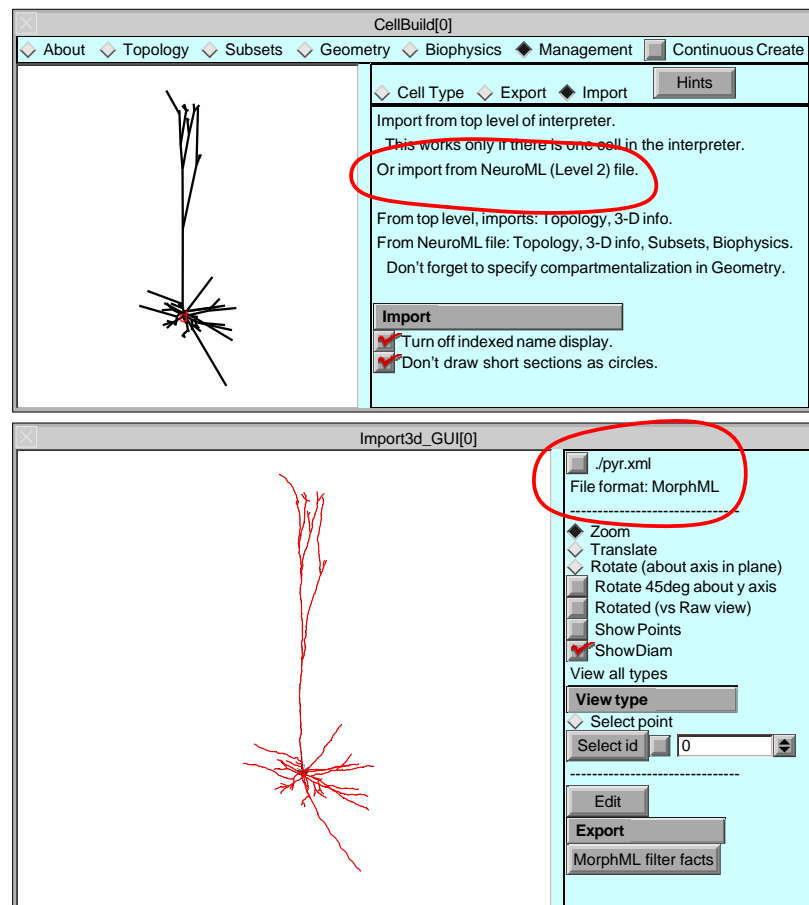
```
>>> import neuron
```

Linux i686
 x86_64

Mac OS X 10.5–8 Python 2.3–4
 2.5–7
 3.0–2

MSWin Cygwin
 MinGW

Launch NEURON NumPy
 Python



```
$ nrniv -python
```

```
NEURON -- VERSION 7.3 ...
```

```
>>> from neuron import h
```

```
>>> print h
```

```
<hoc.HocObject object at 0x2b4f1b81e030>
```

```
>>> print h.hname()
```

```
TopLevelHocInterpreter
```

```
>>> h('''
```

```
... x = 5
```

```
... strdef s
```

```
... s = "hello"
```

```
... func square() { return $1*$1 }
```

```
... ''')
```

```
1
```

```
>>> print h.x, h.s, h.square(4)
```

```
5.0 hello 16.0
```

```
>>> v = h.Vector(4).indgen().add(10)
>>> print v.hname(), len(v), v.size(), v.x[2], v[2]
Vector[1] 4 4.0 12.0 12.0
>>> v.printf()
10      11      12      13
4.0
>>> for x in v: print x
...
10.0
11.0
12.0
13.0
>>>
```

```
>>> import numpy
>>> na = numpy.arange(0, 10, 0.00001) # 0.0131
>>> v = h.Vector(na)                  # 0.0197
>>> v.size()
1000000.0
>>> nb = numpy.array(v)               # 0.0125
>>> nb[999999]
9.999990000000000004
>>> b = list(v)                      # 0.0717
>>> for i in xrange(0, len(nb)):
...     v.x[i] = na[i]
...                                  # 3.7497
```

```

>>> def callback(a = 1, b = 2):
...     print "callback: a=%d b=%d" % (a, b)
...
>>> fih = h.FInitializeHandler(callback)
>>> h.finitialize()
callback: a=1 b=2
1.0
>>> fih = h.FInitializeHandler((callback,\
... (4, 5)))
>>> h.finitialize()
callback: a=4 b=5
1.0
>>>

```

```
# assume hh soma model
```

```

vvec = h.Vector()
vvec.record(soma(.5)._ref_v, sec=soma)

```

```

tvec = h.Vector()
tvec.record(h._ref_t, sec=soma)

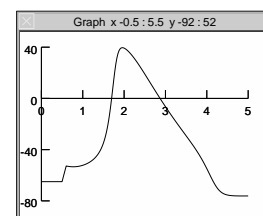
```

```
h.run()
```

```

g = h.Graph()
g.size(0, 5, -80, 40)
vvec.line(g, tvec)

```



```

>>> from neuron import h
>>> soma = h.Section(name = 'soma')
>>> axon = h.Section()
>>> axon.connect(soma(1))
>>> axon.nseg = 5
>>> h.topology()

```

```

| - |          soma(0-1)
    \-----|          PySec_2b371cd17190(0-1)

```

```
1.0
```

```

>>> axon.L = 1000
>>> axon.diam = 1

>>> for sec in h.allsec():
...     sec.cm = 1
...     sec.Ra = 100
...     sec.insert('hh')
...

>>> axon.gnabar_hh = .1
>>> axon(.5).hh.gnabar = .09
>>> for seg in axon:
...     print seg.x, seg.hh.gnabar
...
0.1 0.1
0.3 0.1
0.5 0.09
0.7 0.1
0.9 0.1

```



```
>>> stim = h.IClamp(soma(.5))
>>> stim.delay = .5
>>> stim.dur = .1
>>> stim.amp = .4
```

```
class Cell(object):
    def __init__(self):
        self.topology()
        self.subsets()
        ...
    def topology(self):
        self.soma = h.Section(cell = self)
        self.dend = h.Section(cell = self)
        self.dend.connect(self.soma)
        ...
    def subsets(self):
        self.all = h.SectionList()
        self.all.wholetree(sec=self.soma)
```

