

Table of Contents and Schedule of Presentations

NTC Ted Carnevale

MLH Michael Hines

Hands-on exercises are indicated by an asterisk * in the Page column.

Times shown are approximate, except for lunch.

Thursday, 9/9 Morning session

Time	Speaker	Title	Page
9:00 AM	MLH	Welcome to the NEURON course	
9:15	NTC	Introduction to modeling with NEURON	5
9:45	NTC	Example: single compartment	9 *
10:30	Coffee Break		
10:45	NTC	Fundamental concepts	11
11:15	MLH	Hodgkin-Huxley axon	17 *
12:15	End of morning session		
12:30	Lunch		

Afternoon session

1:30	NTC	Ball and stick model	19 *
3:00	Coffee Break		
3:15	NTC	An outline for creating and using NEURON models	21
3:30	MLH	NMODL: the NEURON Model Description Language	27 *
5:00	End of afternoon session		

Friday, 9/10 Morning session

Time	Speaker	Title	Page
9:00 AM		Questions and Answers	
9:15	MLH	Numerical methods: accuracy, stability, speed	37
10:30		Coffee Break	
10:45	NTC	ModelDB and Model View	49 *
12:15		End of morning session	
12:30		Lunch	

Afternoon session

1:30	NTC	Working with morphometric data	55 *
2:30	NTC	Inhomogeneous channel distributions	59 *
3:30		Coffee Break	
3:45	MLH	Variable time steps and parameter discontinuities	61 *
5:00		End of afternoon session	

Saturday, 9/11 Morning session

Time	Speaker	Title	Page
9:00 AM		Questions and Answers	
9:15	NTC	Networks: synapses, events, and artificial spiking cells	81 *
10:45		Coffee Break	
11:00	MLH	Networks: inhibitory synchronizing network	93 *
12:15		End of morning session	
12:30		Lunch	

Afternoon session

1:30	MLH	Parallel computation: distributed network models	99
3:15		Coffee Break	
3:30	MLH	NEURON + threads	117 *
5:00		End of afternoon session	
Optional	NTC	Initialization	129 *
Optional	NTC	NEURON's tools for analyzing electrotonus	137 *
Optional	MLH	The Channel Builder	141
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).		

The What and the Why of Neural Modeling

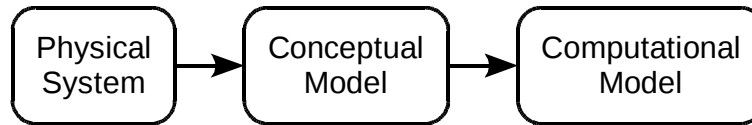
The moment-to-moment processing of information in the nervous system involves the propagation and interaction of electrical and chemical signals that are distributed in space and time.

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



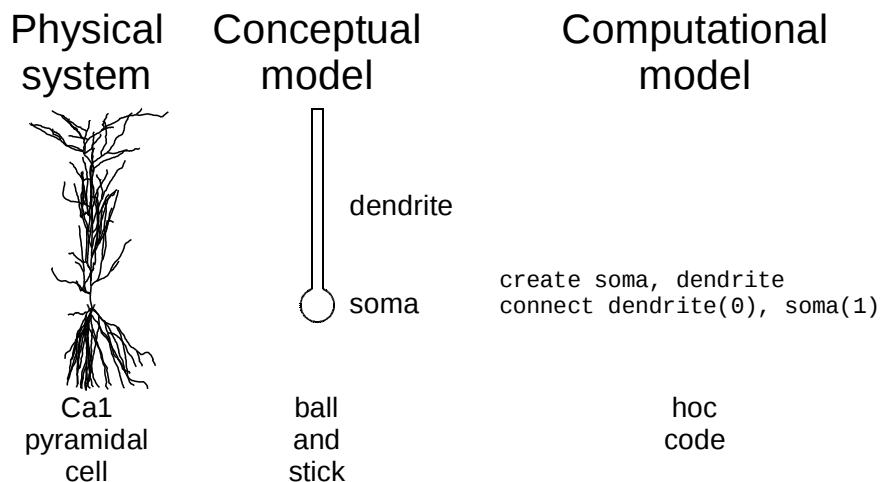
Conceptual model

a simplified representation of the physical system

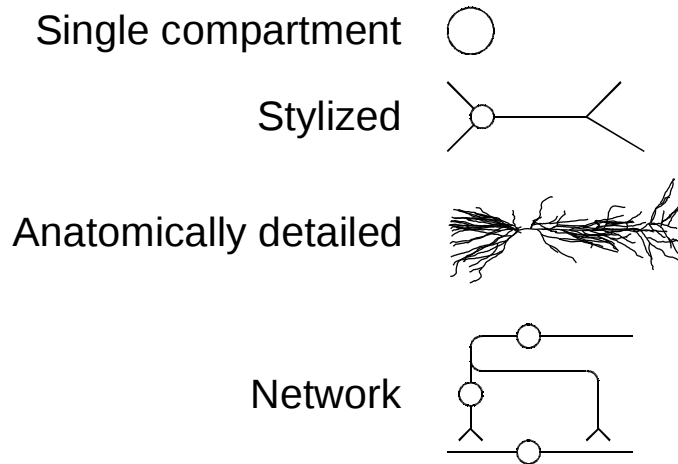
Computational model

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

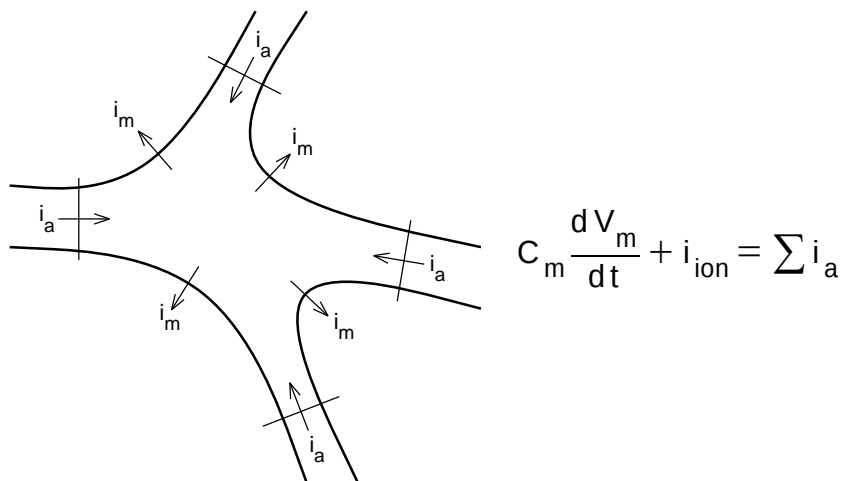
Diffusion, buffers, transport & exchange

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:

- Run simulation
- Change stimulus intensity and duration
- Adjust graphical displays of simulation results
- Adjust dt and Points Plotted / ms

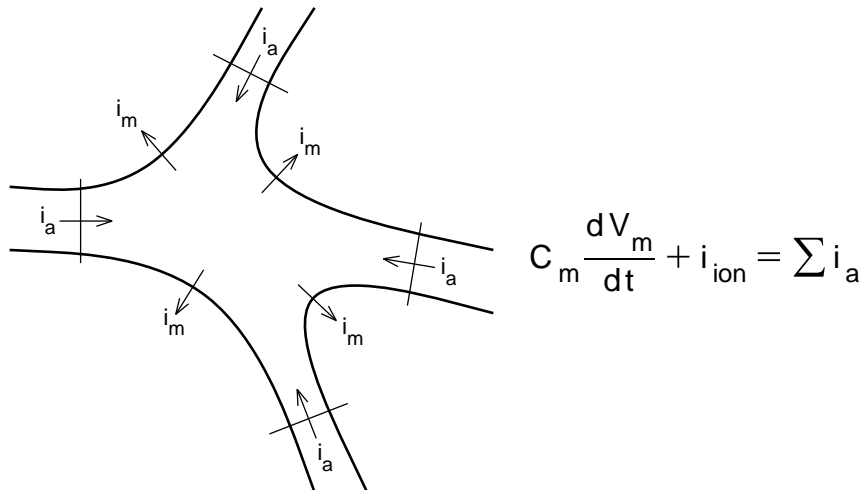
Fundamental Concepts in NEURON

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

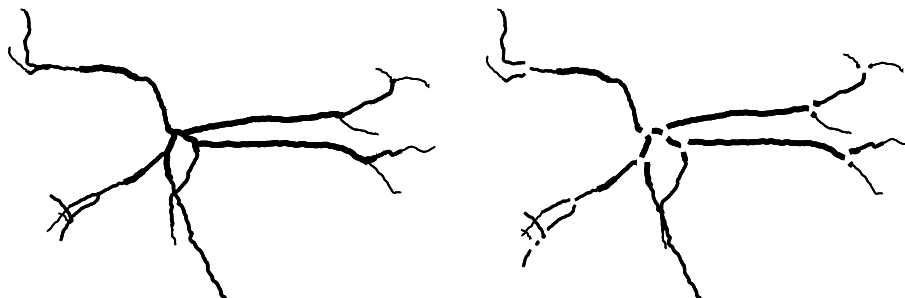
c_j membrane capacitance of compartment j

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

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

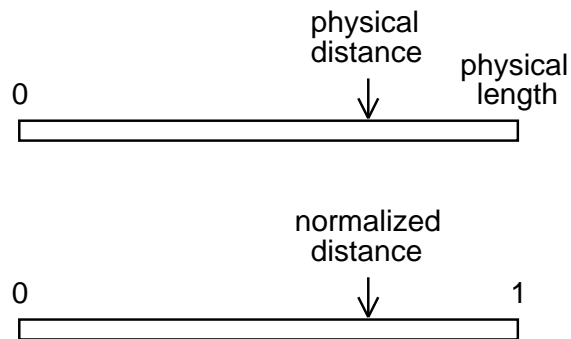
Name	Meaning	Units
diam	diameter	[μm]
cm	specific membrane capacitance	[$\mu\text{f}/\text{cm}^2$]
g_pas	specific conductance of the pas mechanism	[siemens/ cm^2]
v	membrane potential	[mV]

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

nseg=1 

nseg=2 

nseg=3 

Example: axon nseg = 3

To test spatial resolution

forall nseg = nseg*3

and repeat the simulation

Units

Category	Variable	Units
Time	t	[ms]
Voltage	v	[mV]
Current	i	[mA/cm ²] (distributed) [nA] (point process)
Concentration	na i etc.	[mM]
Specific capacitance	cm	[μ f/cm ²]
Length	diam, L	[μ m]
Conductance	g	[S/cm ²] (distributed) [μ S] (point process)
Cytoplasmic resistivity	Ra	[Ω cm]
Resistance	ri	[10 ⁶ Ω]

Physical System



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

Model

Hodgkin-Huxley cable equations

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

$$\begin{aligned} \frac{dm}{dt} &= -\alpha_m m + \beta_m \cdot (1 - m) & \alpha_m &= \frac{.1(V+40)}{1 - e^{-.1(V+40)}} & \beta_m &= 4e^{-(V+65)/18} \\ \frac{dh}{dt} &= -\alpha_h h + \beta_h \cdot (1 - h) & \alpha_h &= .07e^{-.05(V+65)} & \beta_h &= \frac{1}{1 + e^{-.1(V+35)}} \\ \frac{dn}{dt} &= -\alpha_n n + \beta_n \cdot (1 - n) & \alpha_n &= \frac{.01(V+55)}{1 - e^{-1(V+55)}} & \beta_n &= .125e^{-(V+65)/80} \end{aligned}$$

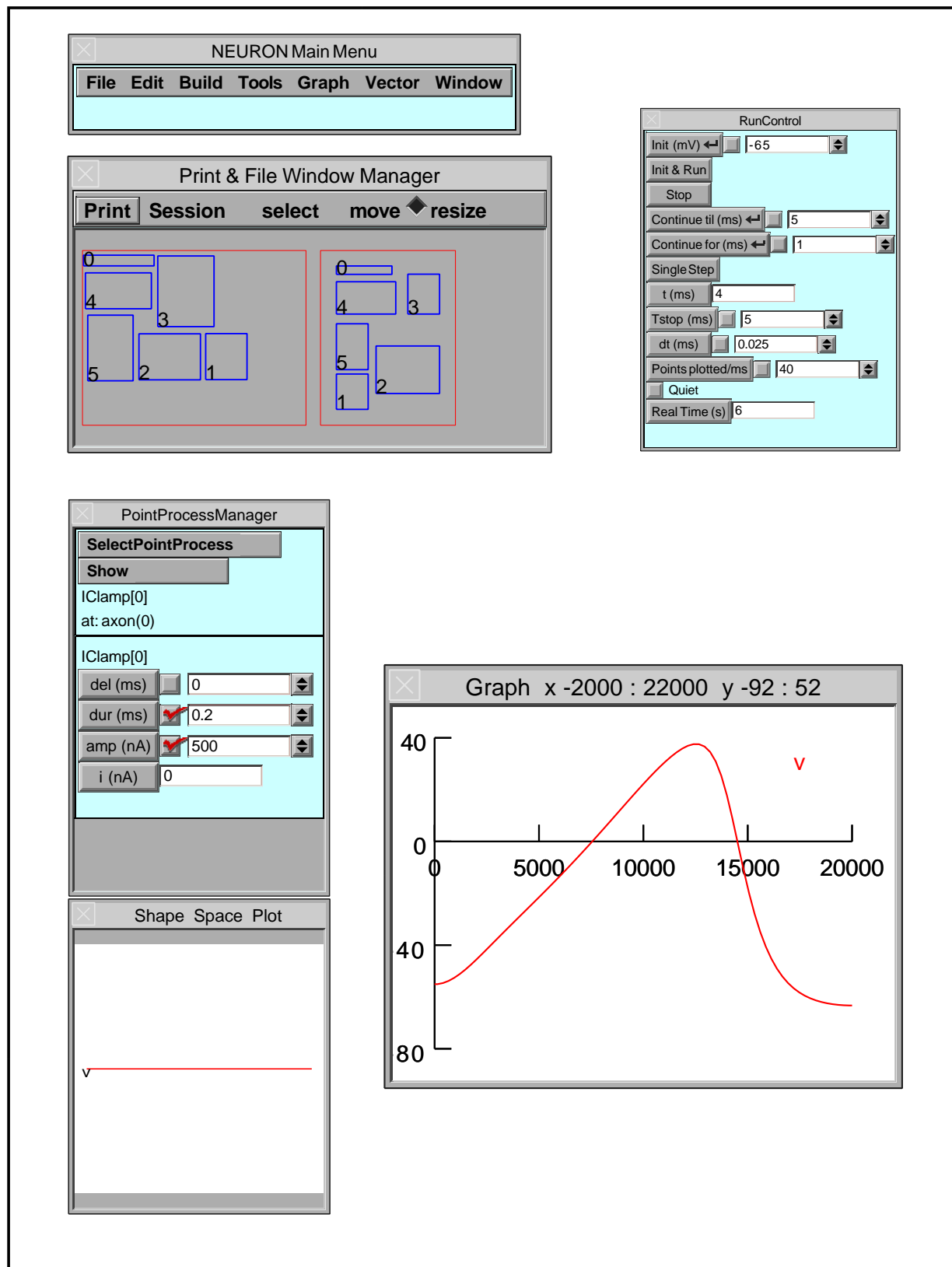
Simulation

Representation

```
create axon
axon {
    nseg = 50
    diam = 100
    L = 20000
    insert hh
}
```

Run NEURON with above spec.

Exercises



Example: Ball & Stick Model

Physical system: anatomically complex cell

Conceptual model: "ball and stick" model

Computational model: soma + dendritic cylinder

Project goals:

- Create model and custom GUI from scratch
- 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

Creating and using NEURON models

Use hoc, Python, and/or GUI to specify:

- Biological properties--anatomy, biophysics
- Instrumentation--signal sources and recording
- User interface--parameter panels, graphs
- Simulation control--dt, tstop, integration method

Hint: keep these separate from each other
for maximum clarity and to save effort

Verify:

- Close match to conceptual model?
- Numerical accuracy adequate?
(spatial grid, integration time step or error criterion)

Specifying biological properties

Topology (branching pattern)

Geometry (diameter, length)
and

Biophysics (membrane capacitance,
ion channels, pumps . . .)

Connections between cells
(synapses, gap junctions)

. . . and anything else that makes sense . . .

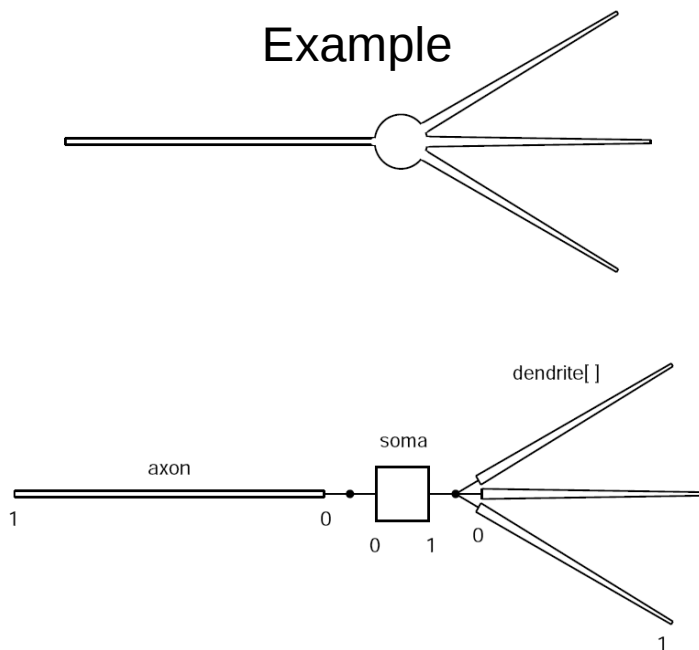
Biological properties: topology

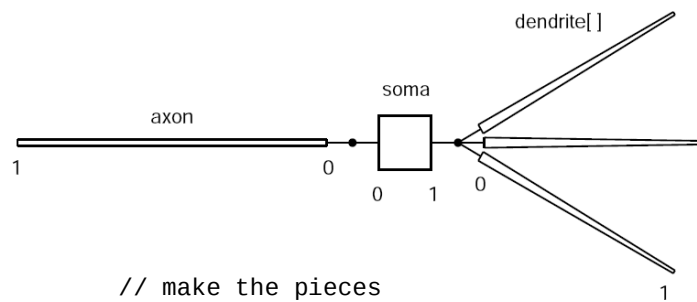
Make the pieces (sections)
create

Specify the default section
access

Assemble the pieces
connect

Example





```
// make the pieces
create soma, axon, dendrite[3]

// specify default section
access soma

// assemble them
connect axon(0), soma(0)
for i=0,2 {
  connect dendrite[i](0), soma(1)
}
```

Biological properties: geometry and biophysics

Compartmentalization

nseg

Geometry

L, diam

Biophysical properties

Density mechanisms: insert t

Examples: ion channels distributed
over the cell surface, pumps,
ion accumulation, buffers

```

soma {
  nseg = 1
  L = 50      // [um] length
  diam = 50   // [um] diameter
  insert hh   // Hodgkin-Huxley currents
}

axon {
  nseg = 21   // odd so a node is at 0.5
  L = 1000
  diam = 1
  insert hh
}

for i=0,2 dendrite[i] {
  nseg = 5
  L = 200
  diam(0:1) = 10:3 // taper
  insert pas       // passive membrane
}

forall Ra = 60 // [ohm cm]

```

Range variables

Vary continuously in space along the length of a section

Examples: v, cm, diam

Section variables

Pertain to an entire section

Examples: Ra (cytoplasmic resistivity), L, nseg

Global variables

Same across all sections

Examples: celsius, t and dt (fixed time step integration)

Instrumentation

This model needs an electrode at the soma to inject stimulating current.

Examples of "point processes":
current clamp, voltage clamp, synapse

Object syntax

```
objref stim
// attach to middle of soma
soma stim = new IClamp(0.5)

stim.del = 1    // [ms] delay
stim.dur = 0.1  // [ms] duration
stim.amp = 60   // [nA] amplitude
```

Simulation control

```
finitialize(-65) // initialize v, state variables, time
// a minimalistic approach for fixed dt simulations

dt = 0.025 // [ms] integration time step
tstop = 5 // [ms]

proc simulate() {
  // show start time & initial somatic v
  print t, v(0.5) // soma is default section

  while (t < tstop) {
    fadvance() // advance solution by dt
    // function calls to save or plot results, e.g.
    print t, v(0.5)
    // statements to change model parameters
  }
}
```


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)) {
    ...
}
```

Density mechanism

Point Process

NMODL

```

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)
}

```

```

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

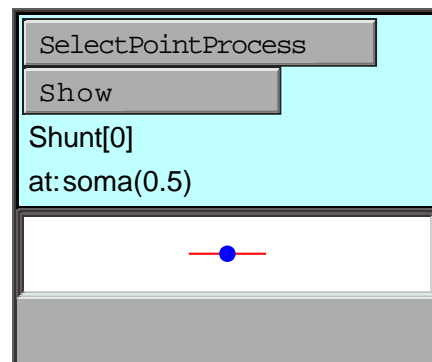
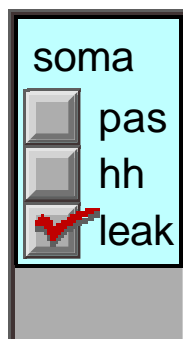
PARAMETER {
  r = 1 (gigaohm) <1e-9,1e9>
  e = 0 (millivolt)
}

ASSIGNED {
  i (nanoamp)
  v (millivolt)
}

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

```

GUI



Interpreter

```

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

```

```

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

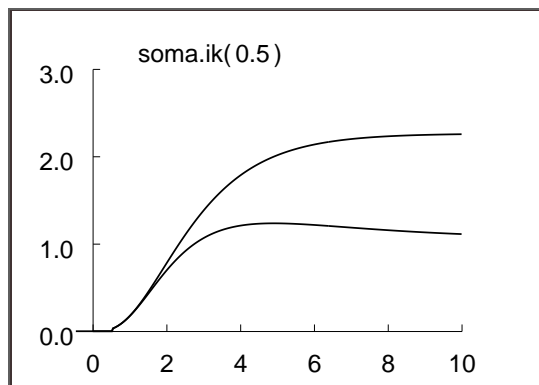
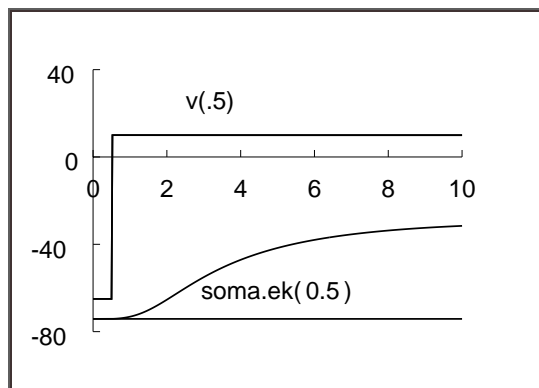
```

Ion Channel

```

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

```

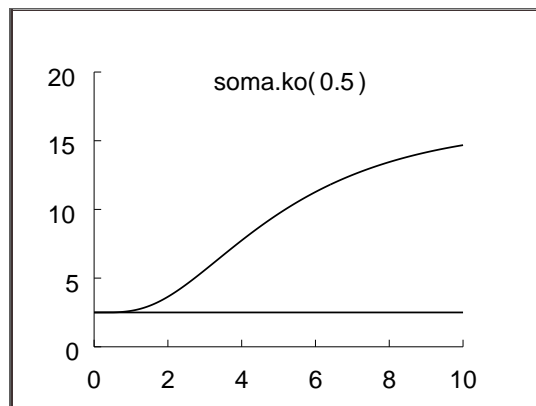


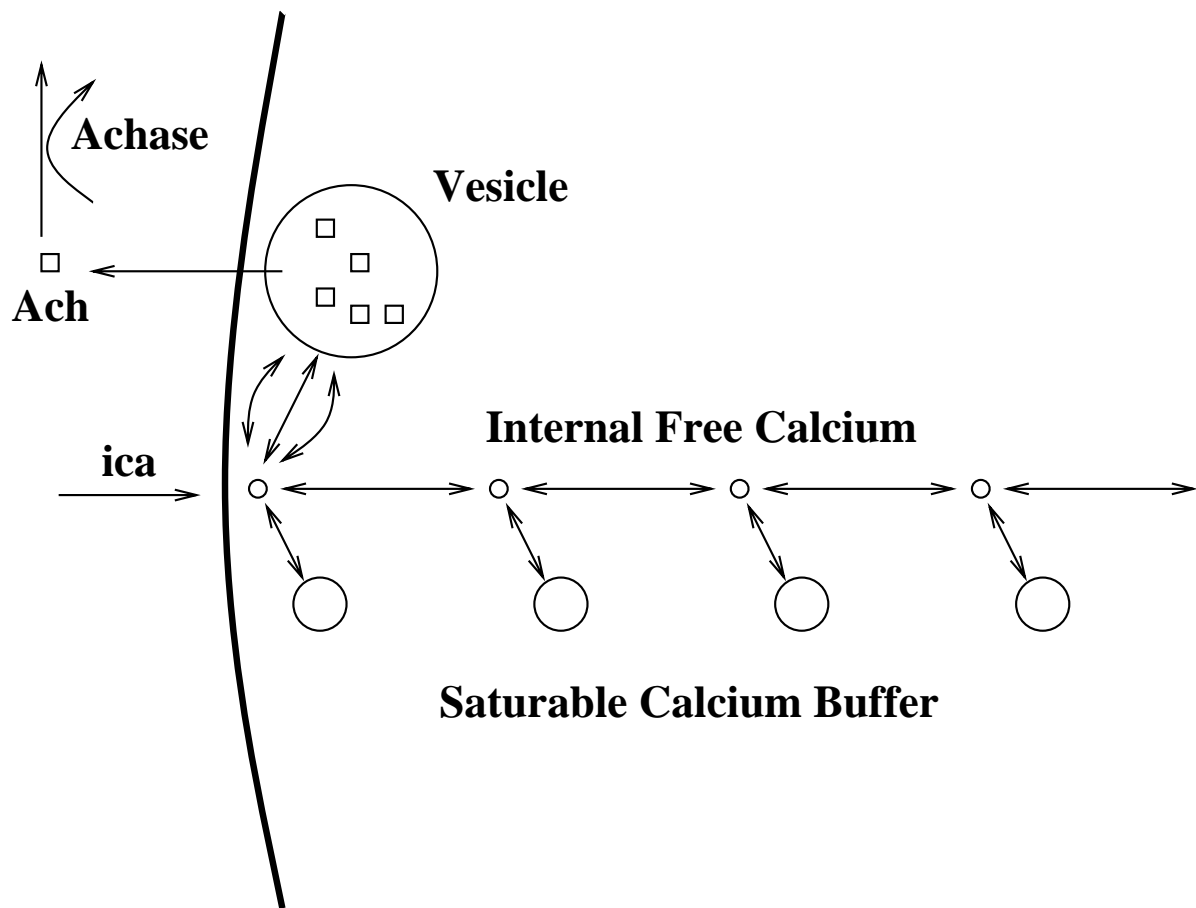
Ion Accumulation

```

NEURON {
  USEION k READ ik WRITE ko
}
BREAKPOINT {
  SOLVE state METHOD cnexp
}
DERIVATIVE state {
  ko' = ik/fhspace/F*(1e8)
  + k*(kbath - ko)
}

```





```

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.
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?

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

UNIX

In the directory containing the desired mod files:

```
nrnivmodl  
nrngui
```

Select NEURONMainMenu/Build/singlecompartment.

MSWIN

Launch mknrndll from the icon in the NEURON program group.

Navigate to the directory containing the desired mod files.
Select "Make nrnmech.dll".

Launch nrngui from the icon in the NEURON program group.

Select NEURONMainMenu/File/RecentDir to change the working dir and load nrnmech.dll.
Select NEURONMainMenu/Build/singlecompartment.

single.hoc

```
load_file("stdgui.hoc")  
create soma  
access soma  
// area 100 um2 means mA/cm2 identical to nA  
{diam=10 L=10/PI}
```

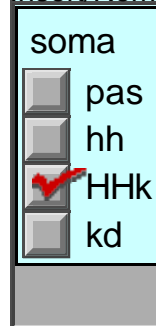
: Hodgkin - Huxley k channel

```

NEURON {
  SUFFIX HHk
  USEION k READ ek WRITE ik
  RANGE gkbar, ik, g
  GLOBAL inf, tau
}
UNITS {
  (mA) = (milliamp)
  (mV) = (millivolt)
}
PARAMETER {
  gkbar= 0.036 (mho/cm2) <0,1e9>
}
STATE {
  n
}
ASSIGNED {
  v (mV)
  ek (mV)
  celsius (degC)
  ik (mA/cm2)
  inf
  tau (ms)
  g (mho/cm2)
}
INITIAL {
  rate(v)
  n = inf
}
BREAKPOINT {
  SOLVE states METHOD cnexp
  g = gkbar*n*n*n*n
  ik = g*(v - ek)
}
DERIVATIVE states {
  rate(v)
  n' =(inf - n)/tau
}
FUNCTION alp(v(mV))/(ms) { LOCAL q10
  v = -v - 65
  q10 = 3^((celsius - 6.3)/10 (degC))
  alp = q10 * 0.01(/ms-mV)*expM1(v + 10, 10(mV))
}
FUNCTION bet(v(mV))/(ms) { LOCAL q10
  v = -v - 65
  q10 = 3^((celsius - 6.3)/10 (degC))
  bet = q10 * 0.125(/ms)*exp(v/80(mV))
}
FUNCTION expM1(x (mV),y (mV)) (mV) {
  if (fabs(x/y) < 1e-6) {
    expM1 = y*(1 - x/y/2)
  }else{
    expM1 = x/(exp(x/y) - 1)
  }
}
PROCEDURE rate(v (mV)) {LOCAL a, b
  TABLE inf, tau DEPEND celsius FROM -100 TO 100 WITH 200
  a = alp(v)  b=bet(v)
  tau = 1/(a + b)
  inf = a/(a + b)
}

```

Insert/Remove Mechanisms



soma(0 - 1) (Parameters)

soma(0 - 1) (Parameters)

nseg = 1

L (um) 3.1831

Ra (ohm-cm) 35.4

diam (um) 10

cm (uF/cm2) 1

gkbar_HHk(mho/cm2) 0.036

ek (mV) -77

soma(0.5) (States)

soma(0.5) (States)

v -65

n_HHk 0

HHk (Globals)

inf_HHk 0.9725

tau_HHk(ms) 0.92617

usetable_HHk 1

soma(0.5) (Assigned)

soma(0.5) (Assigned)

v -65

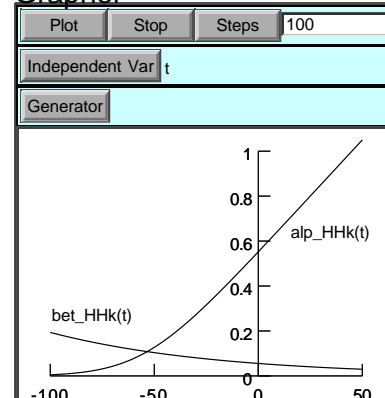
i_cap 0

ik_HHk(mA/cm2) 0

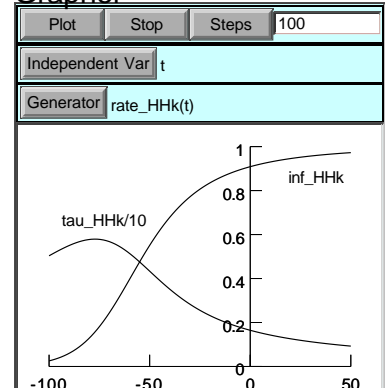
g_HHk(mho/cm2) 0

ik (mA/cm2) 0

Grapher



Grapher

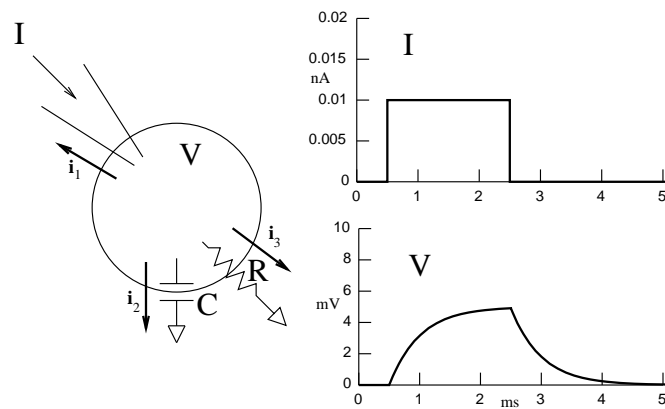


Compartmental Modeling

Not much mathematics required.

Good judgment essential!

1



$$i_1 + i_2 + i_3 = 0$$

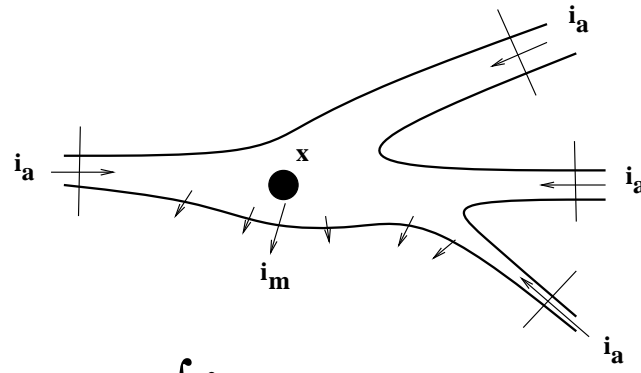
$$i_1 = -I$$

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

$$i_3 = V / R$$

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

2

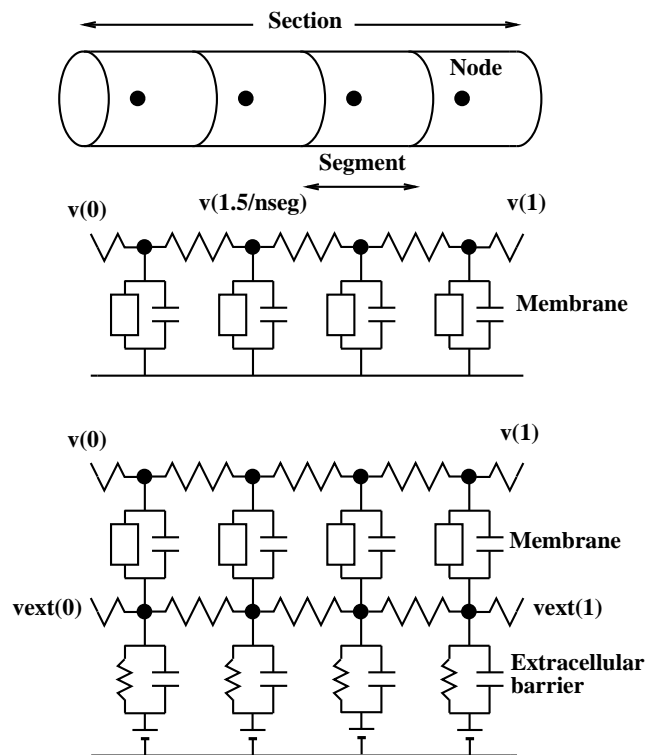


The diagram shows a neuron segment with a central node 'x'. Axial currents i_a flow into and out of the segment at its boundaries. Membrane currents i_m flow out of the segment across its surface. Below the diagram are two equations:

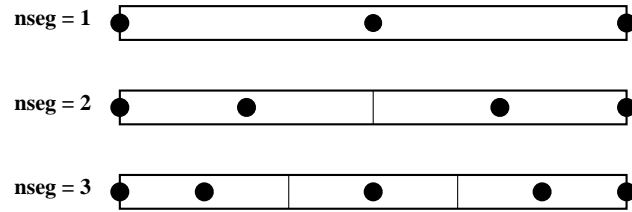
$$\int i_m = \sum i_a$$

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

3



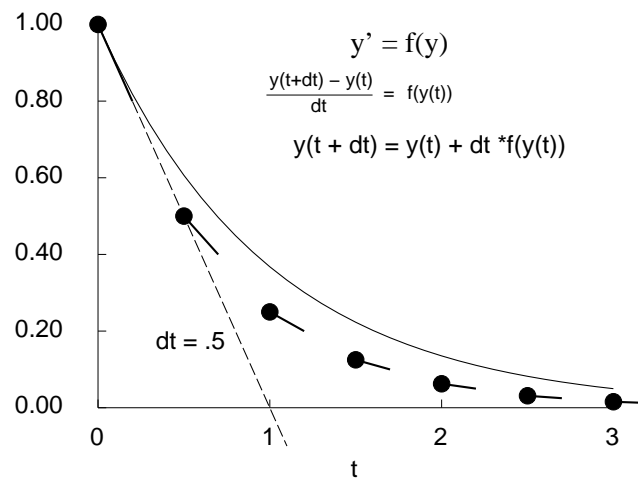
4



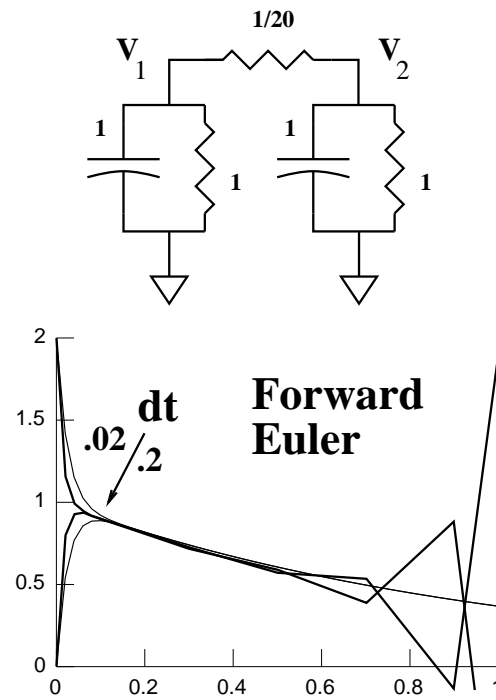
forall nseg *= 3

5

Forward Euler

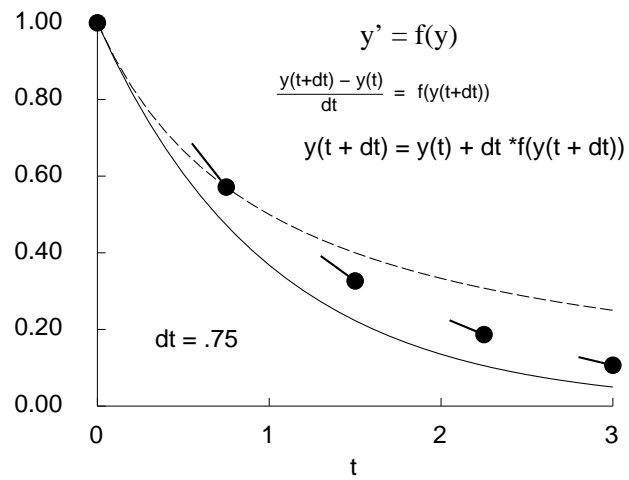


6

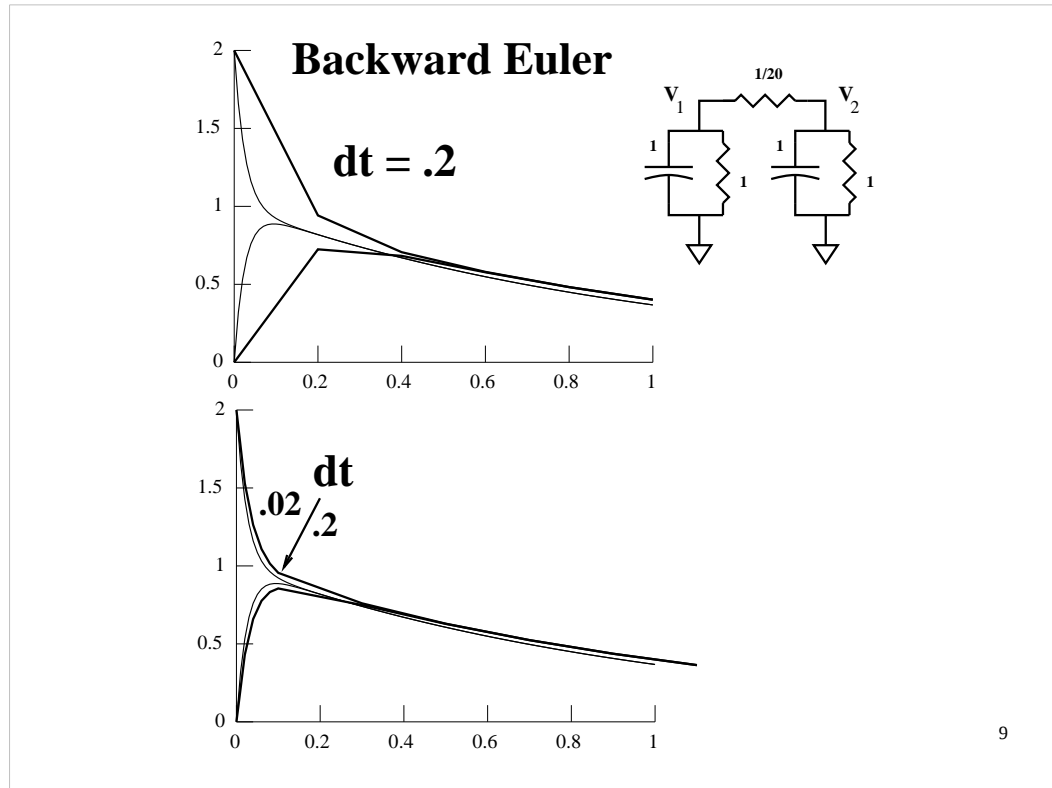


7

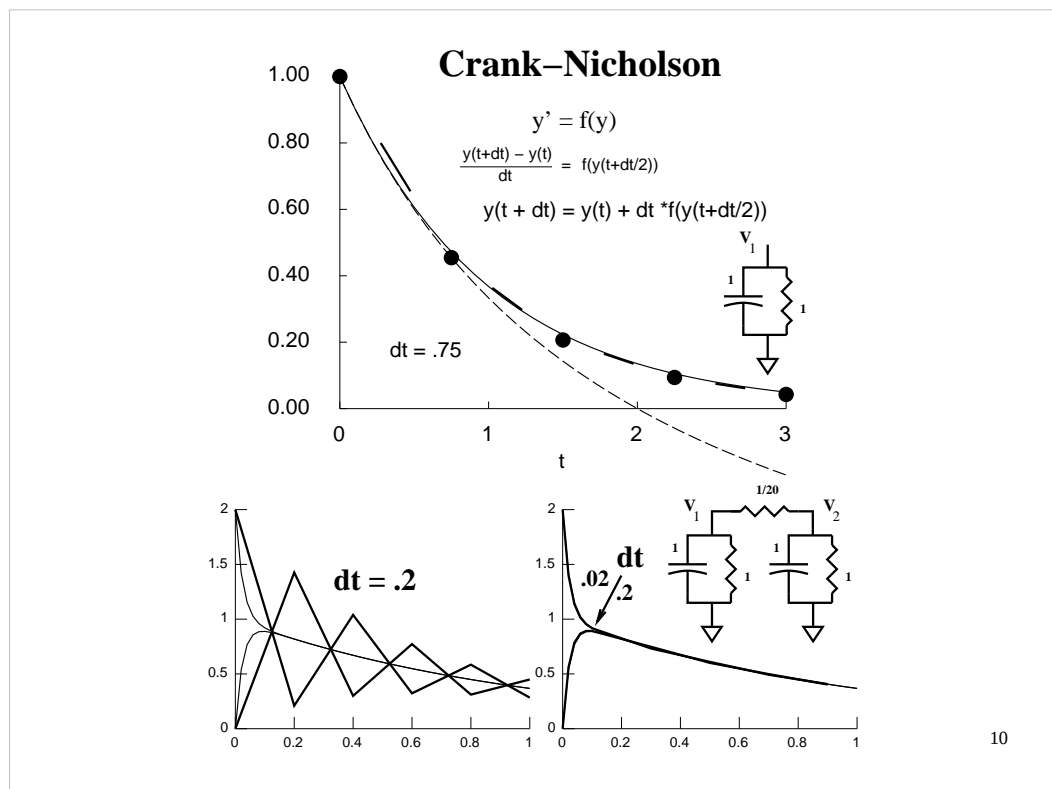
Backward Euler



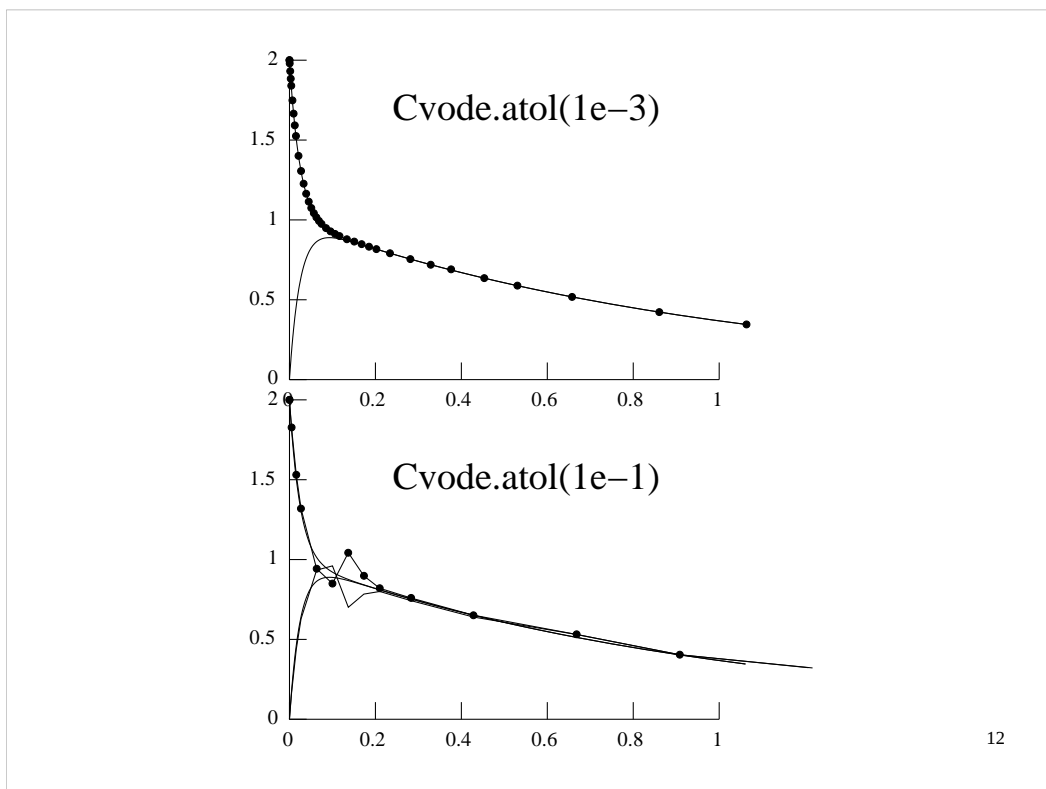
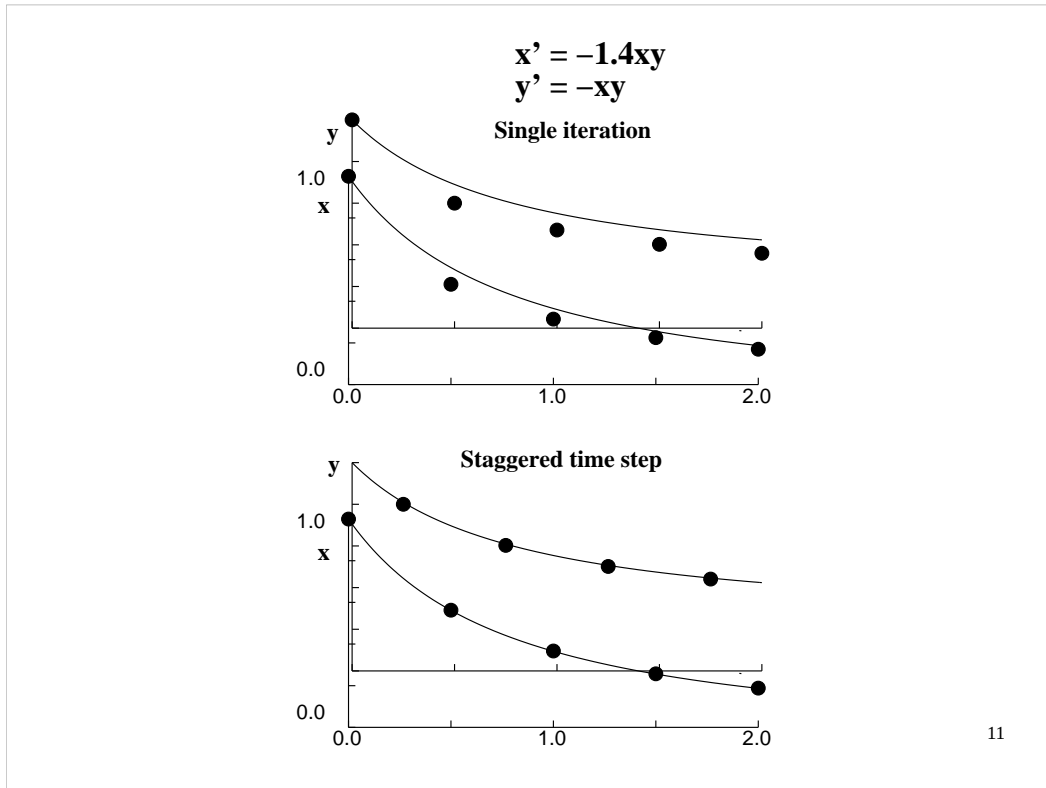
8

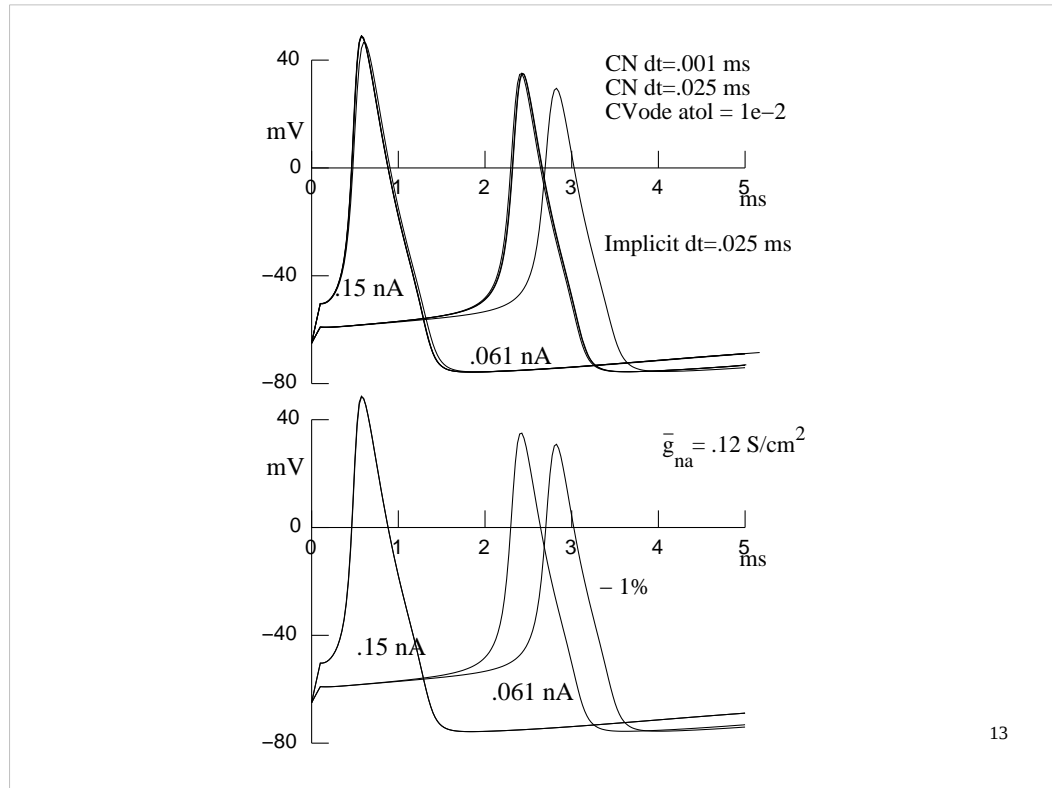


9

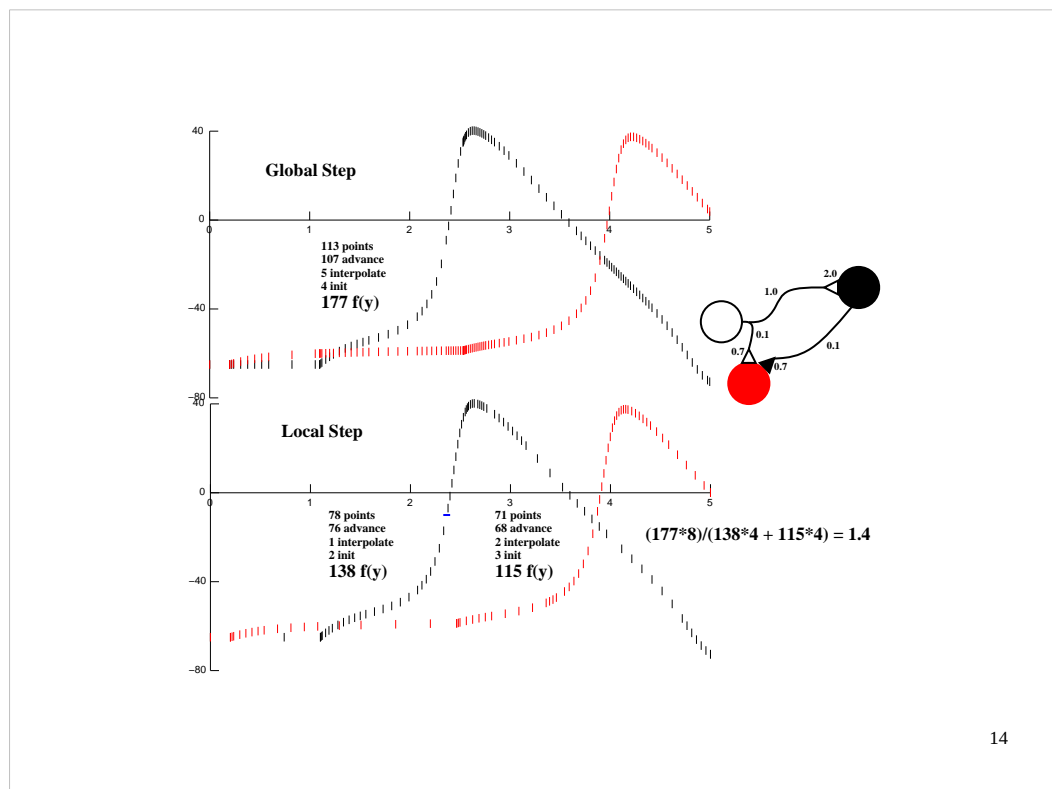


10





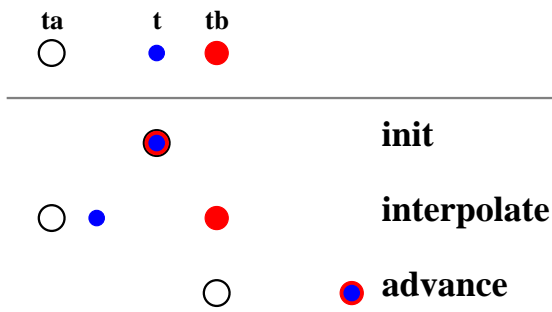
13



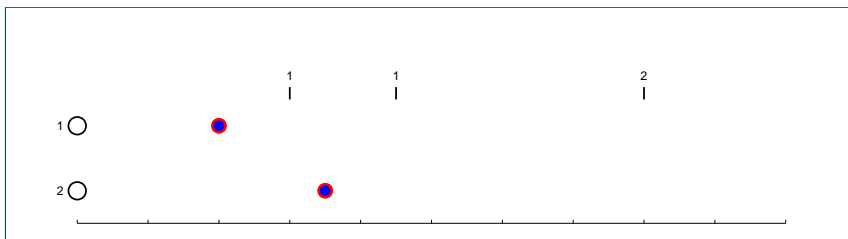
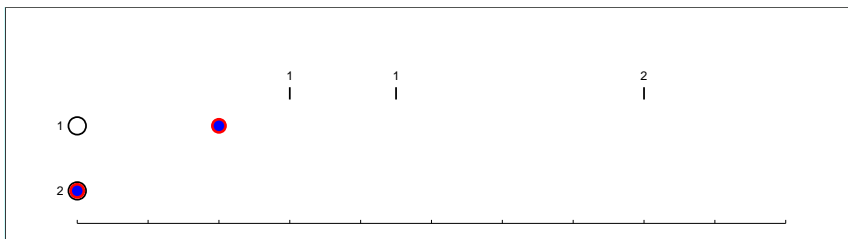
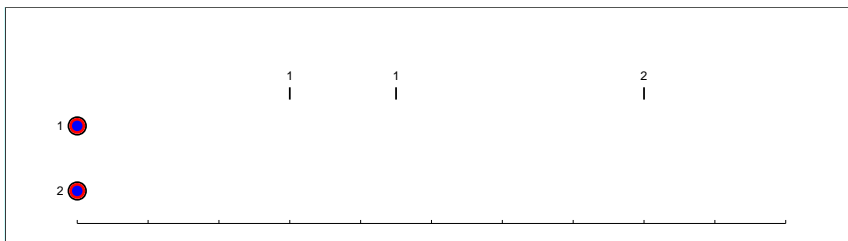
14

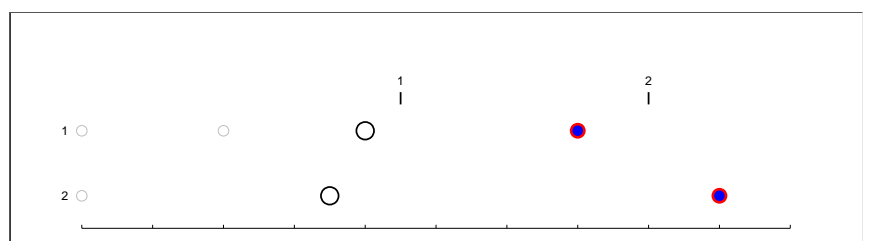
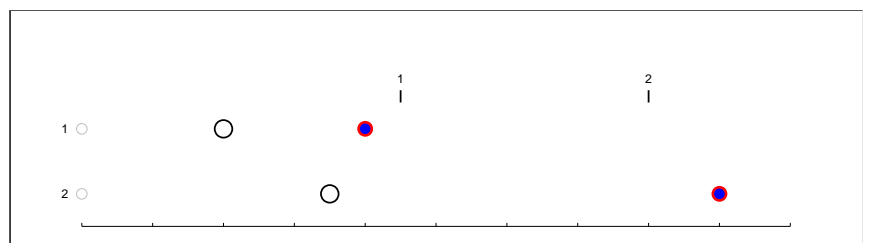
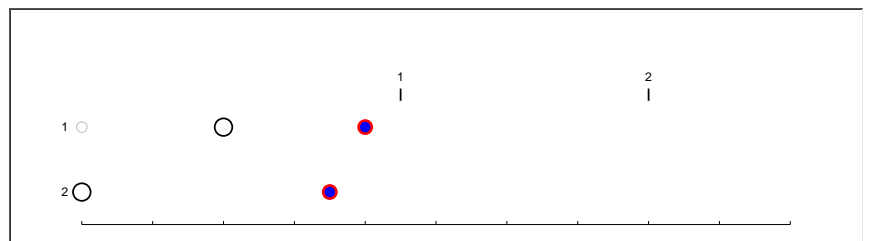
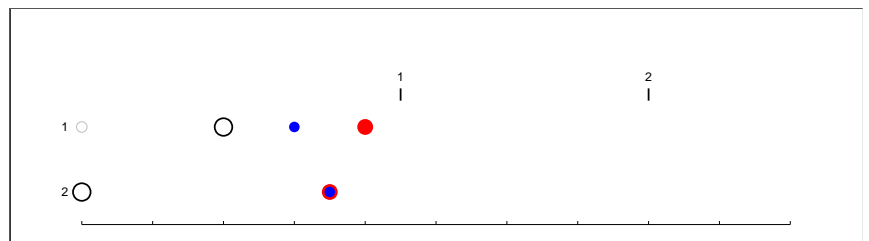
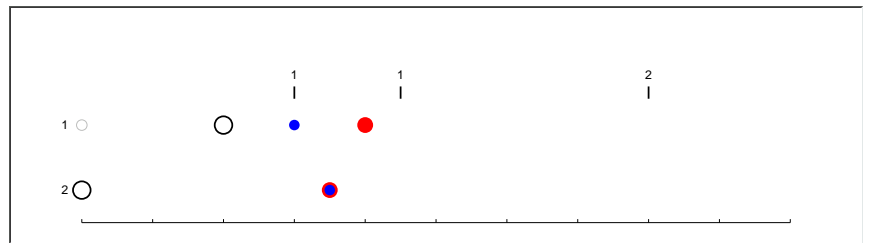
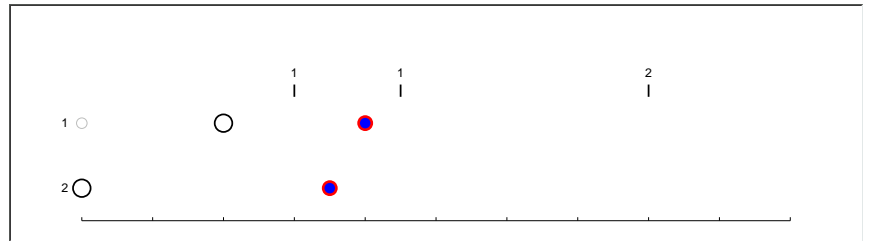
One integrator instance per cell

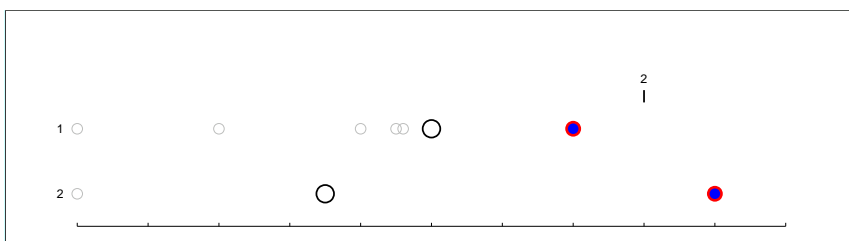
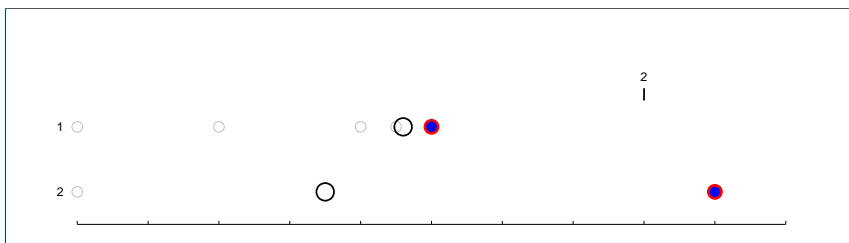
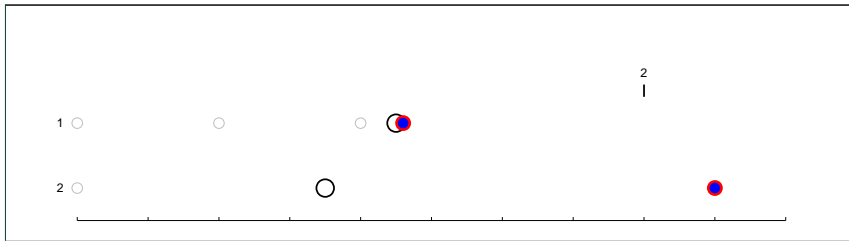
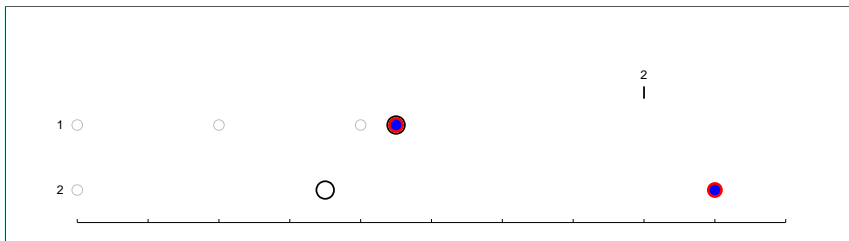
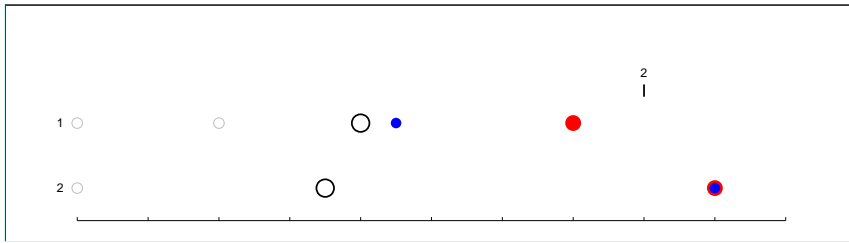
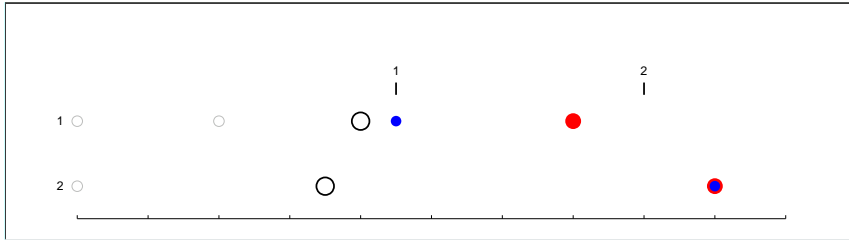
$$\forall i, j: ta_i \leq tb_j$$

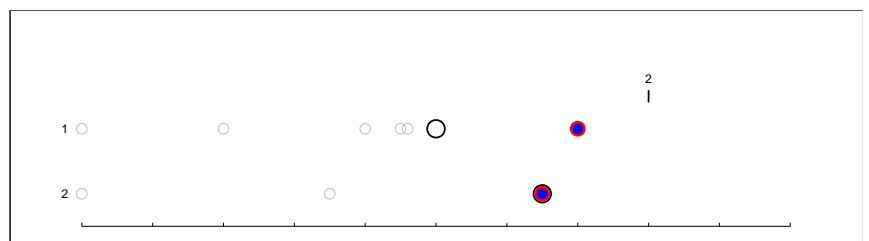
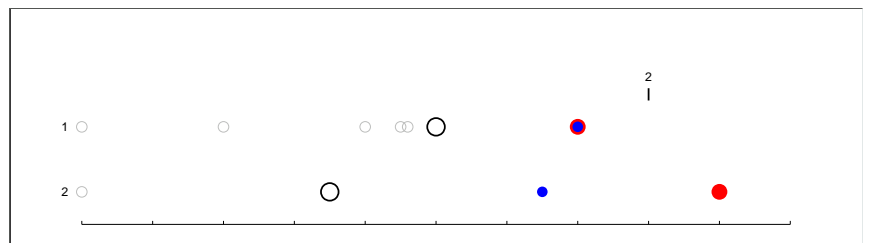
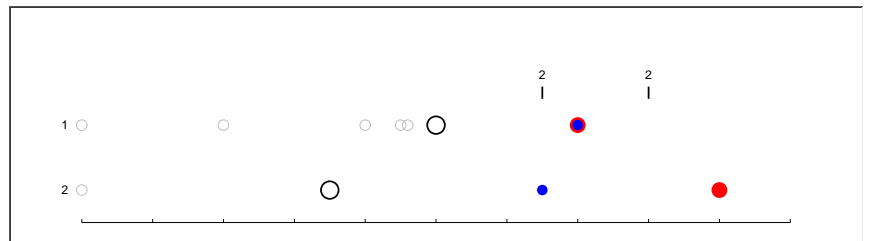
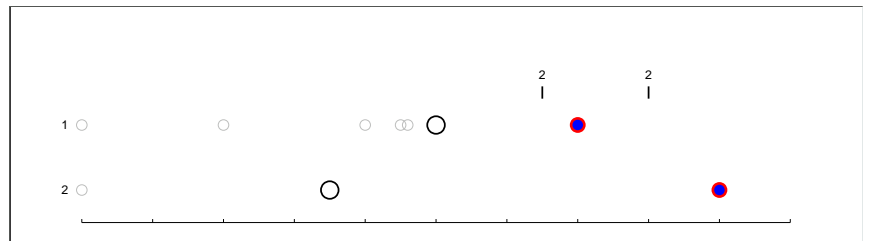
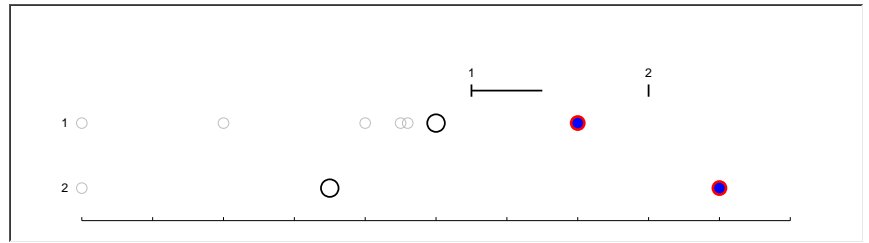
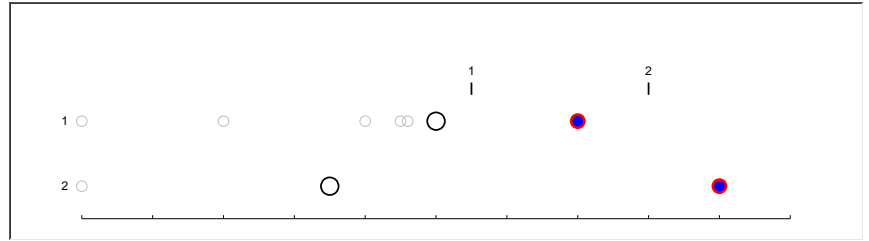


15









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."

7

<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. About [model sharing in general](#) and [ModelDB in particular](#).

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

Find models by
[* model name](#)
[* first author](#)
[* each author](#)
[* Region/circuits](#)

Find models for
[* Cell type](#)
[* Current](#)
[* Receptor](#)
[* Transmitters](#)
[* Topic](#)
[* Simulators](#)
[* Methods](#)

Find models of
[* Networks](#)
[* Neurons](#)
[Synapses](#)
[* 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](#)

[Search for models containing specific combinations of keywords](#)

[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](#)

8

Accommodates models from a wide range of simulation environments

552 entries as of 6/12/2010

BioPAX	1	KInNeSS	2	Q/Quick/Turbo Basic	2
Brian	4	MATLAB	101	QuB	1
C / C++	53	MCell	1	R	1
Content	1	MOOSE / PyMOOSE	1	SABER	1
CSIM	4	MVASpike	1	SBML	1
CalC	8	MadSim	1	SNNAP	21
Catacomb	2	NCS	1	SciLab	3
CellExcite	2	NEST	2	Simulink	6
CellML	1	NEURONPM	2	Sspice	1
Chemesis	2	Network	1	Topographica	1
Dynamics Solver	1	Neuron	274	Virtual Cell	3
Emergent/PDP++	3	Octave	1	XML	3
FORTTRAN	5	PCSIM	1	XPP	56
GNU/Next/Openstep	1	PSpice	2	neuroConstruct	1
Genesis	20	Pascal/Delphi	3	parplex	2
IGOR Pro	3	PyNN	2		
Java	7	Python	6		

9

Search results

Models by moore

1. [MEG of Somatosensory Neocortex \(Jones et al. 2007\)](#)
Jones SR, Pritchett DL, Stufflebeam SM, Hamalainen M, Moore CI (2007) Neural correlates of tactile detection: a combined magnetoencephalography and biophysically based computational modeling study. *J Neurosci* 27:10751-64 [[PubMed](#)]
2. [Nerve terminal currents at lizard neuromuscular junction \(Lindgren, Moore 1989\)](#)
Lindgren CA, Moore JW (1989) Identification of ionic currents at presynaptic nerve endings of the lizard. *J Physiol* 414:201-22 [[PubMed](#)]
3. [Presynaptic calcium dynamics at neuromuscular junction \(Stockbridge, Moore 1984\)](#)
Stockbridge N, Moore JW (1984) Dynamics of intracellular calcium and its possible relationship to phasic transmitter release and facilitation at the frog neuromuscular junction. *J Neurosci* 4:803-11 [[PubMed](#)]
4. [Site of impulse initiation in a neuron \(Moore et al 1983\)](#)
Moore JW, Stockbridge N, Westerfield M (1983) On the site of impulse initiation in a neurone. *J Physiol* 336:301-11 [[PubMed](#)]
5. [Current flow during PAP in squid axon at diameter change \(Joyner et al 1980\)](#)
Joyner RW, Westerfield M, Moore JW (1980) Effects of cellular geometry on current flow during a propagated action potential. *Biophys J* 31:183-94 [[PubMed](#)]
6. [Conduction in uniform myelinated axons \(Moore et al 1978\)](#)
Moore JW, Joyner RW, Brill MH, Waxman SD, Najjar-Joa M (1978) Simulations of conduction in uniform myelinated fibers. Relative sensitivity to changes in nodal and internodal parameters. *Biophys J* 21:147-60 [[PubMed](#)]
7. [Temperature-Sensitive conduction at axon branch points \(Westerfield et al 1978\)](#)
Westerfield M, Joyner RW, Moore JW (1978) Temperature-sensitive conduction failure at axon branch points. *J Neurophysiol* 41:1-8 [[PubMed](#)]
8. [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](#)]

10

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* 336:301-11 [[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: [Spinal motor neuron](#);

Cell Type(s): [Na⁺](#) [K⁺](#);

Channel(s): [Na⁺](#) [K⁺](#);

Gap Junctions:

Receptor(s):

Gene(s):

Transmitter(s):

Simulation: [Neuron](#);

Environment:

Model Concept(s): [Action Potential Initiation](#); [Simplified Models](#);

Implementer(s): [Hines, Michael](#);

Search NeuronDB for information about: [Spinal motor neuron](#); [K⁺](#); [Na⁺](#);

Model files	Download zip file	Auto-launch	Help downloading and running models
<ul style="list-style-type: none"> moore83 README mosinit.hoc nit.hoc startses 	<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>		

11

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?

13

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

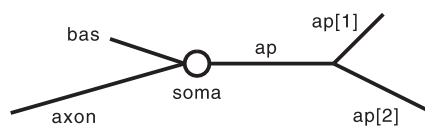
- Other tests

Spatially inhomogeneous parameters

Rules

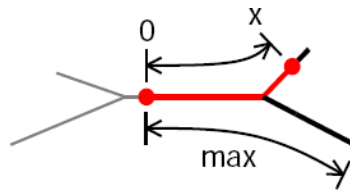
- None (arbitrary values)
- Constant over sets of sections
use SectionLists (CellBuilder Subsets)
- A function of position
hoc or ?

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

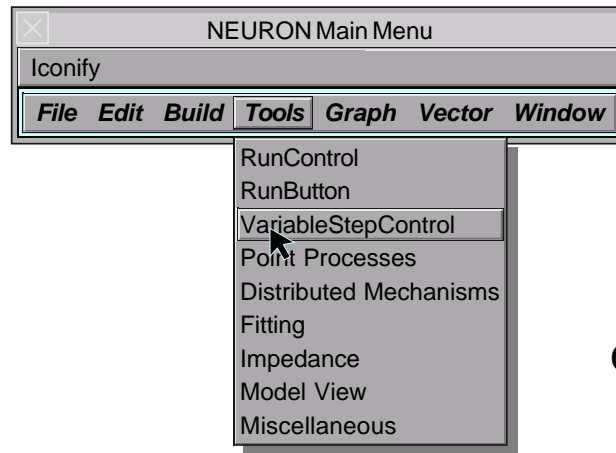
parameter that depends on distance

function f that governs the relationship

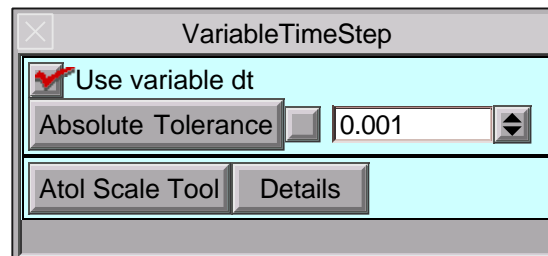
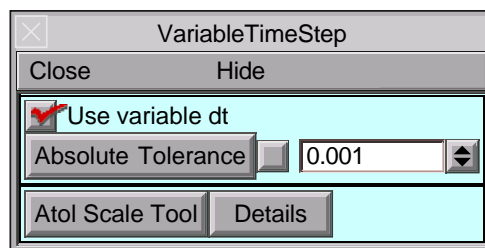
between the parameter and p

2. Verify the implementation

How? hoc or GUI (CellBuilder, Model View)



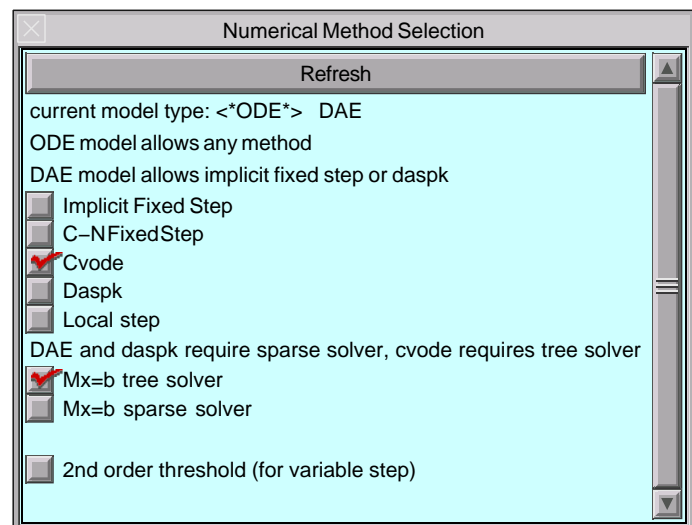
`cvar_active(1)`

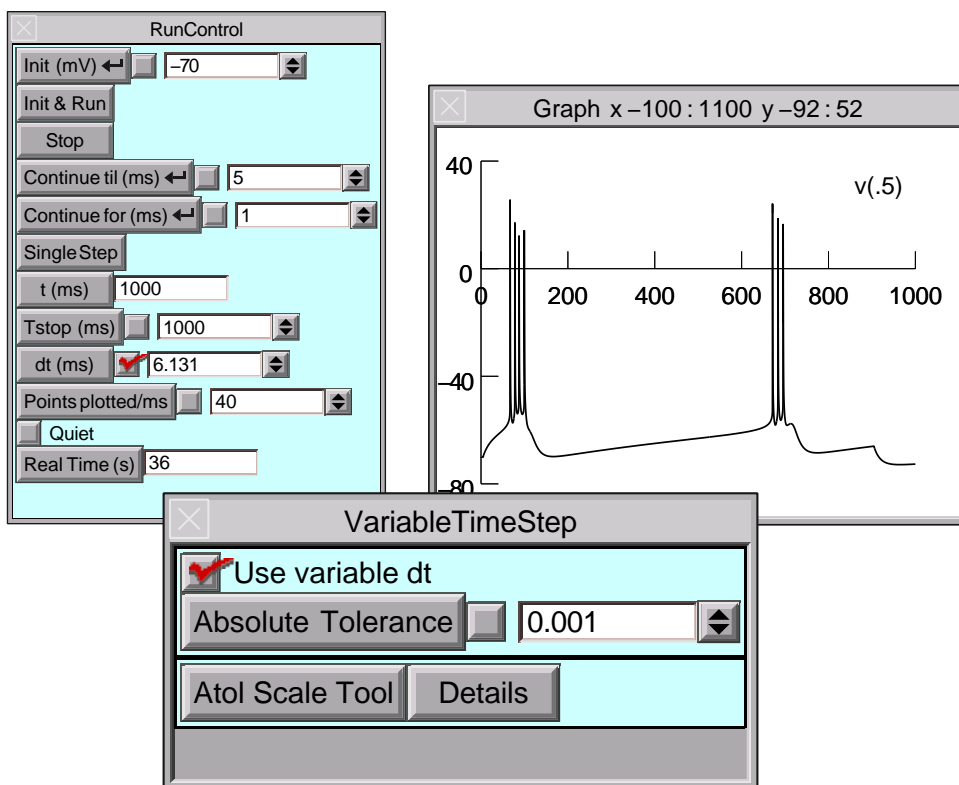
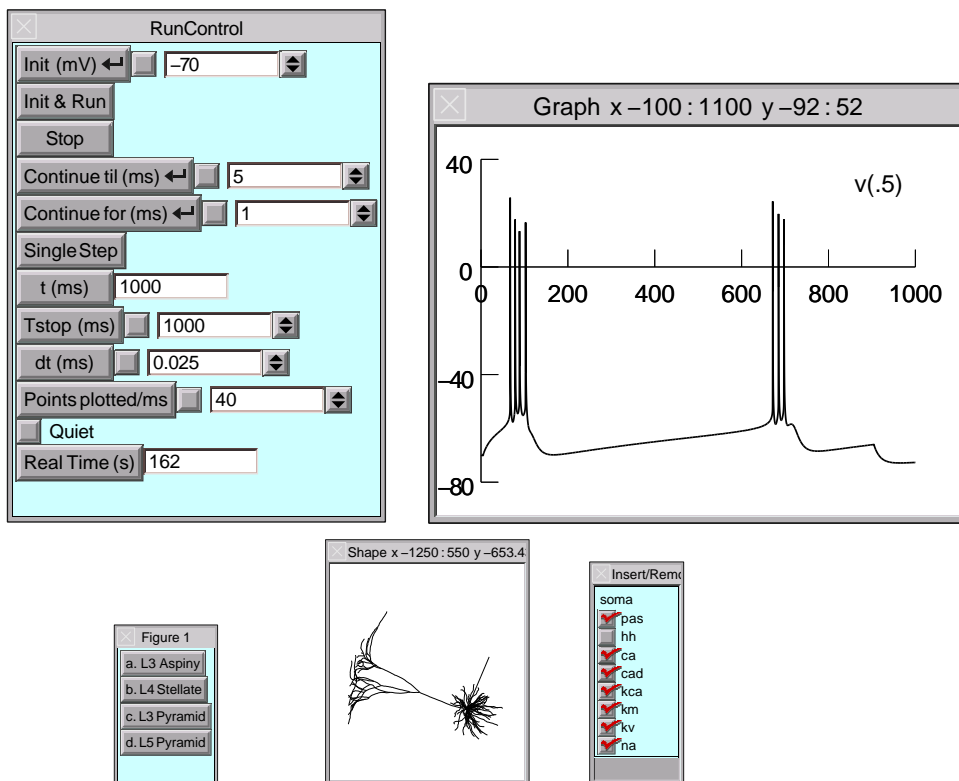


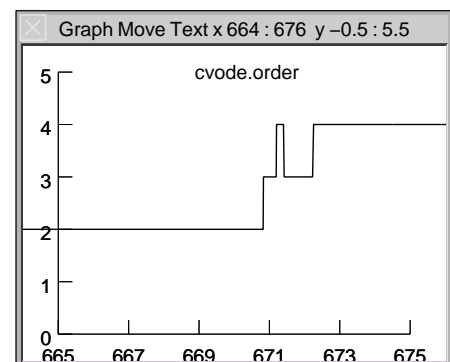
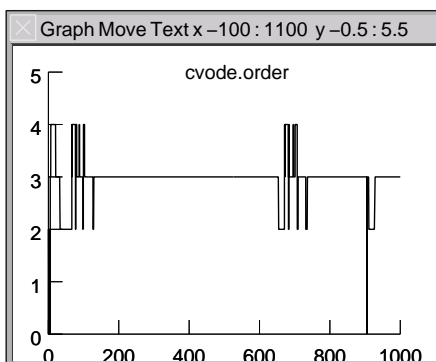
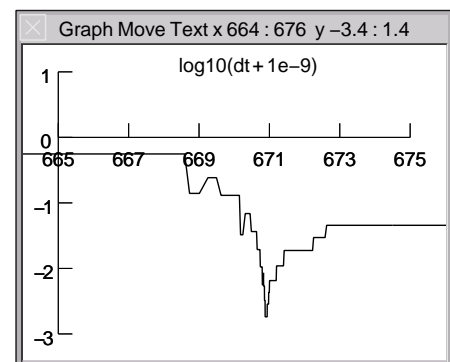
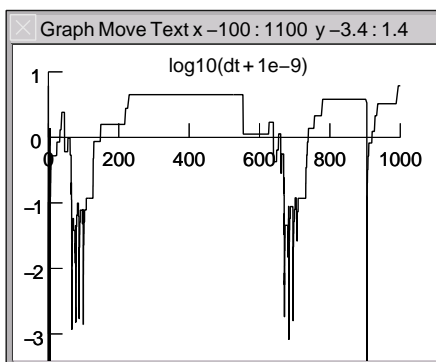
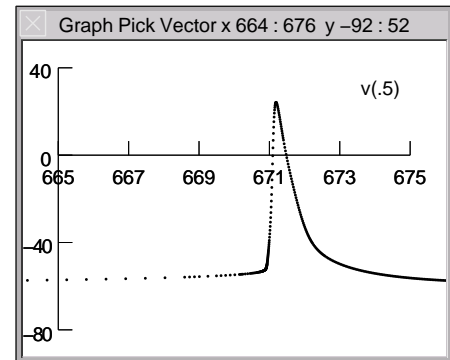
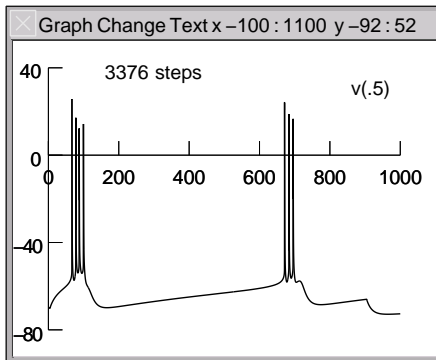
Absolute Tolerance Scale Factors

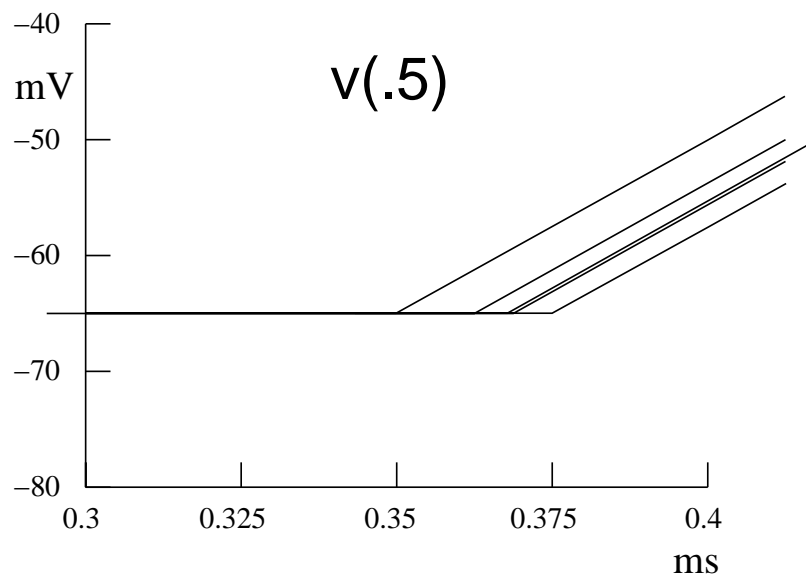
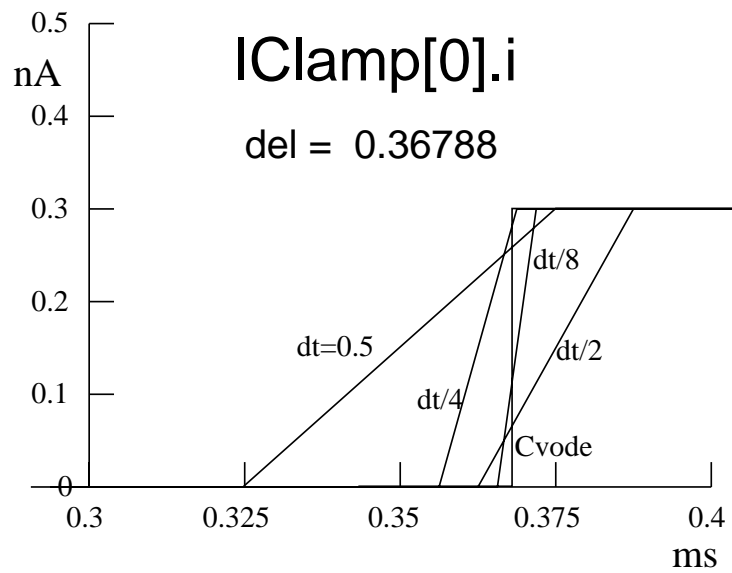
Analysis Run Rescale Original

	*10	/10	Hints
v	1	65	0
ca_cadifpmp	1e-06	3e-06	0
pump_cadifpmp	1e-15	1e-13	0
pumpca_cadifpmp	1e-15	3.6e-15	0
oca_cachan	1	0.053	0
n_HHk	1	0.32	0
m_HHna	1	0.053	0
h_HHna	1	0.6	0
Ves_trel	1	0.0004	0
B_trel	1	0	0
Ach_trel	1	0	0
X_trel	1	0	0









ModelDB: Model Information

<http://senselab.med.yale.edu/senselab/ModelDB/ShowModel.asp?m...>**Spinal Motor Neuron: McIntyre et al 2002**

Simulation of peripheral nervous system (PNS) myelinated axon. This model is described in detail in: McIntyre CC, Richardson AG, Grill WM. (2002)

Reference: McIntyre CC, Richardson AG, Grill WM (2002) Modeling the excitability of Mammalian nerve fibers: influence of afterpotentials on the recovery cycle. *J Neurophysiol* 87:995-1006 [[PubMed](#)]

Citations [Citation Browser](#)

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

Model Type: [Axon](#);

Cell Type(s): [Spinal motor neuron](#);

Channel(s): [I_{Na,p}](#); [I_{Na,t}](#); [I_K](#); [I_{Sodium}](#); [I_{Potassium}](#);

Receptor(s):

Transmitter(s):

Simulation Environment: [Neuron](#);

Model Concept(s): [Axonal Action Potentials](#); [Action Potentials](#);

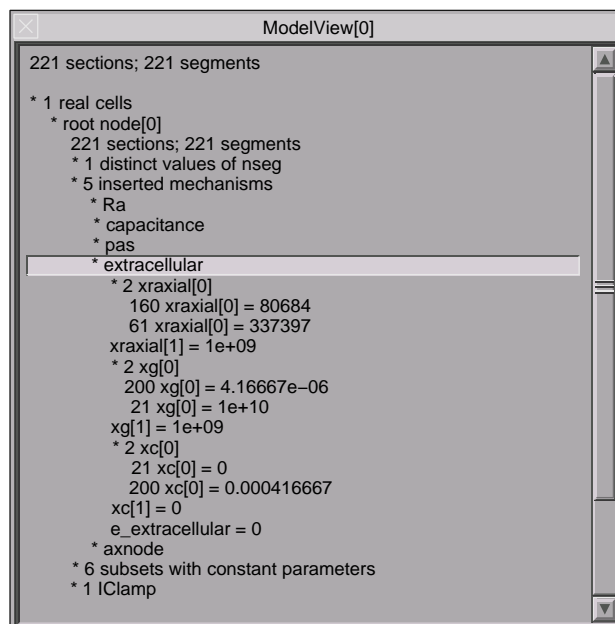
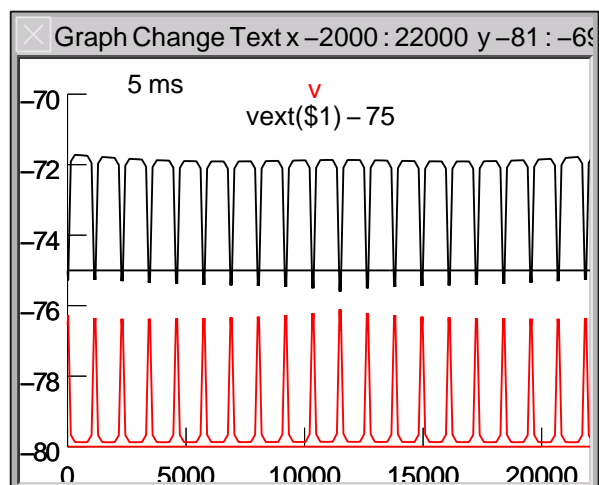
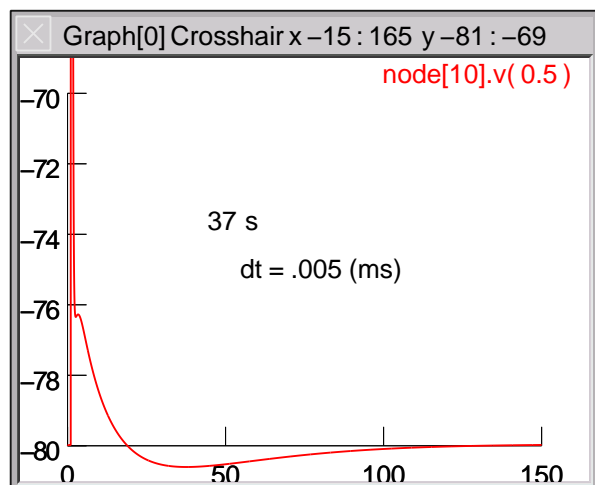
Implementer(s): [MacIntyre, CC](#);

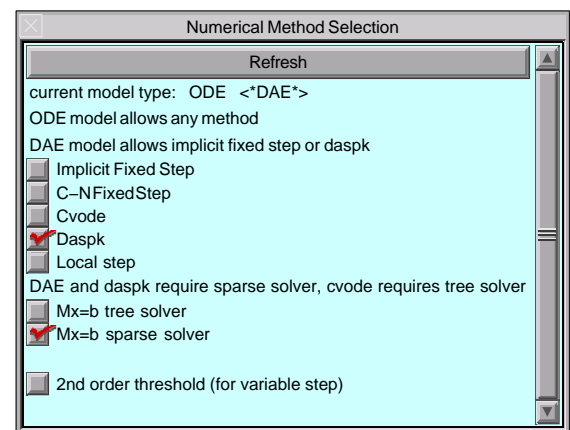
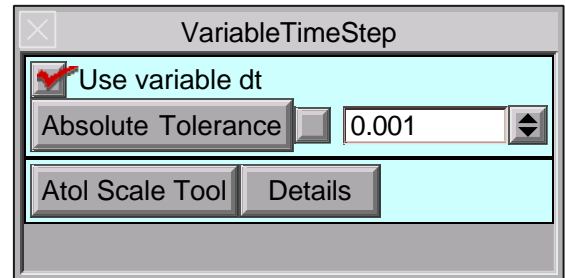
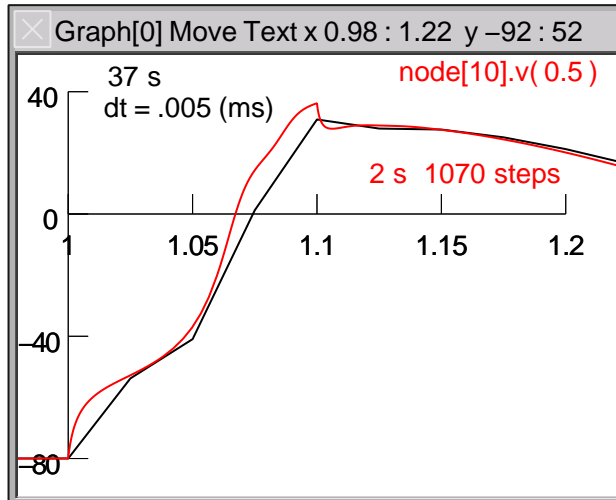
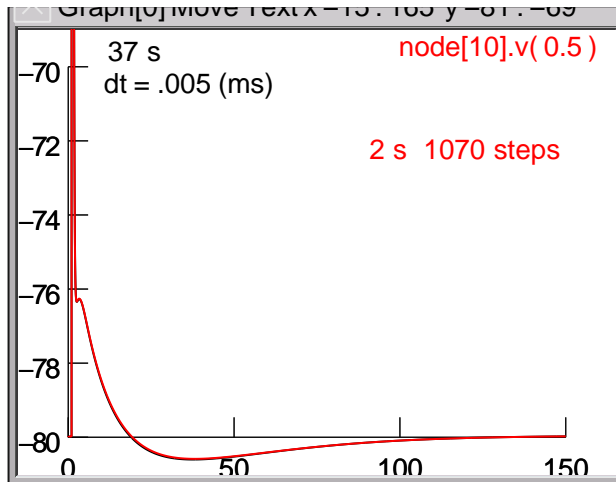
Search NeuronDB for information about: [Spinal motor neuron](#); [I_{Na,p}](#); [I_{Na,t}](#); [I_K](#); [I_{Sodium}](#); [I_{Potassium}](#);

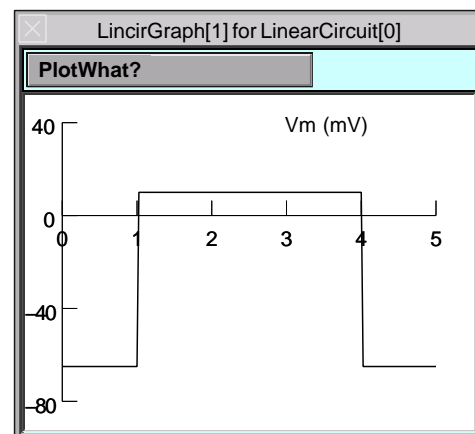
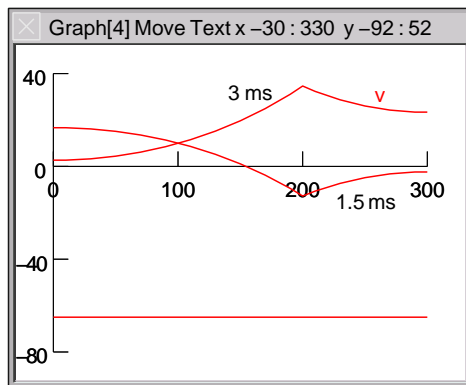
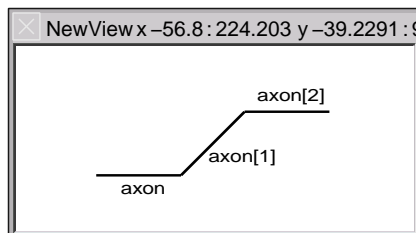
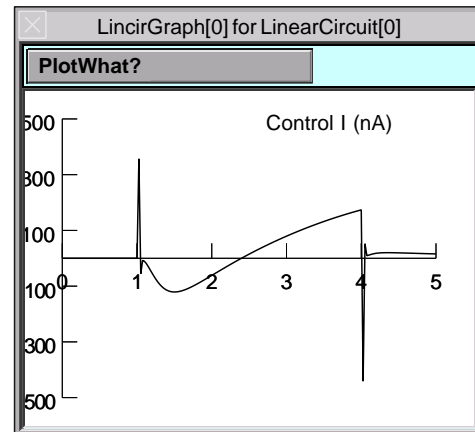
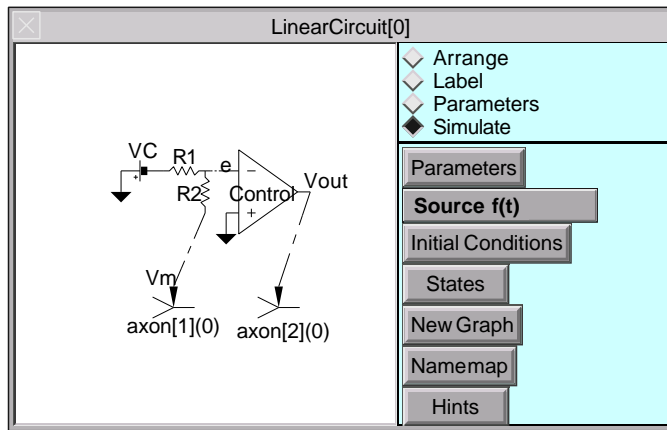
Model files	Download zip file	Auto-launch	Help downloading and running models
<ul style="list-style-type: none"> <ul style="list-style-type: none"> MRGaxon README AXNODE.mod MRGaxon.hoc mosinit.hoc MRGaxon.ses 	<p>SIMULATION OF PNS MYELINATED AXON</p> <p>This model is described in detail in:</p> <p>McIntyre CC, Richardson AG, and Grill WM. Modeling the excitability of mammalian nerve fibers: influence of afterpotentials on the recovery cycle. <i>Journal of Neurophysiology</i> 87:995-1006, 2002.</p> <p>The model is set up to reproduce part of Fig 2A from this paper.</p> <p>This model can not be used with NEURON v5.1 as errors in the extracellular mechanism of v5.1 exist related to xc. The original stimulations were run on v4.3.1. NEURON v5.2 has corrected the limitations in v5.1 and can be used to run this model.</p> <p>Please contact mcintyre@bme.jhu.edu if you have any questions about</p>		

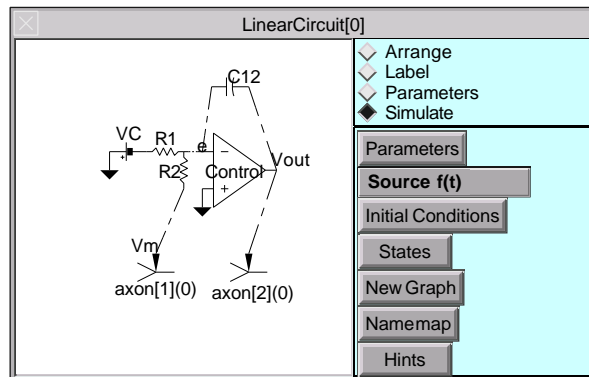
Total site hits since January 1, 2002: **346093**

[ModelDB Home](#) [SenseLab Home](#) [Help](#)
 Questions, comments, problems? Email the [ModelDB Administrator](#)
[How to cite ModelDB](#)









Values for LinearCircuit[0]

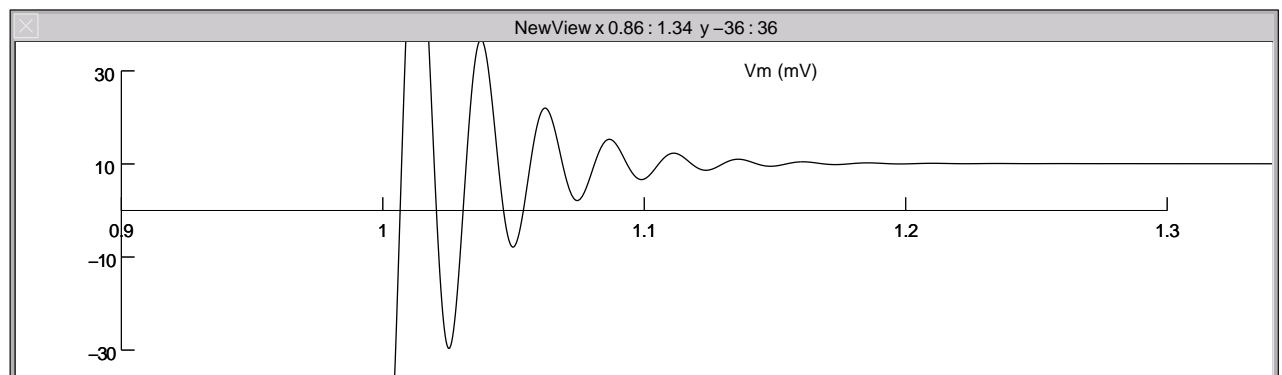
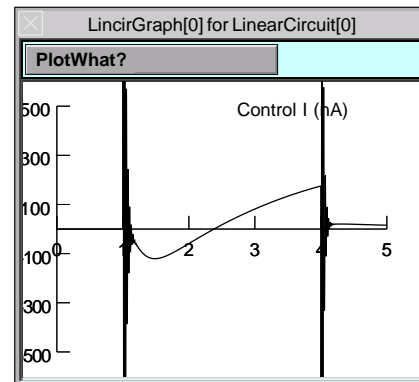
Control Gain	1e+05
Control Tau (ms)	0
R1 (Mohm)	1e+05
R2 (Mohm)	1e+05
C12 (nF)	1e-08

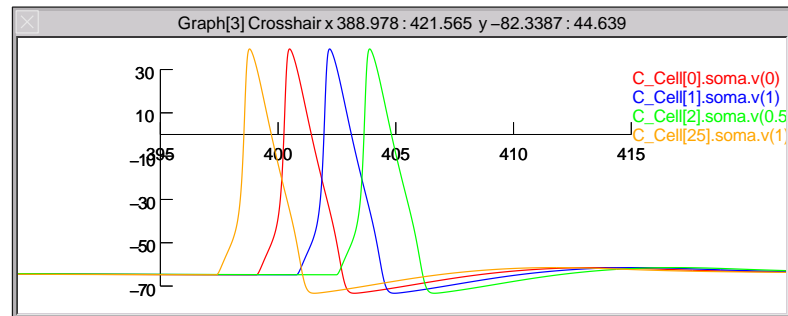
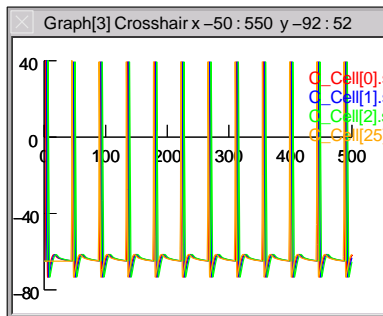
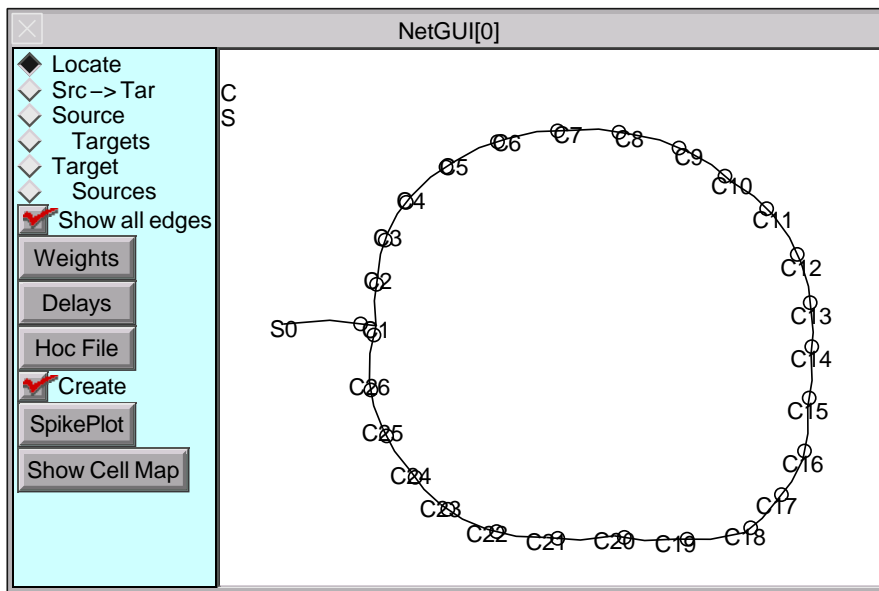
VariableTimeStep

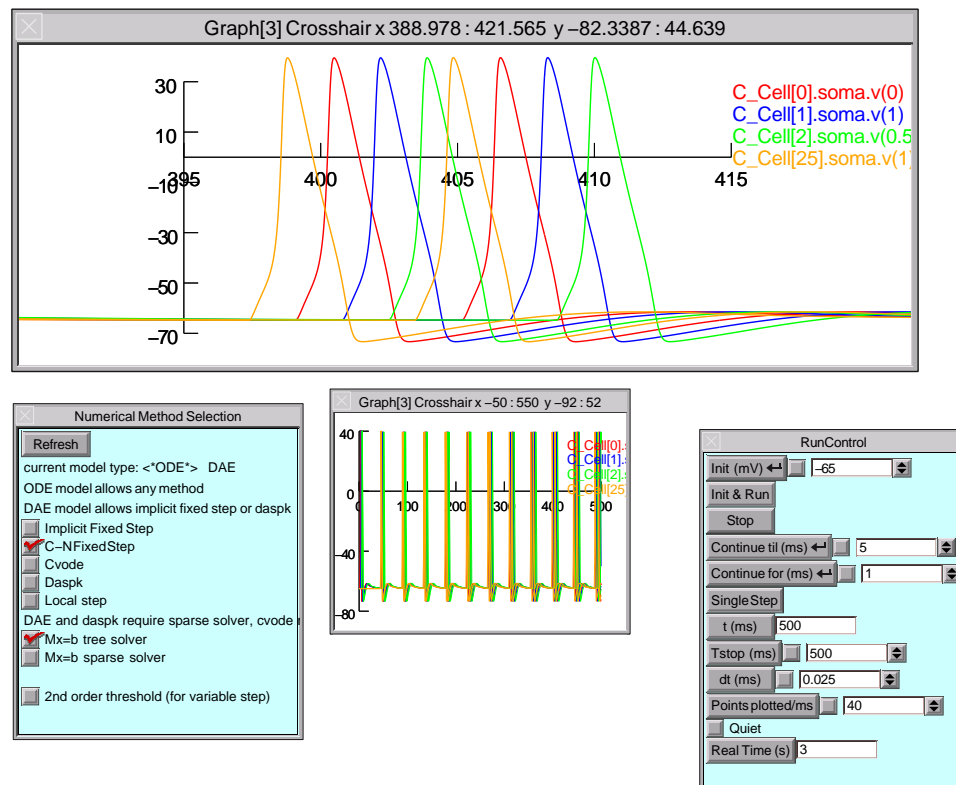
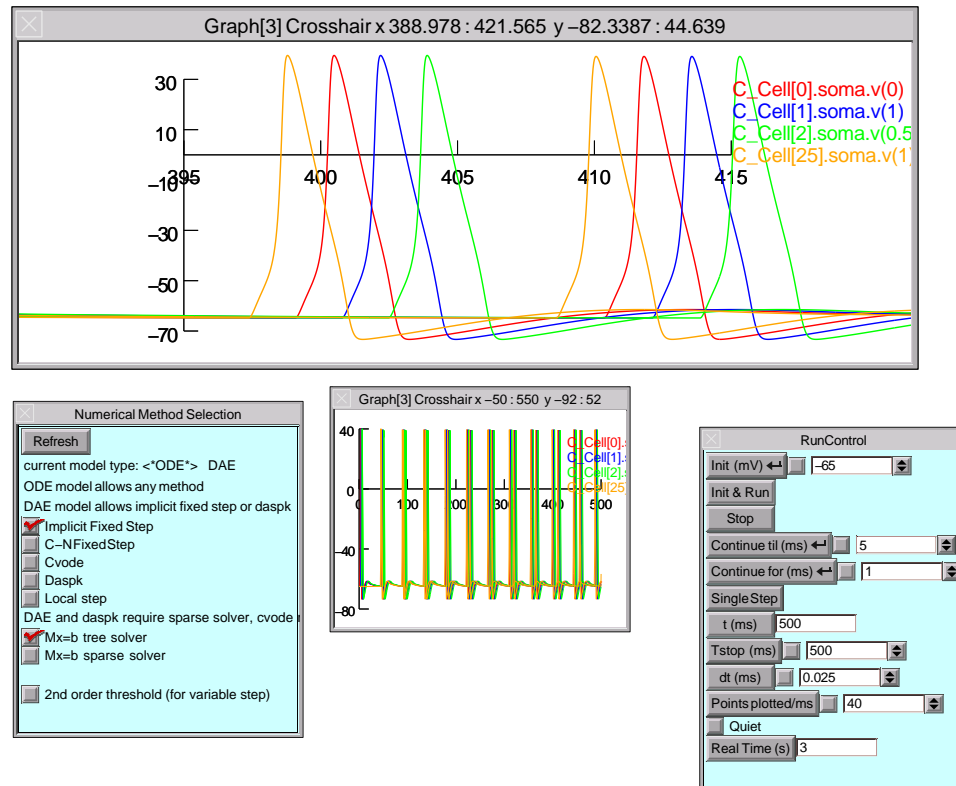
☒ Use variable dt

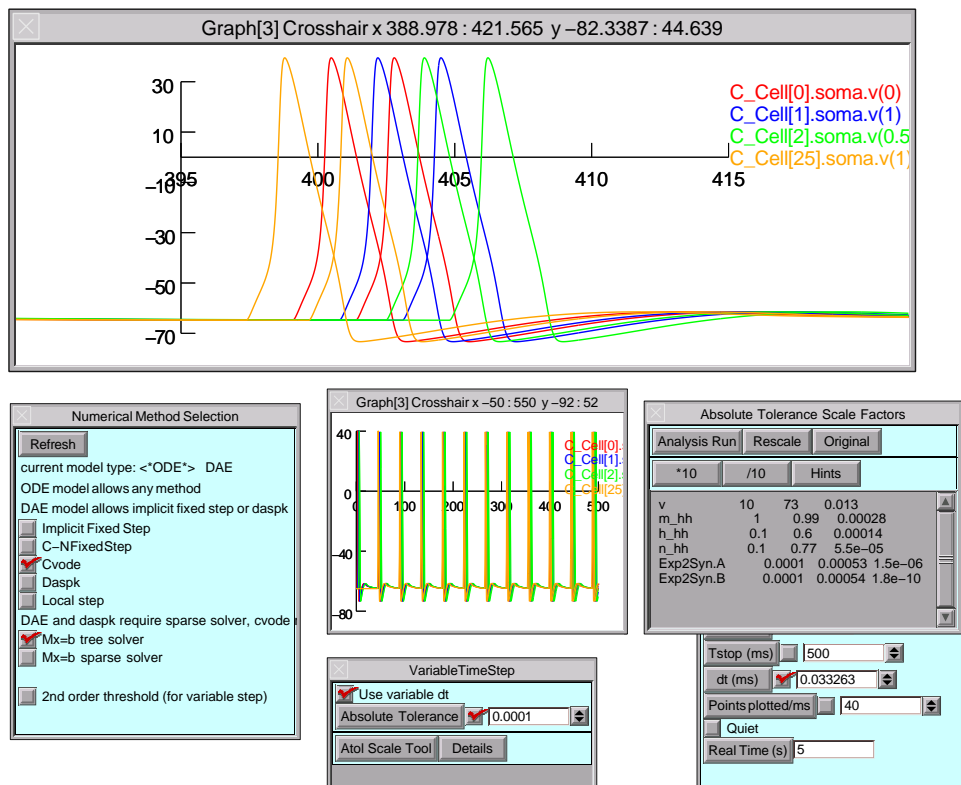
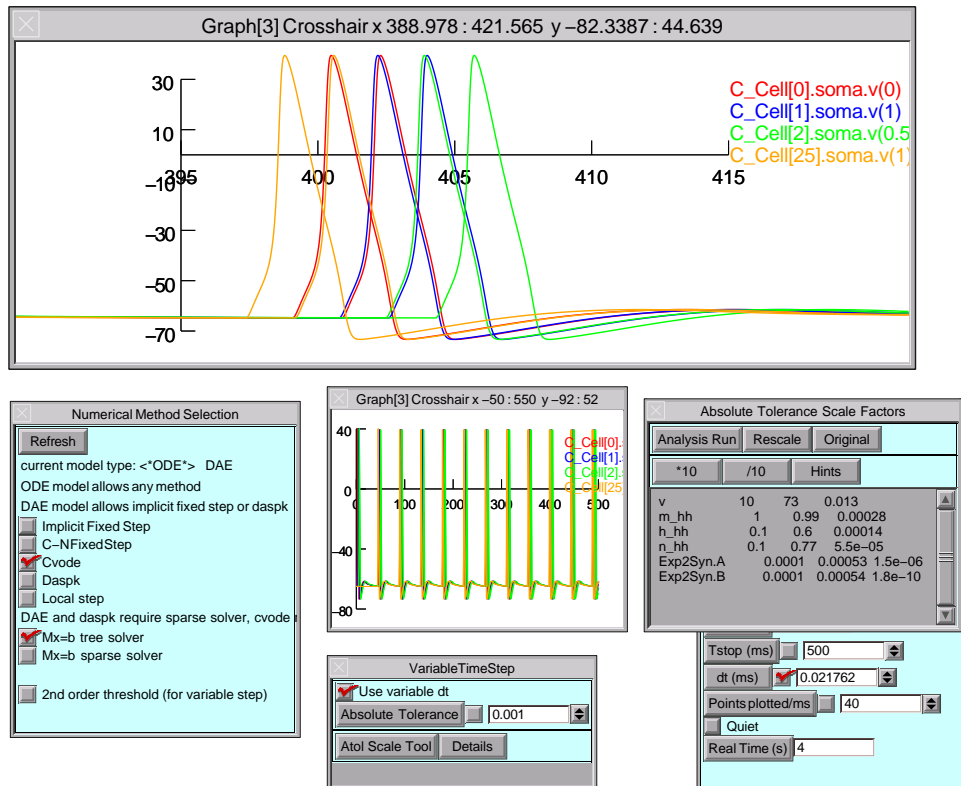
Absolute Tolerance 0.001

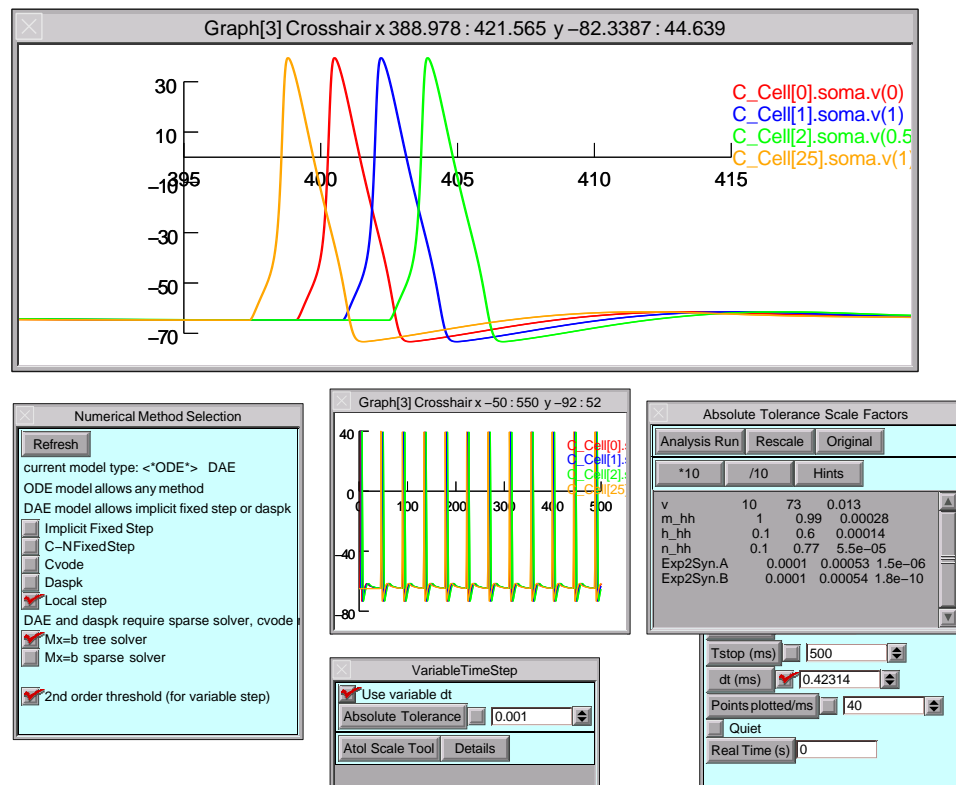
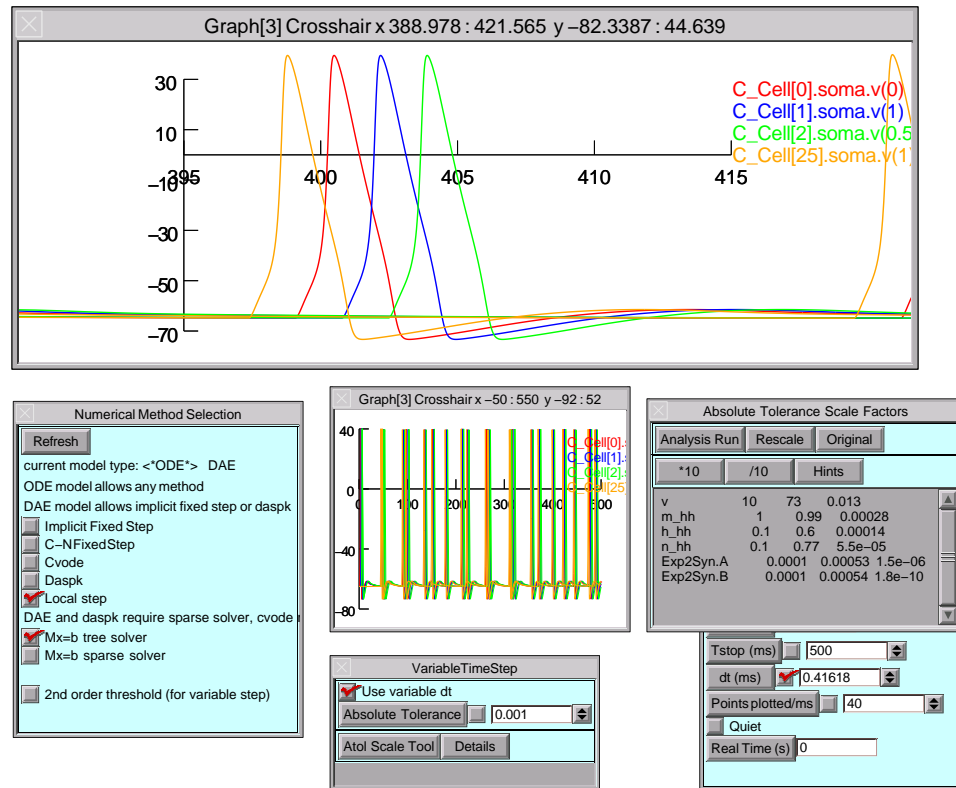
Atol Scale Tool Details











```
STATE { o }
```

```
BREAKPOINT {
  SOLVE state
  ik = gbar*o*(v - ek)
}
```

```
LOCAL fac
```

```
PROCEDURE state() {
  rate(v)
  o = o + fac*(oinf - o)
}
```

```
PROCEDURE rate(v (mV)) {
  LOCAL a
  a = alp(v)
  tau = 1/(a + bet(v))
  oinf = a*tau
  fac = (1 - exp(-dt/tau))
}
```

```
STATE { o }
```

```
BREAKPOINT {
  SOLVE state METHOD cnexp
  ik = gbar*o*(v - ek)
}
```

```
DERIVATIVE state {
  rate(v)
  o' = (oinf - o)/tau
}
```

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

```
BREAKPOINT {
  if (t >= del) { ← at_time(del)
    i = f(t-del)
  }else{
    i = 0
  }
}
```

(deprecated)

```

INITIAL {
    on = 0
    net_send(del, 1)
}

BREAKPOINT {
    if (t >= del) {
        i = f(t-del)
    }else{
        i = 0
    }
}

BREAKPOINT {
    if (on == 1) {
        i = f(t-del)
    }else{
        i = 0
    }
}

NET_RECEIVE(w) {
    if (flag == 1) {
        on = 1
    }
}

```

TITLE minimal model of GABA_A receptors

COMMENT

Minimal kinetic model for GABA_A receptors

Model of Destexhe, Mainen & Sejnowski, 1994:

(closed) + T \leftrightarrow (open)

The simplest kinetics are considered for the binding of transmitter (T) to open postsynaptic receptors. The corresponding equations are in similar form as the Hodgkin–Huxley model:

$$dr/dt = \alpha * [T] * (1-r) - \beta * r$$

$$I = g_{\max} * [\text{open}] * (V - E_{\text{rev}})$$

where [T] is the transmitter concentration and r is the fraction of receptors in the open form.

If the time course of transmitter occurs as a pulse of fixed duration, then this first-order model can be solved analytically, leading to a very fast mechanism for simulating synaptic currents, since no differential equation must be solved (see Destexhe, Mainen & Sejnowski, 1994).

```

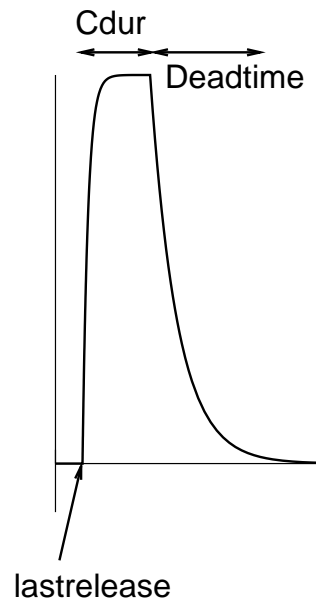
PROCEDURE release() { LOCAL q
:will crash if user hasn't set pre with the connect statement

q = ((t - lastrelease) - Cdur) : time since last release ended

                                : ready for another release?
if (q > Deadtime) {
  if (pre > Prethresh) {       : spike occurred?
    C = Cmax                   : start new release
    R0 = R
    lastrelease = t
  }
} else if (q < 0) {             : still releasing?
  : do nothing
} else if (C == Cmax) {        : in dead time after release
  R1 = R
  C = 0.
}

if (C > 0) {                   : transmitter being released?
  R = Rinf + (R0 - Rinf) * exp(-(t - lastrelease) / Rtau)
} else {                       : no release occurring
  R = R1 * exp(-Beta * (t - (lastrelease + Cdur)))
}
}

```



```

...
dr/dt = alpha * [T] * (1-r) - beta * r

```

where [T] is the transmitter concentration and r is the fraction of receptors in the open form.

```

...

```

```

INITIAL {
  t0 = 0
  r = 0
}

```

```

DERIVATIVE state {
  r' = (rinf - r)/rtau
}

```

```

NET_RECEIVE(w) {
  if (flag == 0) { : external spike, transmitter on
    rinf = alpha*T/(alpha*T + beta)
    rtau = 1/(alpha*T + beta)
    net_send(Cdur, 1)
  } else if (flag == 1) { :transmitter off
    rinf = 0
    rtau = 1/beta
  }
}
}

```

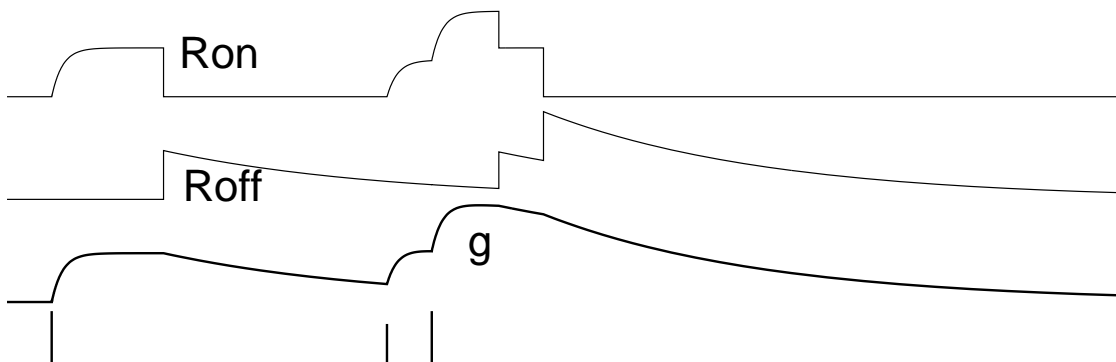
```

STATE {Ron Roff}
INITIAL {
  Ron = 0  Roff = 0
  Rinf = Alpha / (Alpha + Beta)
  Rtau = 1 / (Alpha + Beta)
  Rdelta = Rinf*(1 - exp(-Cdur/Rtau))
  synon = 0
}
BREAKPOINT {
  SOLVE release METHOD cnexp
  g = (Ron + Roff)*1(umho)
  i = g*(v - Erev)
}

DERIVATIVE release {
  Ron' = (synon*Rinf - Ron)/Rtau
  Roff' = -Beta*Roff
}

NET_RECEIVE(weight) {
  if (flag == 0) { : spike - T on
    synon = synon + weight
    net_send(Cdur, 1)
  }else{ : transmitter off
    synon = synon - weight
    Ron = Ron - weight*Rdelta
    Roff = Roff + weight*Rdelta
  }
}

```



```

setpointer gabaa[i], cell[j].axon.v(1)
gabaa[i].Prethresh = -10

```



```

cell[j].axon { nc = new NetCon(&v(1), gabaa[i]) }
nc.threshold = -10
nc.delay = 0

```

```
proc advance() {
  fadvance()
  if (t == t1) { p() }
}
```



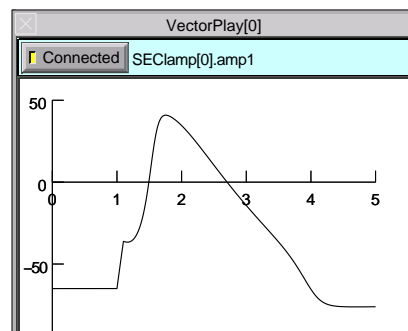
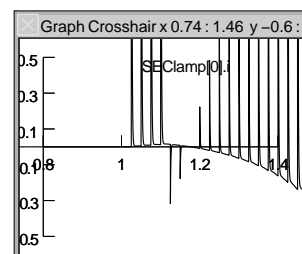
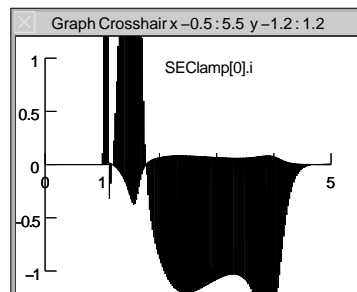
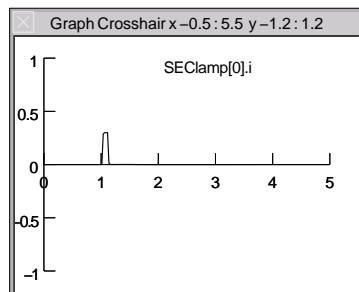
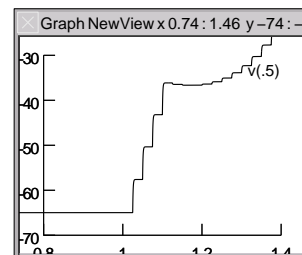
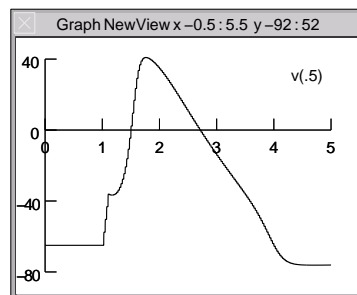
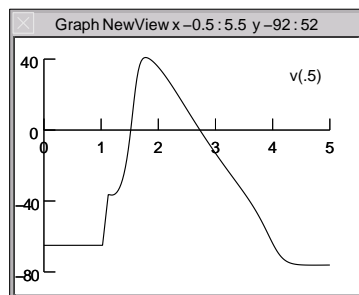
```
fi = new FInitializeHandler("ev()")
proc ev() {
  ccode.event(t1, "p()")
}
```

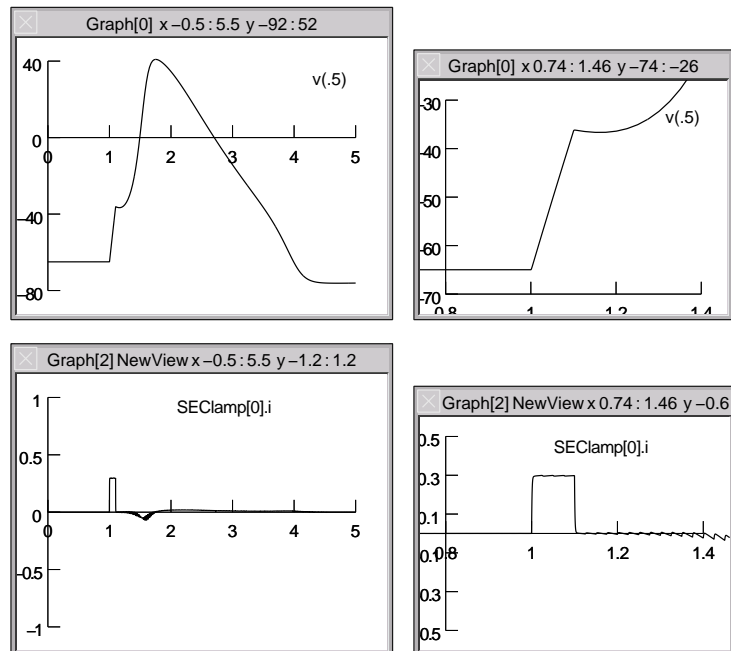
```
proc advance() {
  fadvance()
  if (soma.v(.5) > 10) { p() }
}
```



```
soma { nc = new NetCon(&v(.5), nil)
  nc.threshold = 10
  nc.record("p()")
}
```

```
proc p() {
  // if ANY parameters or states
  // change then be sure to
  ccode.re_init()
}
```





```
soma vvec.play(&SEClamp[0].amp1, tvec, 1)
```


Networks: spike-triggered synaptic transmission, events, and artificial spiking cells

1. Define the types of cells
2. Create each cell in the network
3. Connect the cells

Communication between cells

Gap junctions

Synaptic transmission
graded
spike-triggered

Graded synaptic transmission

Physical system:

A presynaptic variable governs
continuous transmitter release

Transmitter modulates
a postsynaptic property



Problem: how does postsynaptic cell know V_{pre} ?

Graded synaptic transmission *continued*

Answer: use POINTER to link postsynaptic variable
to the presynaptic variable

NMODL specification of synaptic mechanism:

```
NEURON {
  POINT_PROCESS Syn
  POINTER v_pre
}
```

hoc usage

```
objref syn
dend syn = new Syn(0.5)
setpointer syn.v_pre, precell.axon.v(1)
```

Spike-triggered synaptic transmission

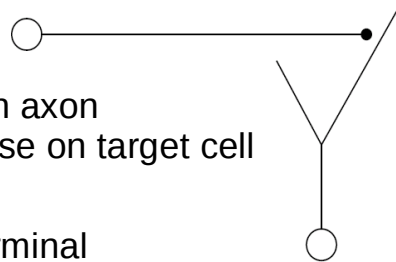
Physical system:

Presynaptic neuron with axon
that projects to synapse on target cell

Conceptual model:

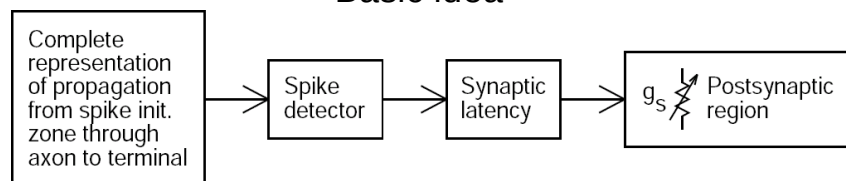
Spike in presynaptic terminal
triggers transmitter release;
presynaptic details unimportant

Postsynaptic effect described by
DE or kinetic scheme that is perturbed by
occurrence of a presynaptic spike

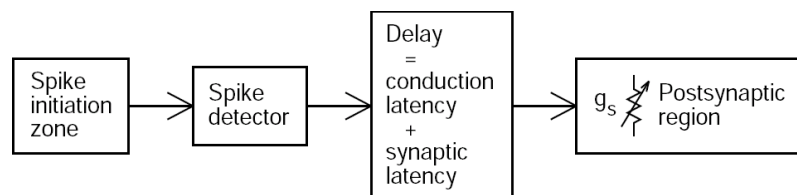


Spike-triggered transmission: computational implementation

Basic idea



More efficient: "virtual spike propagation"



The NetCon class

hoc usage

```
netcon = new NetCon(source, target)
presection netcon = new NetCon(&v(x), \
    target, threshold, delay, weight)
```

Defaults

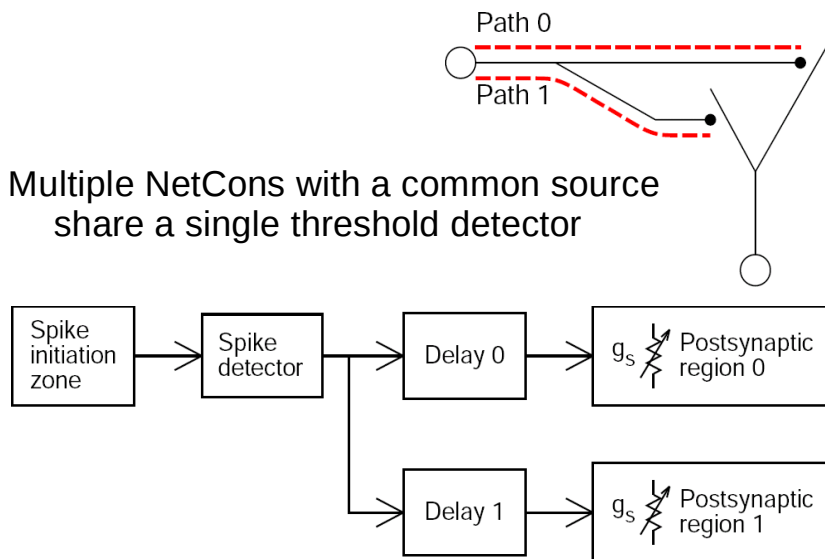
```
threshold = 10
delay = 1 // must be >= 0
weight = 0
```

NMODL specification of synaptic mechanism

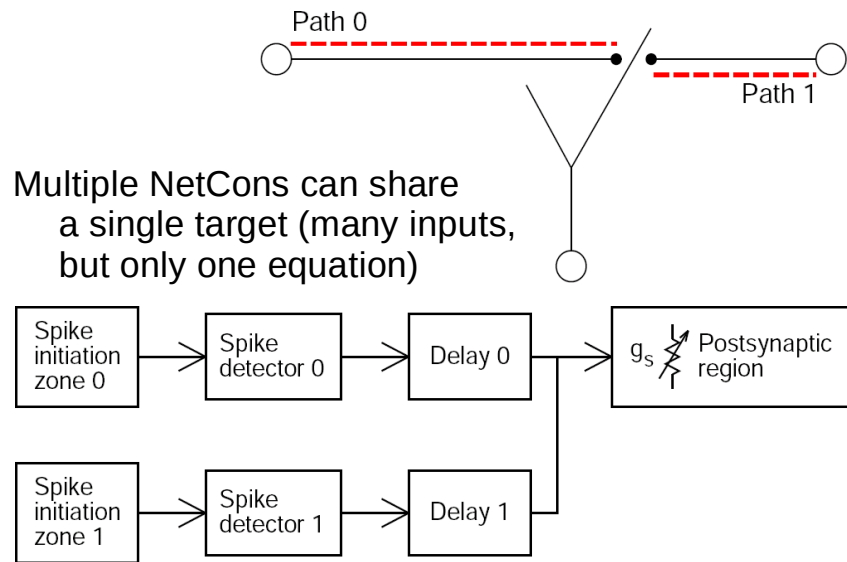
```
NET_RECEIVE(weight(microsiemens)) {
    . . .
}
```

Efficient divergence

Multiple NetCons with a common source
share a single threshold detector



Efficient convergence



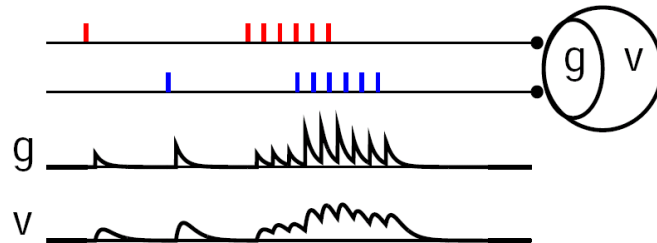
Example: g_s with fast rise and exponential decay

```

NEURON {
  POINT_PROCESS ExpSyn
  RANGE tau, e, i
  NONSPECIFIC_CURRENT i
}
... declarations ...
INITIAL { g = 0 }
BREAKPOINT {
  SOLVE state METHOD cnexp
  i = g*(v-e)
}
DERIVATIVE state { g' = -g/tau }
NET_RECEIVE(w (uS)) { g = g + w }

```

g_s with fast rise and exponential decay *continued*

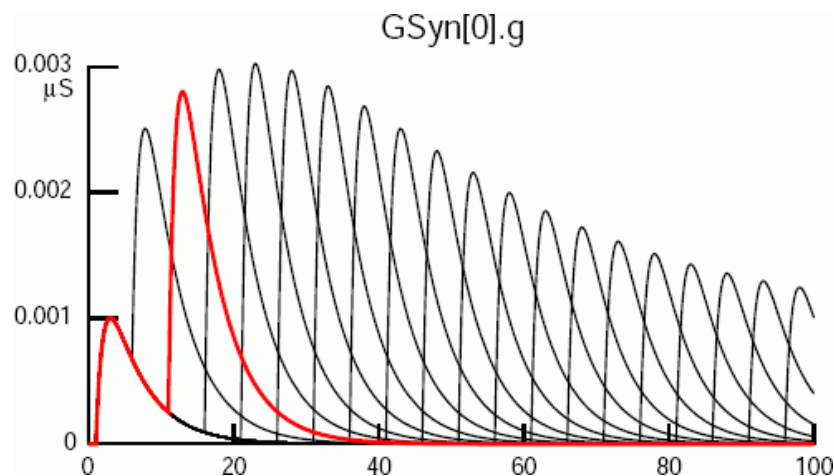


```

BREAKPOINT {
    SOLVE state METHOD cnexp
    i = g*(v-e)
}
DERIVATIVE state { g' = -g/tau }
NET_RECEIVE(w (uS)) { g = g + w }

```

Example: use-dependent synaptic plasticity

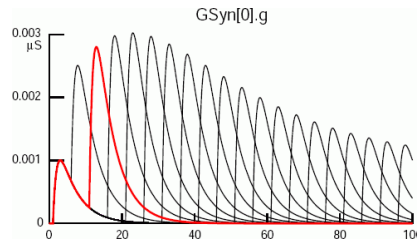


Use-dependent synaptic plasticity *continued*

```

BREAKPOINT {
  SOLVE state METHOD cnexp
  g = B - A
  i = g*(v-e)
}
DERIVATIVE state {
  A' = -A/tau1
  B' = -B/tau2
}
NET_RECEIVE(weight (uS), w, G1, G2, t0 (ms)) {
  INITIAL {w=0 G1=0 G2=0 t0=t}
  G1 = G1*exp(-(t-t0)/Gtau1)
  G2 = G2*exp(-(t-t0)/Gtau2)
  G1 = G1 + Ginc*Gfactor
  G2 = G2 + Ginc*Gfactor
  t0 = t
  w = weight*(1 + G2 - G1)
  g = g + w
  A = A + w*factor
  B = B + w*factor
}

```



Artificial spiking cells

"Integrate and fire" cells

Prerequisite: all state variables must be
analytically computable from a new initial condition

Orders of magnitude faster than numerical integration

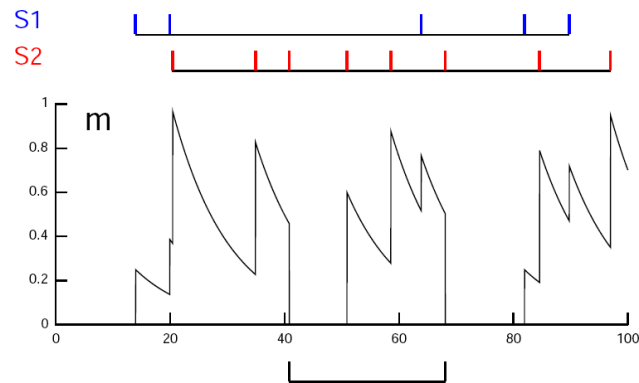
Event-driven simulation run time is

proportional to # of received events

independent of # of cells, # of connections,
and problem time

Hybrid networks

Example: leaky integrate and fire model

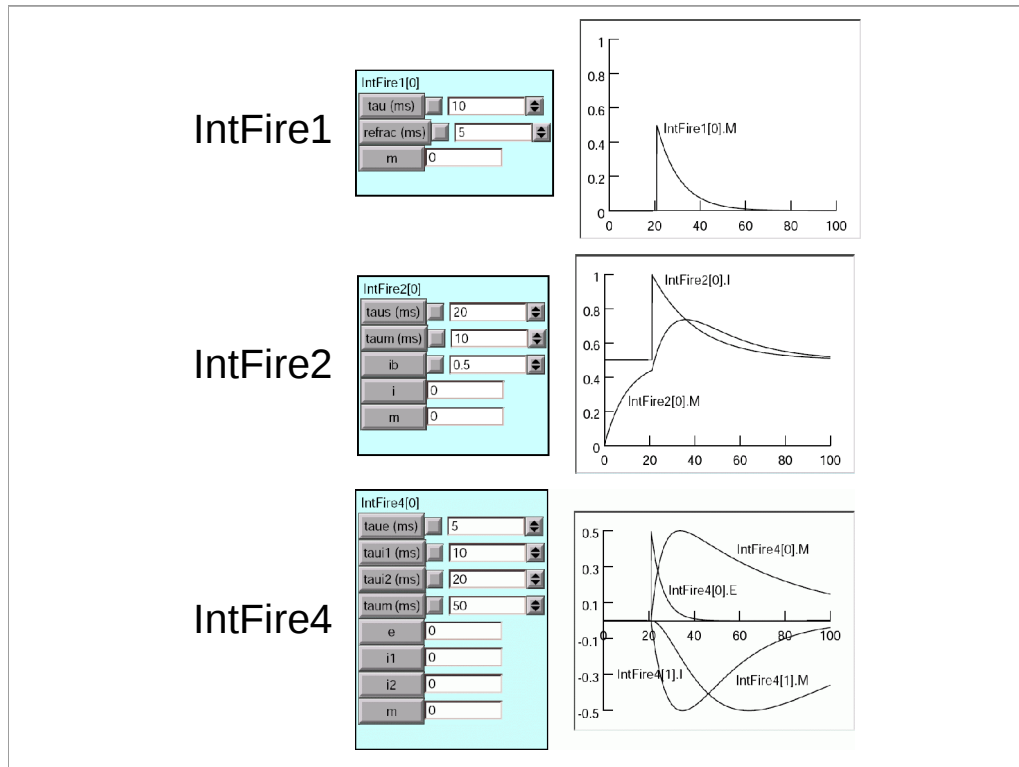


Leaky integrate and fire model *continued*

```

NEURON {
  ARTIFICIAL_CELL IntFire
  RANGE tau, m
}
... declarations ...
INITIAL { m = 0    t0 = t }
NET_RECEIVE (w) {
  m = m*exp(-(t-t0)/tau)
  t0 = t
  m = m + w
  if (m > 1) {
    net_event(t)
    m = 0
  }
}

```



Defining the types of cells

Artificial spiking cells

ARTIFICIAL_CELL with a NET_RECEIVE block
 that calls net_event

NetStim, IntFire1, IntFire2, IntFire4

Biophysical model cells

"Real" model cells

Sections and density mechanisms

Synapses are POINT_PROCESSES
 that affect membrane current
 and have a NET_RECEIVE block,
 e.g. ExpSyn, Exp2Syn

Defining types of biophysical model cells

Encapsulate in a class

```
begintemplate Cell
  public soma, E, I
  create soma
  objref E, I
  proc init() {
    soma {
      insert hh
      E = new ExpSyn(0.5)
      I = new Exp2Syn(0.5)
      I.e = -80
    }
  }
endtemplate Cell

objref bag_of_cells
bag_of_cells = new List()
for i = 1,1000 bag_of_cells.append(new Cell())
```

Connecting cells

Which setup strategy is more efficient?

Iterate over sources

```
for each cell {
  connect this cell to its targets
}
```

or iterate over targets?

```
for each cell {
  connect sources to this cell
}
```

Connecting cells

For a net distributed over multiple CPUs,
it is most efficient to iterate over targets first.

```
for each cell {  
    connect sources to this cell  
}
```


Synchronizing network parameters

Close Hide

Number of all to all cells

All to all connection weight ☒

Delay (ms) ☒

Cell time constant (ms)

Lowest natural interval

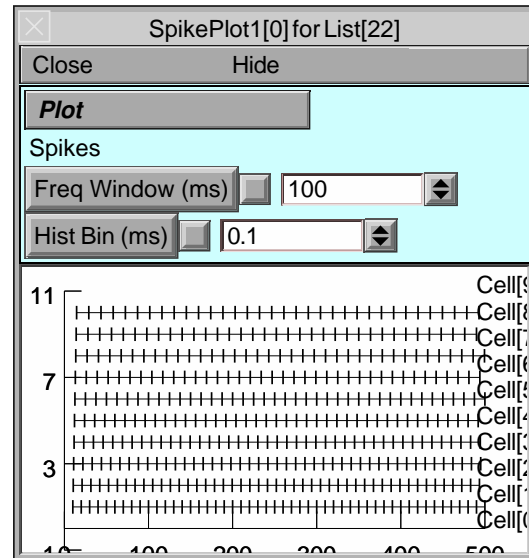
Highest natural interval

NEURONDemonstrations

Close Hide

Synchronizing net (artificial cells)

- ☐ Patch: HH
- ☐ Stylized
- ☐ Pyramidal
- ☐ Release
- ☒ Synchronizing net (artificial cells)
- ☒ LinearCircuit: DynamicClamp
- ☐ Stochastic Single Channels: HH
- ☐ No model



Synchronizing network parameters

Close Hide

Number of all to all cells

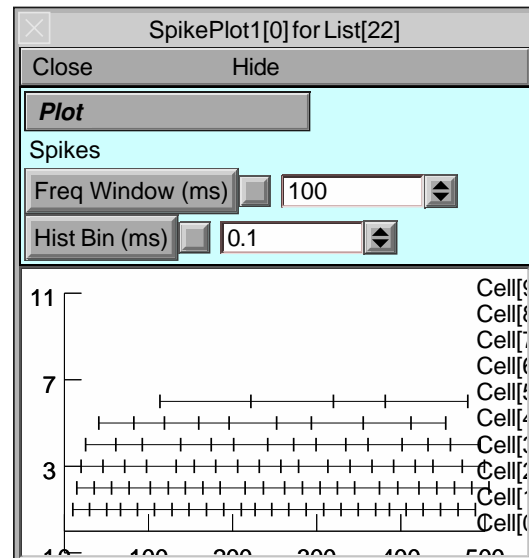
All to all connection weight

Delay (ms) ☒

Cell time constant (ms)

Lowest natural interval

Highest natural interval



Synchronizing network parameters

Close Hide

Number of all to all cells

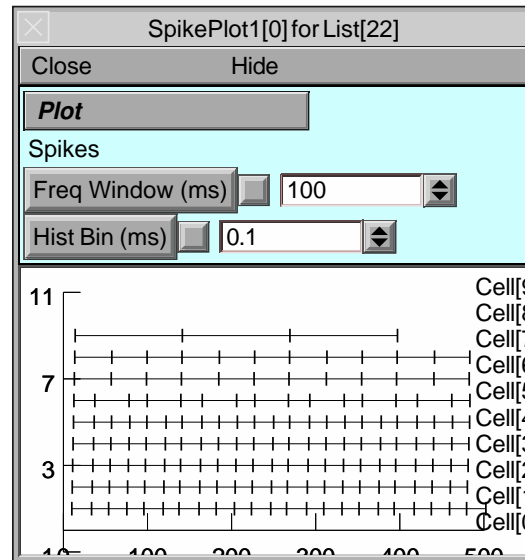
All to all connection weight

Delay (ms)

Cell time constant (ms)

Lowest natural interval

Highest natural interval



Synchronizing network parameters

Close Hide

Number of all to all cells

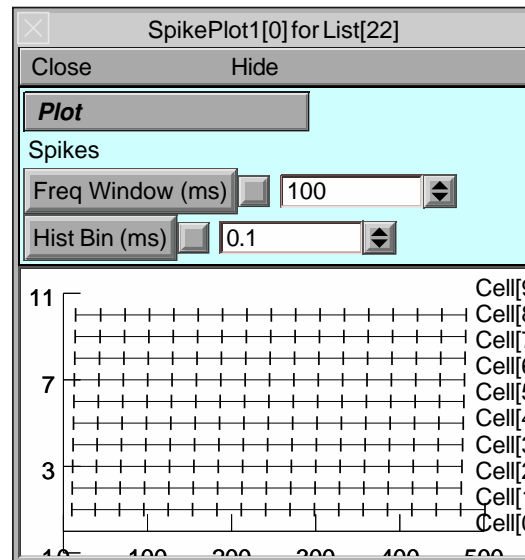
All to all connection weight

Delay (ms) ☒

Cell time constant (ms)

Lowest natural interval

Highest natural interval



```

NEURON {
  ARTIFICIAL_CELL IntervalFire
  RANGE tau, m, invl
  : m plays the role of voltage
}

: dm/dt = (minf - m)/tau
: input event adds w to m
: when m = 1, or event makes m >= 1 cell fires
: minf is calculated so that the natural
: interval between spikes is invl

INITIAL {
  minf = 1/(1 - exp(-invl/tau))
  m = 0
  t0 = t
  net_send(firetime(), 1)
}

FUNCTION M() {
  M = minf + (m - minf)*exp(-(t - t0)/tau)
}

FUNCTION firetime()(ms) { : m < 1 and minf > 1
  firetime = tau*log((minf-m)/(minf - 1))
}

NET_RECEIVE (w) {
  m = M()
  t0 = t
  if (flag == 0) {
    m = m + w
    if (m > 1) {
      m = 0
      net_event(t)
    }
    net_move(t+firetime())
  }else{
    net_event(t)
    m = 0
    net_send(firetime(), 1)
  }
}

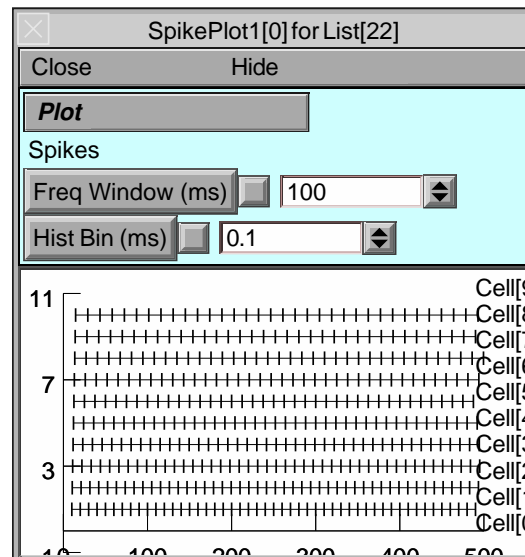
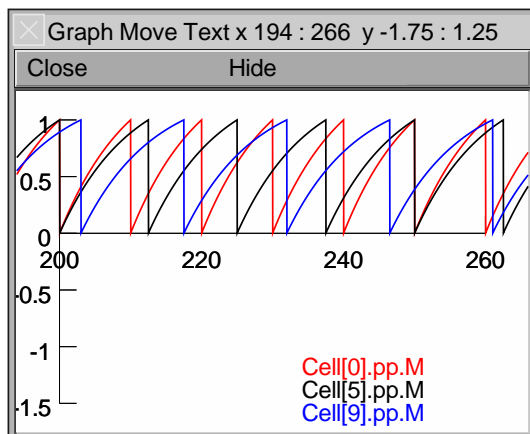
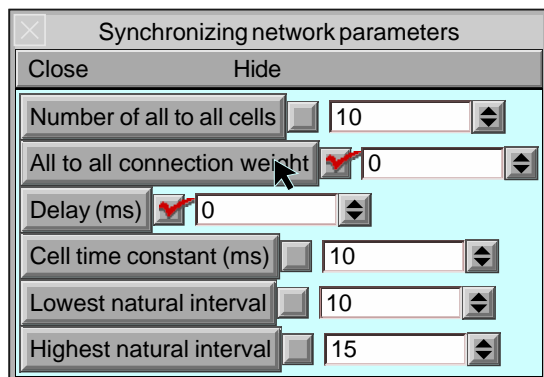
```

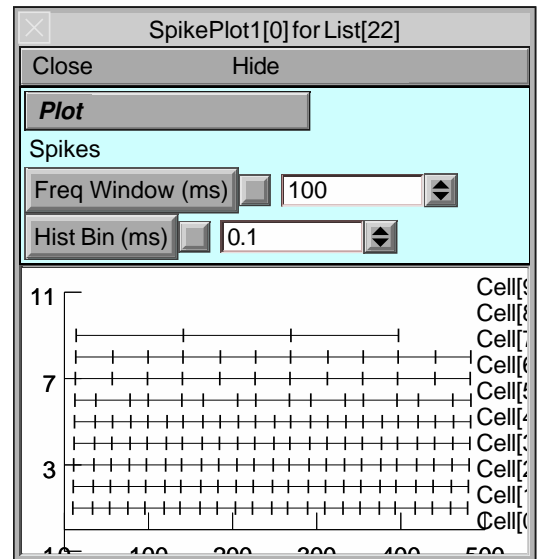
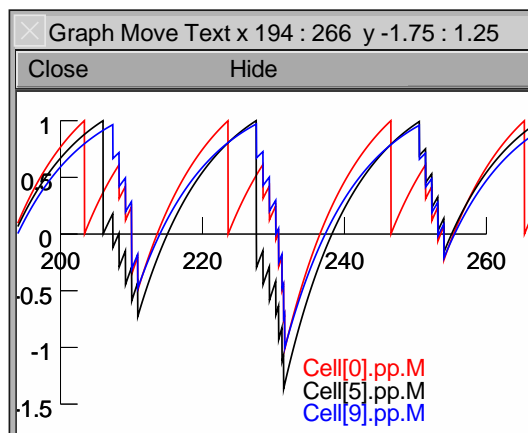
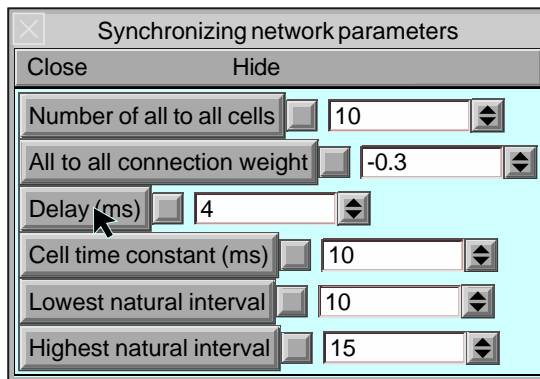
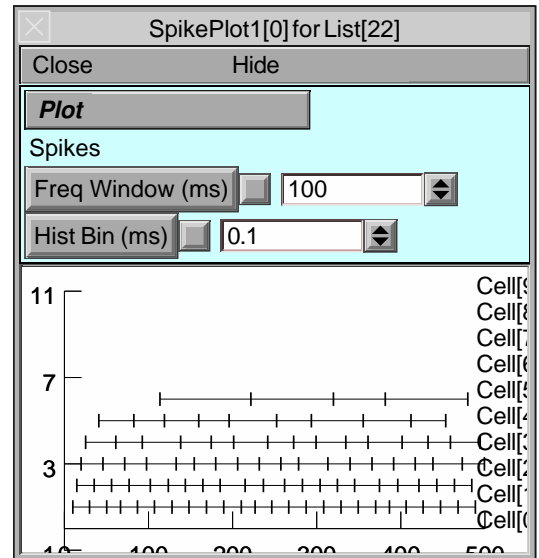
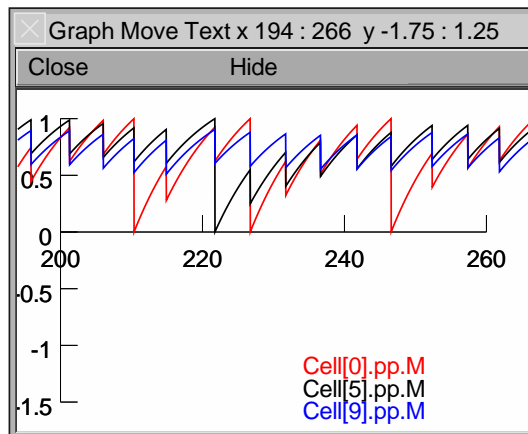
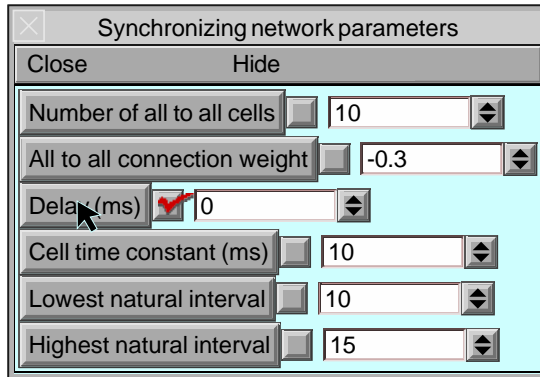
```

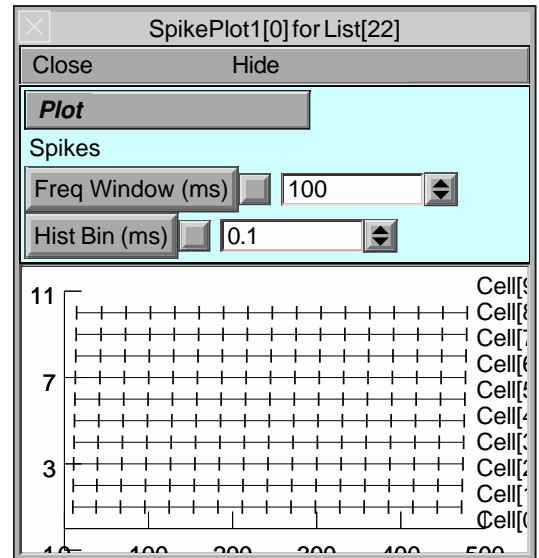
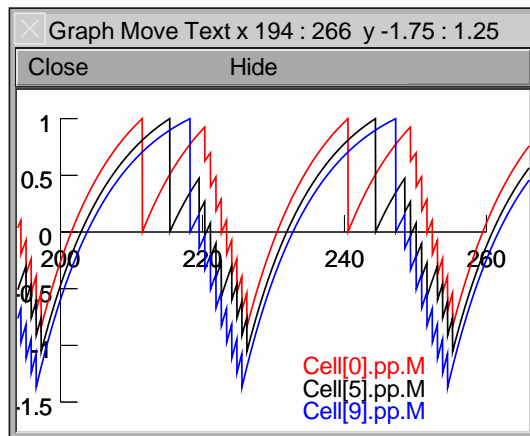
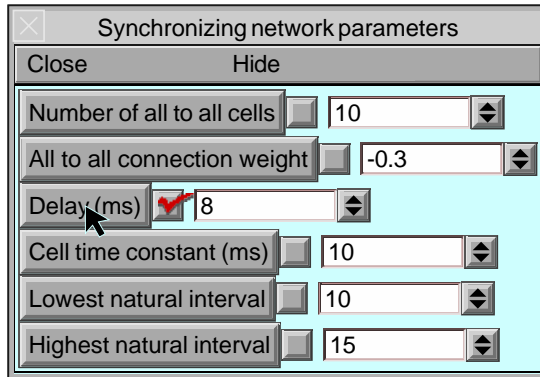
objref cells, nclist
{cells = new List() nclist = new List()}

proc createnet() {local i, j
  ncell = $1
  nclist.remove_all()
  cells.remove_all()
  for i=0, ncell-1 {
    cell_append(new Cell(), i, 0, 0)
  }
  for i=0, ncell-1 for j=0, ncell-1 if (i != j) {
    nc_append(i, j, -1, 0, 1)
  }
}

```







Parallel Computation

"Faster" is the only reason

But...

- greater programming complexity
- new kinds of bugs
- ...and not much help for fixing them.

Can the day or week of user effort be recovered?

- 8192 processor EPFL IBM BlueGene
- 1 hour at 700MHz
- 3 months at 3GHz

Parallel Computation

A simulation run takes about a second

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

A simulation run takes hours.

- want to spread the problem over several machines.

Parallel Computation

A simulation run takes hours.

want to spread the problem over several machines.

Network

Subnets on different machines

Cells communicate by:

logical spike events with significant
axonal, synaptic delay.

postsynaptic conductance depends
continuously on presynaptic voltage.

gap junctions

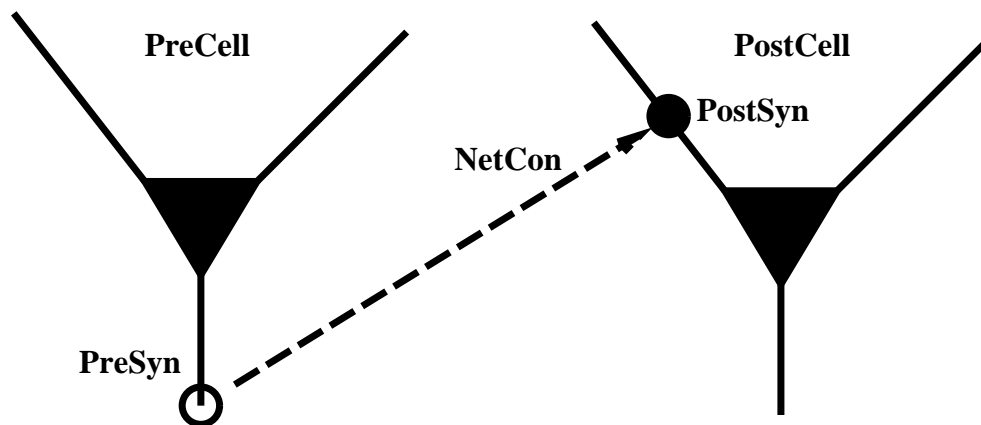
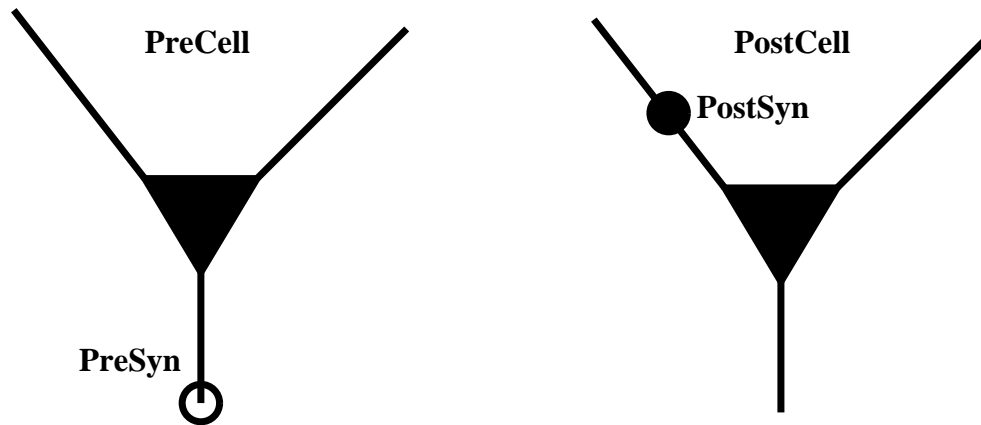
Parallel Computation

A simulation run takes hours.

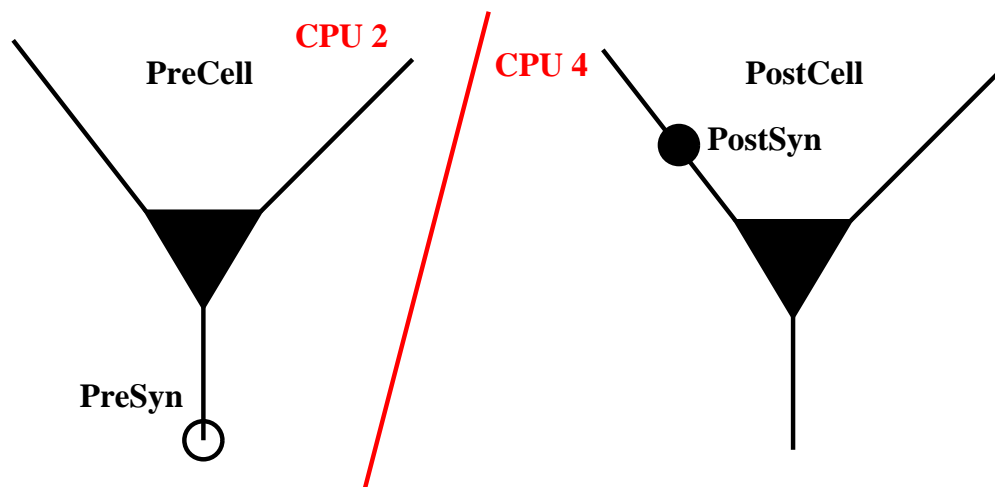
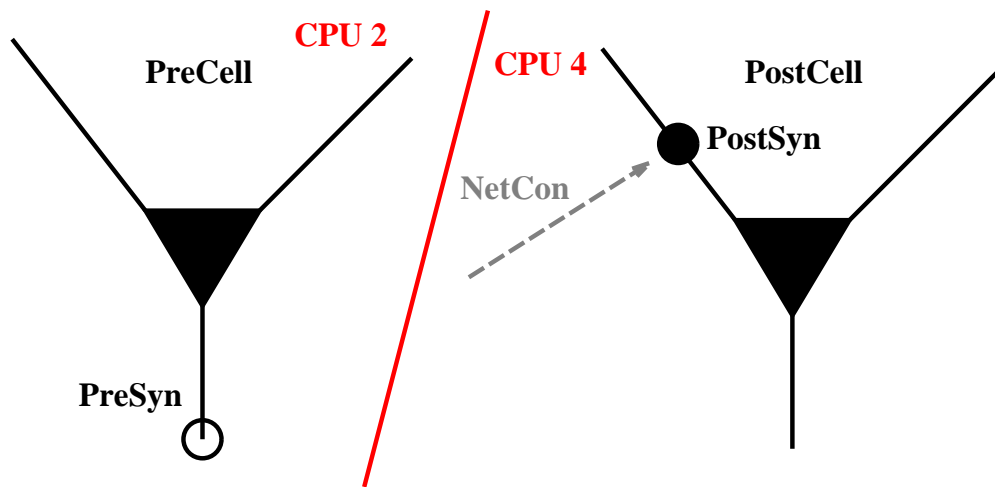
want to spread the problem over several machines.

Single cells

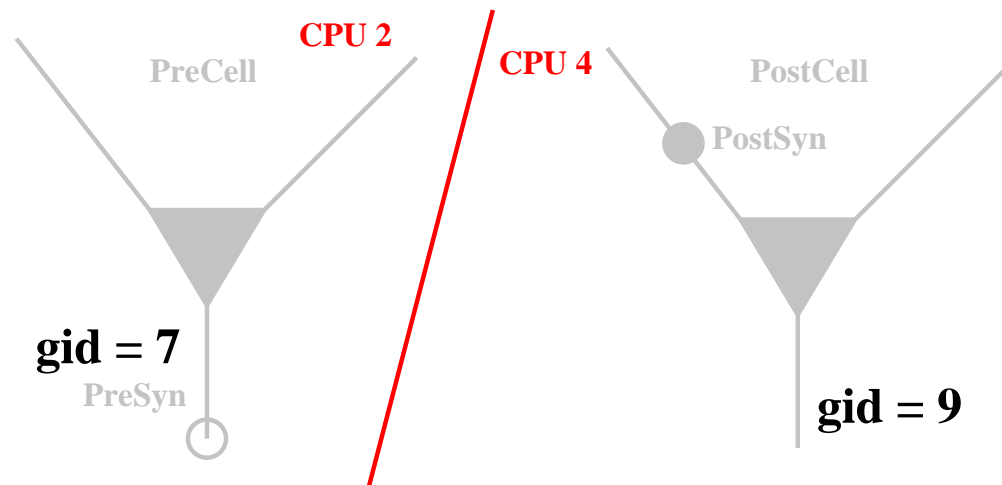
portions of the tree cable equation on
different machines.



```
nc = new NetCon(PreSyn, PostSyn)
```



```
pc = new ParallelContext()
```



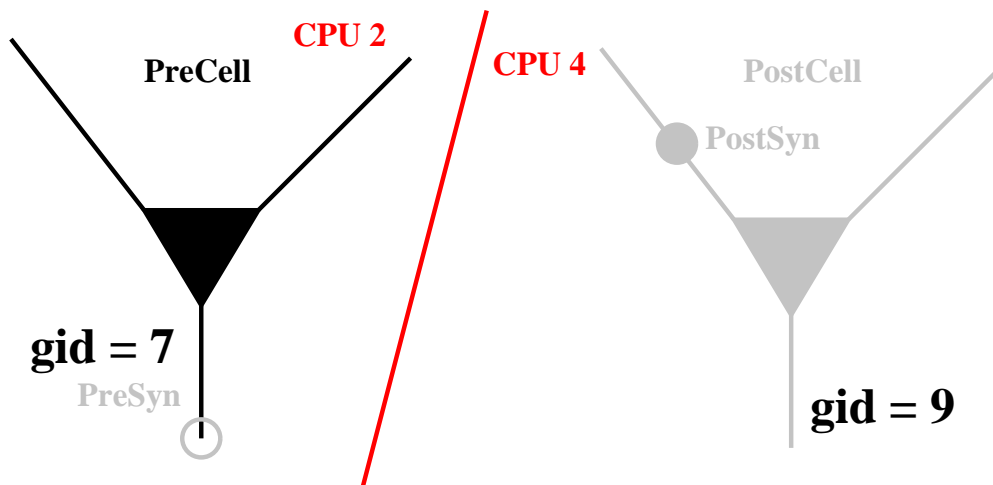
Every spike source (cell) must have a global id number.

CPU 0		...	CPU 3		CPU 4	
pc.id	0		pc.id	3	pc.id	4
pc.nhost	5		pc.nhost	5	pc.nhost	5
ncell	14		ncell	14	ncell	14
gid			gid		gid	
0			3		4	
5			8		9	
10			13			

An efficient way to distribute:

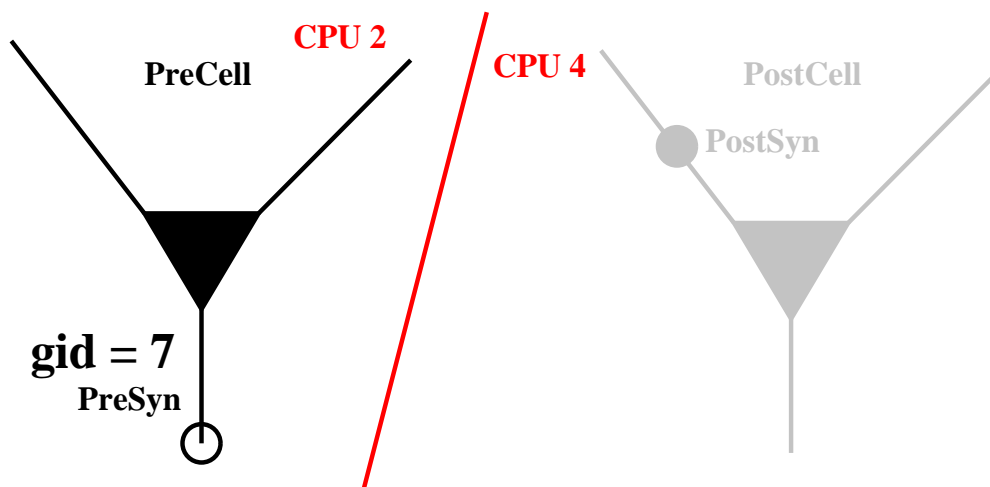
```
for (gid = pc.id; gid < ncell; gid += pc.nhost)
    pc.set_gid2node(gid, pc.id)
    ...
}
```

body executed only ncell/nhost times, not ncell.



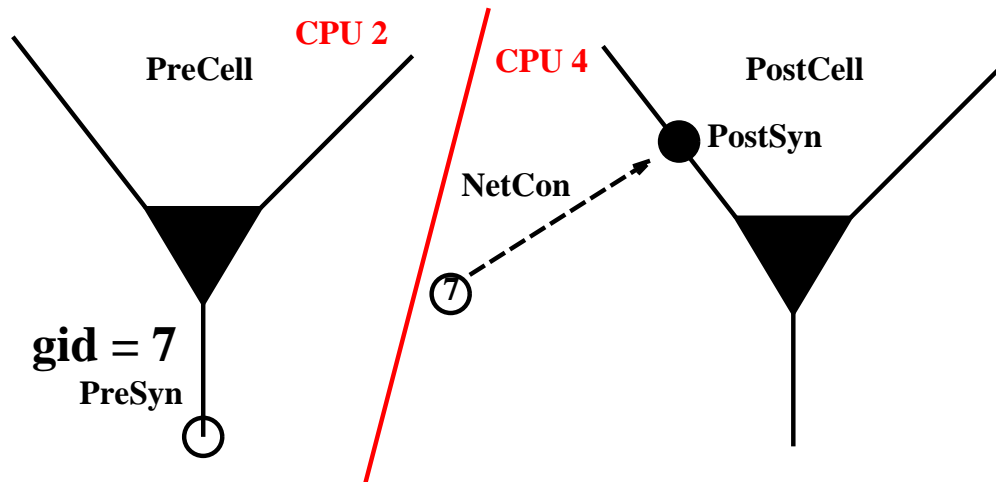
Create cell only where the gid exists.

```
if (pc.gid_exists(7)) {
    PreCell = new Cell()
}
```



Associate gid with spike source.

```
nc = new NetCon(PreSyn, nil)
pc.cell(7, nc)
```



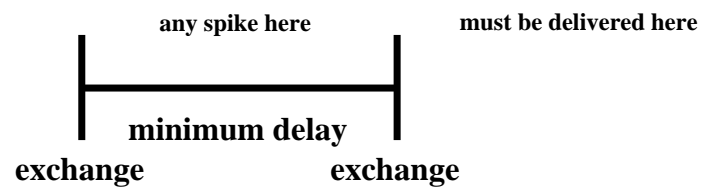
Create NetCon on CPU where target exists.

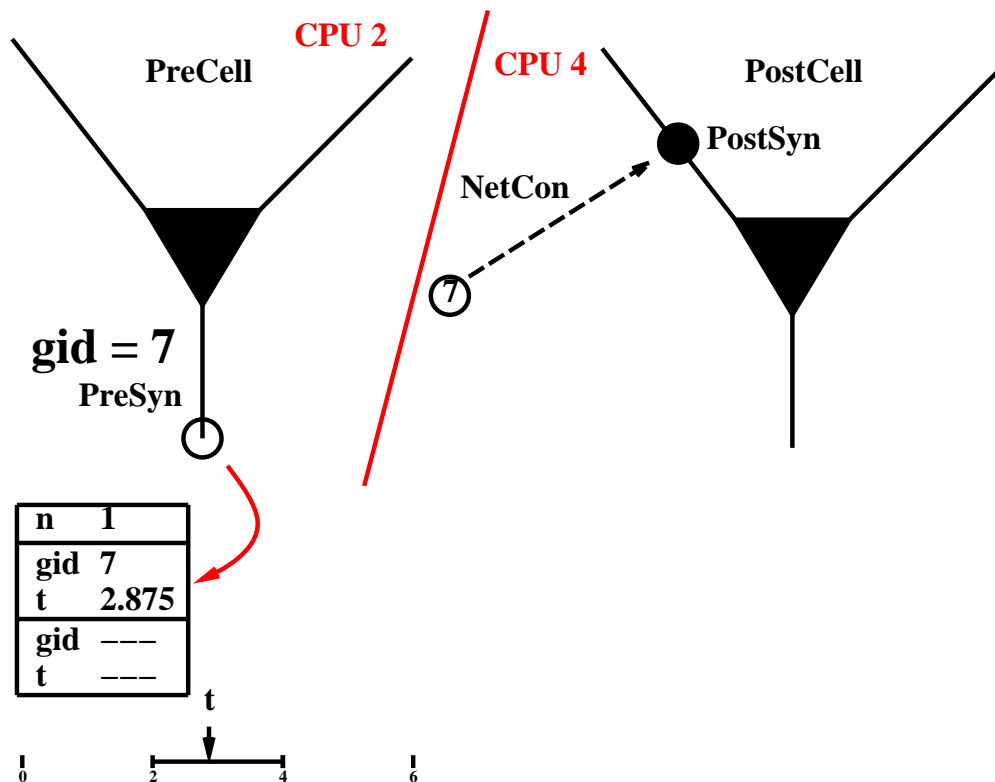
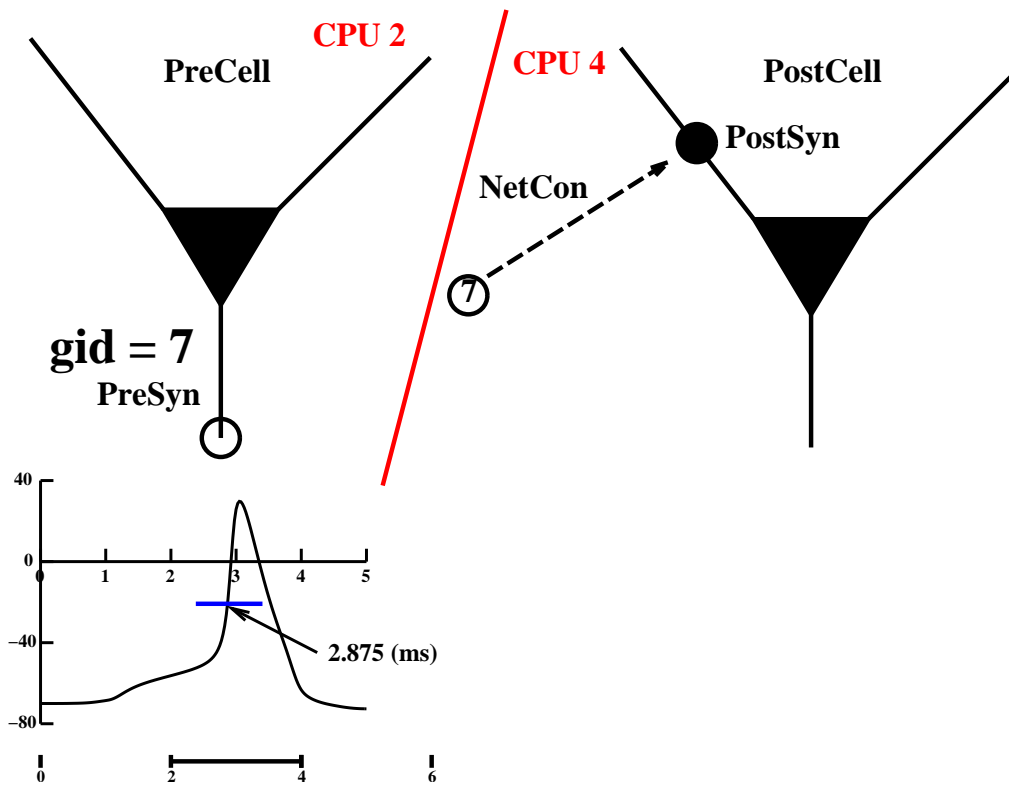
```
nc = pc.gid_connect(7, PostSyn
```

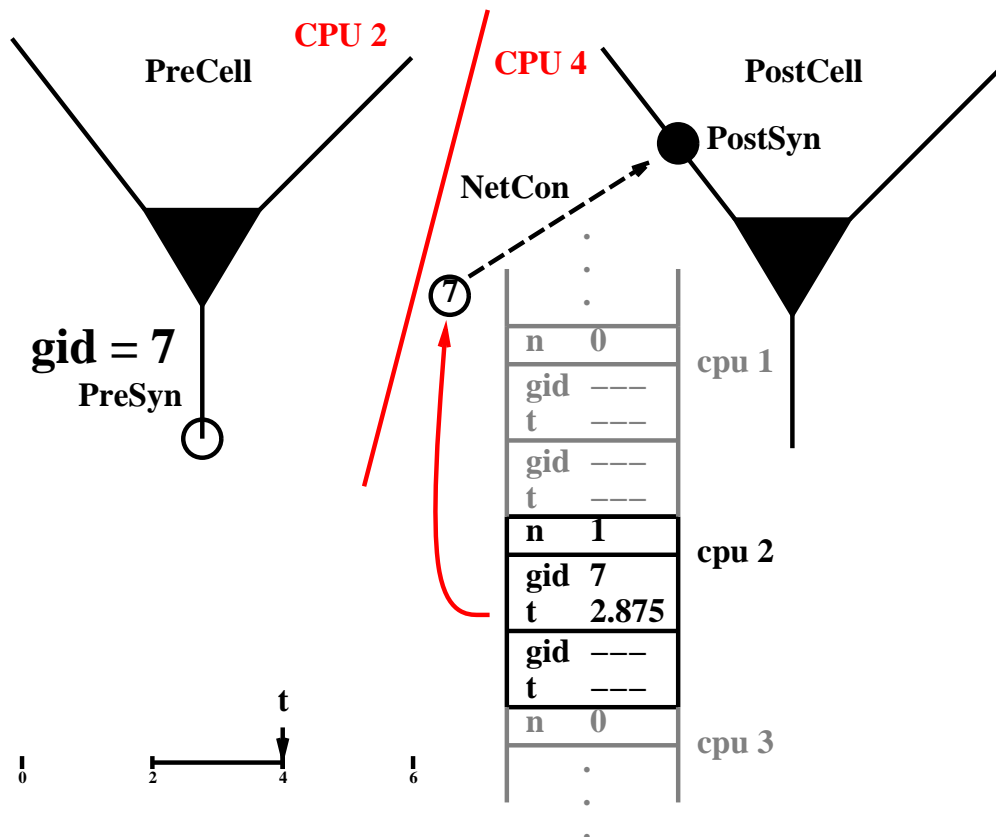
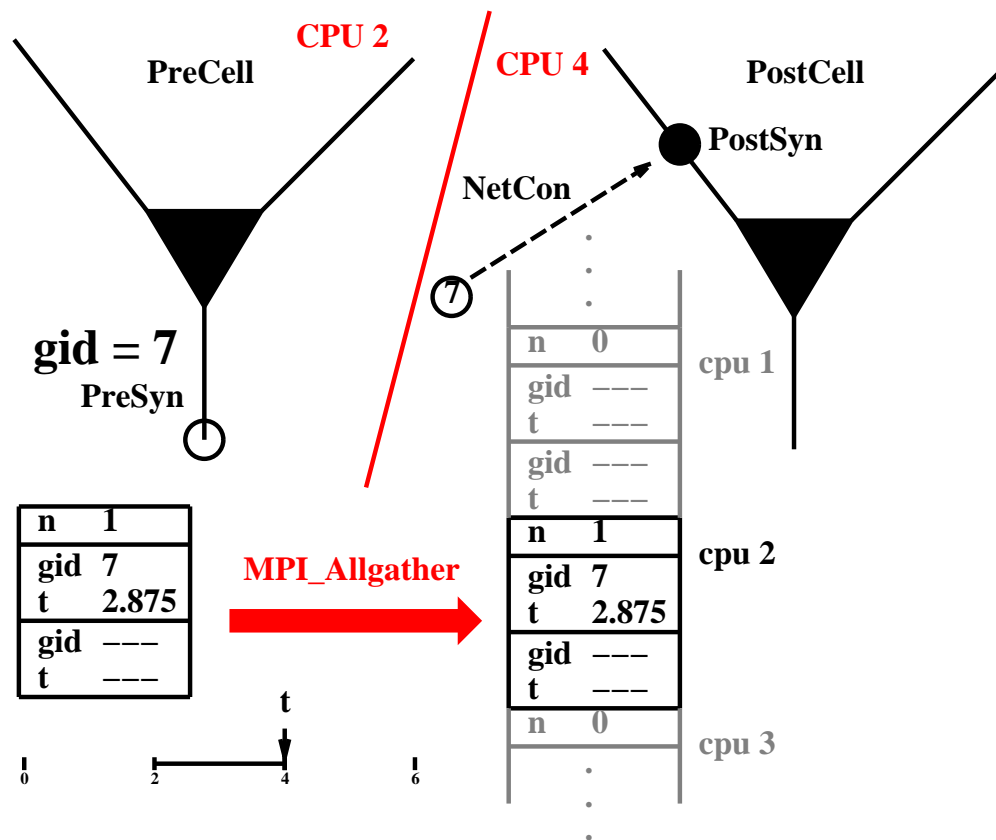
Run using the idiom

```
pc.set_maxstep(10)
stdinit()
pc.solve(tstop)
```

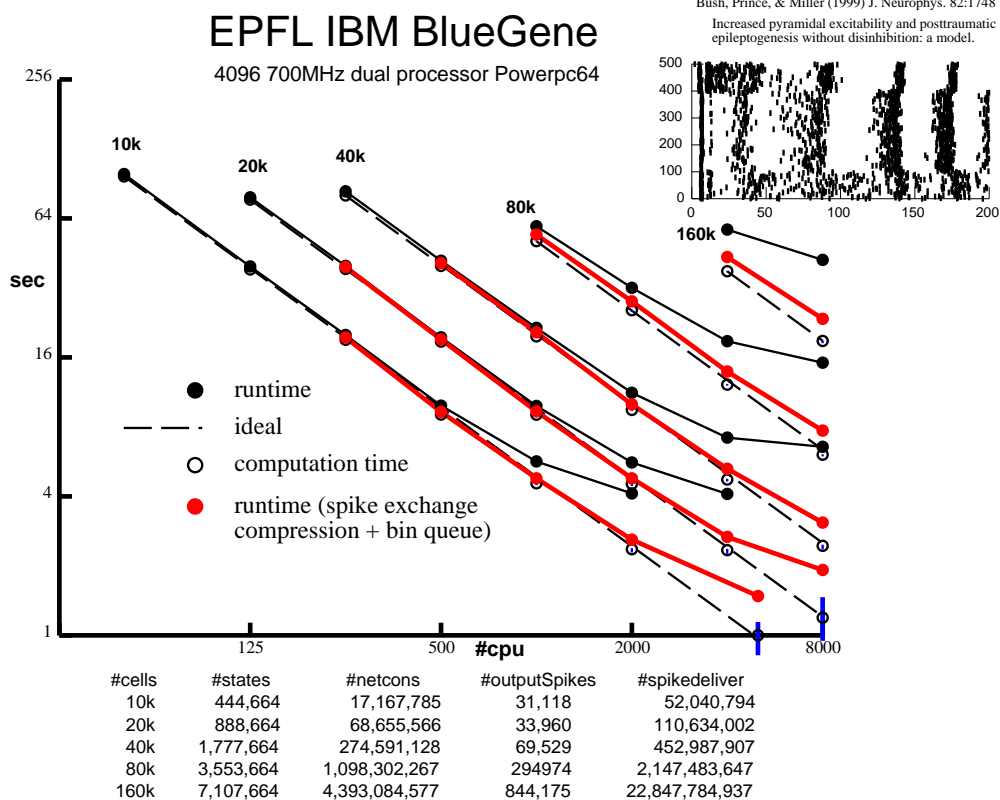
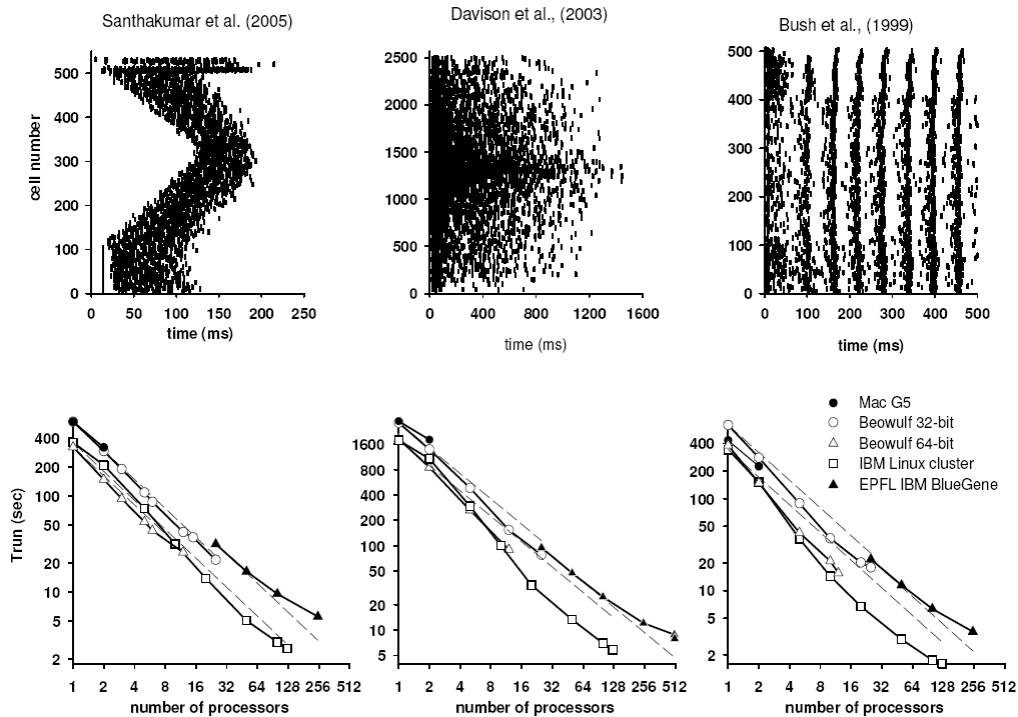
pc.set_maxstep() uses
MPI_Allreduce
to determine minimum delay.



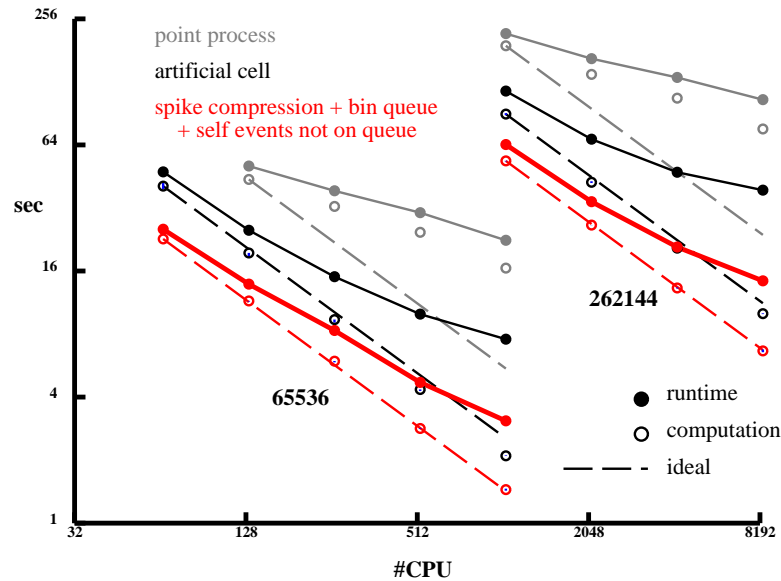




Migliore et al (2006) J. Comput. Neurosci. 21(2):119



Artificial Spiking Net Performance

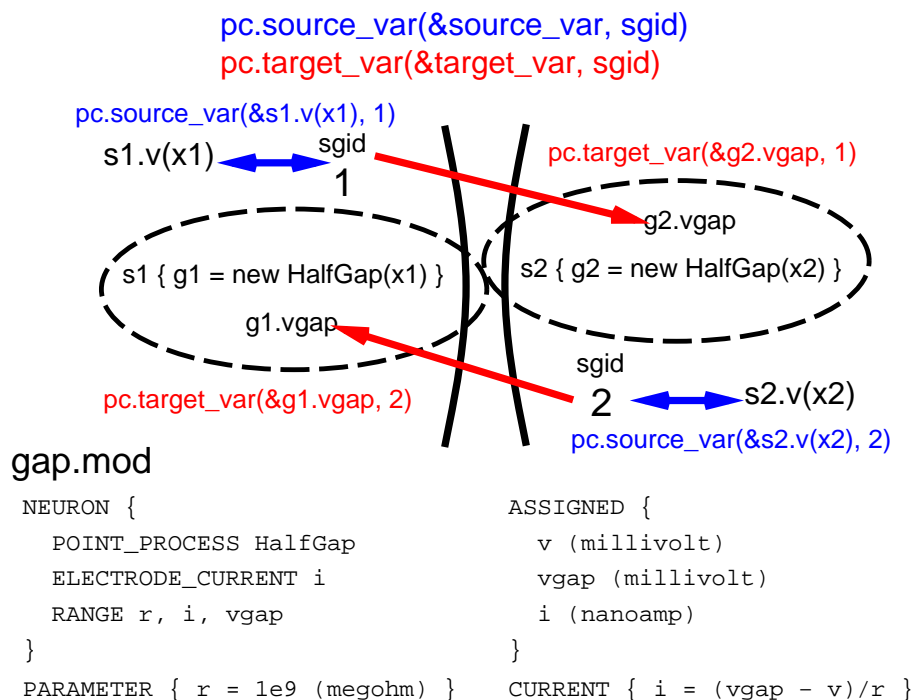


Each cell fires randomly every 10 to 20 ms.
65K cells, 1000 random connections per cell
256K cells, 10,000 random connections per cell

tstop = 200(ms)
delay = 1(ms)
weight = 0

Gap Junction Specification

Continuous Voltage Exchange

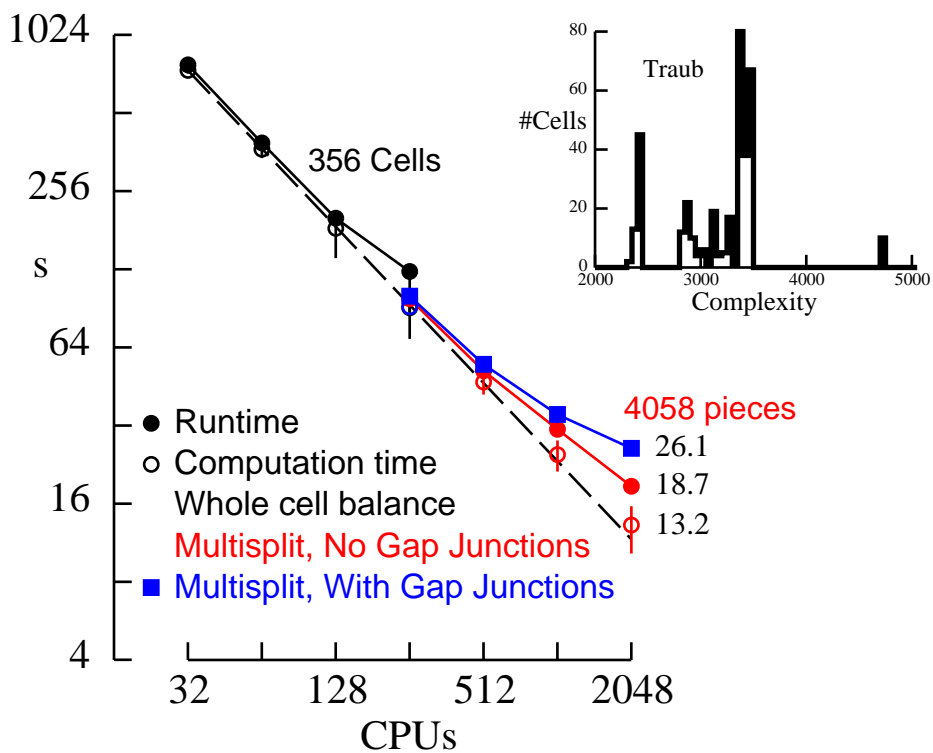
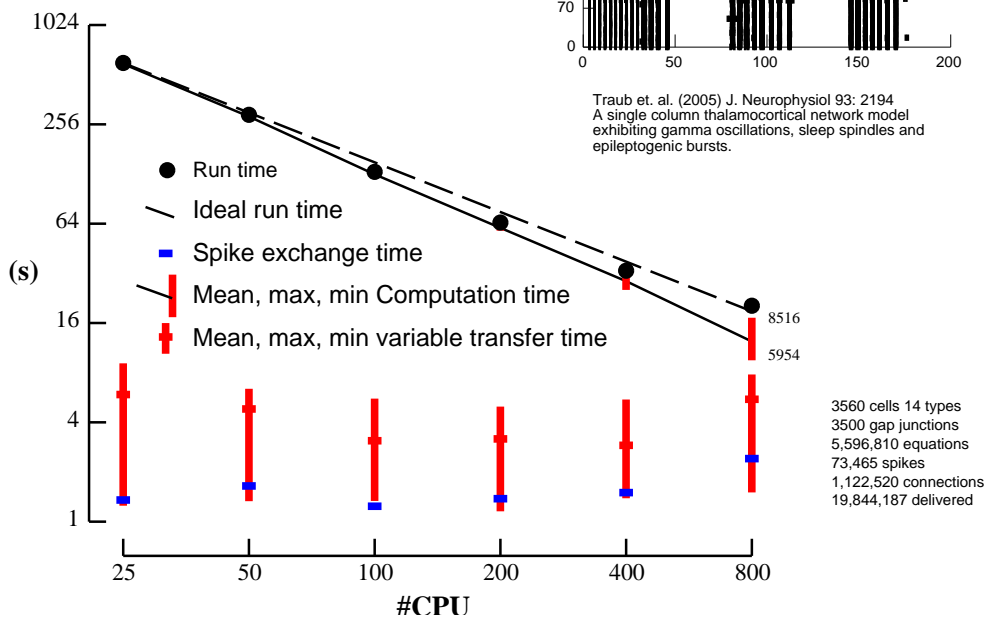


Pittsburgh Supercomputing Center

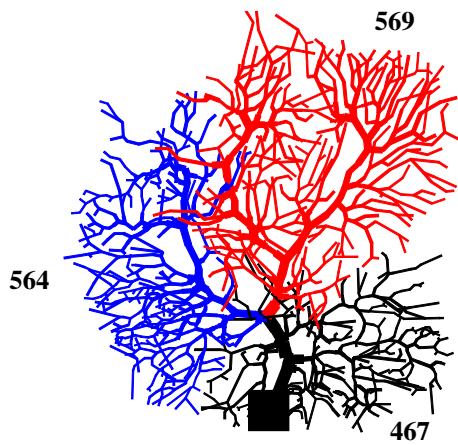
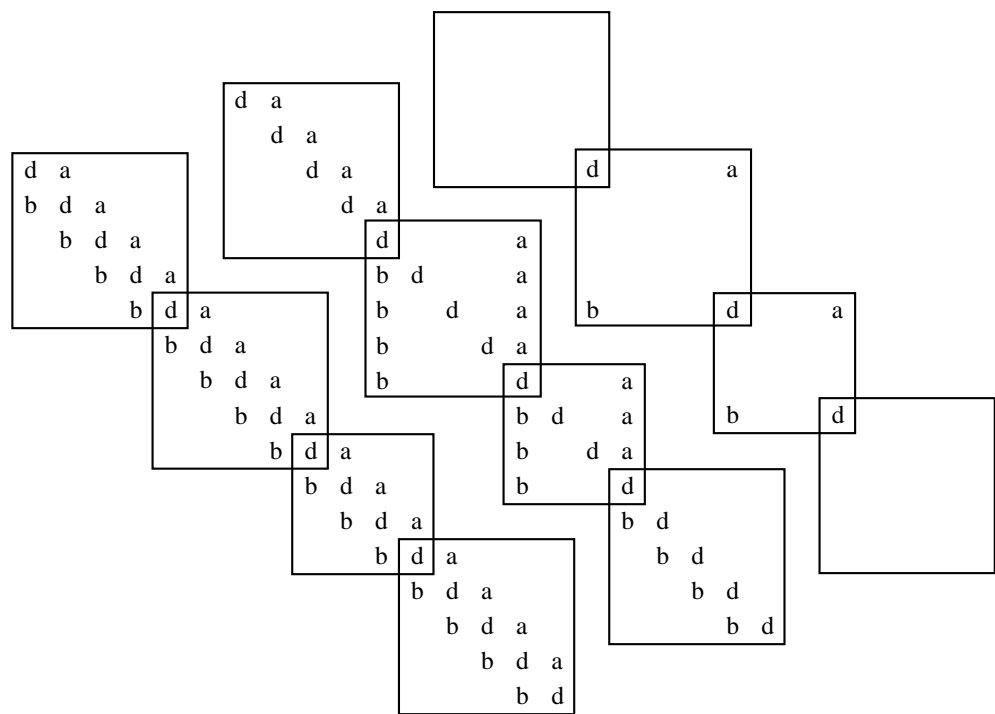
Bigben

Cray XT3

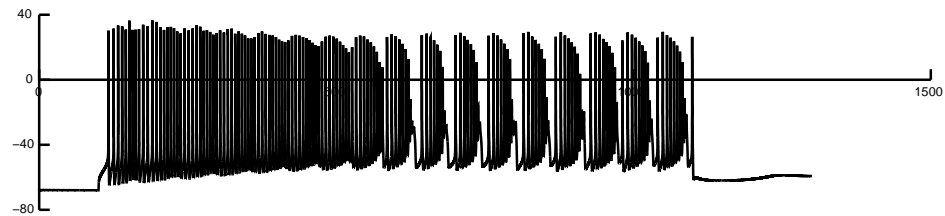
2068 2.4 GHz Opteron Processors



Multisplit Gaussian Elimination

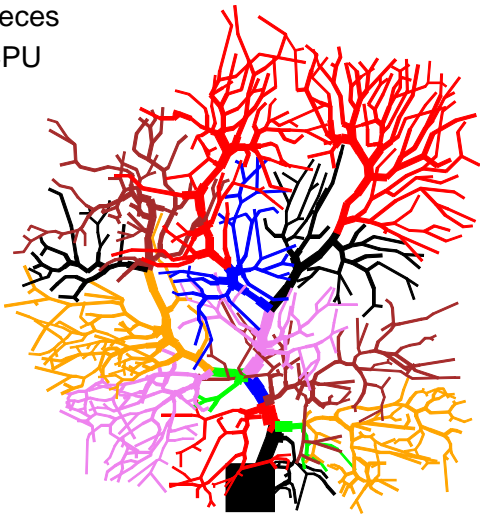


#CPU	Runtime (s)		
1	54.8		
3	22.2	19.5 expected	18.3 ideal



De Schutter & Bower (1994) J. Neurophysiol 71:375
Ported to NEURON by Jenny Davie, Volker Steuber, and Arnd Roth.

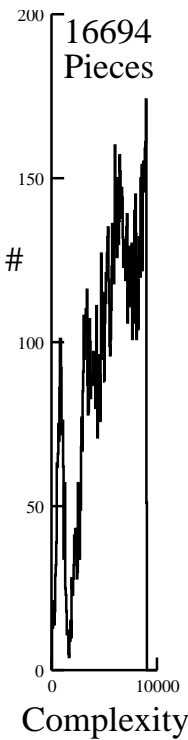
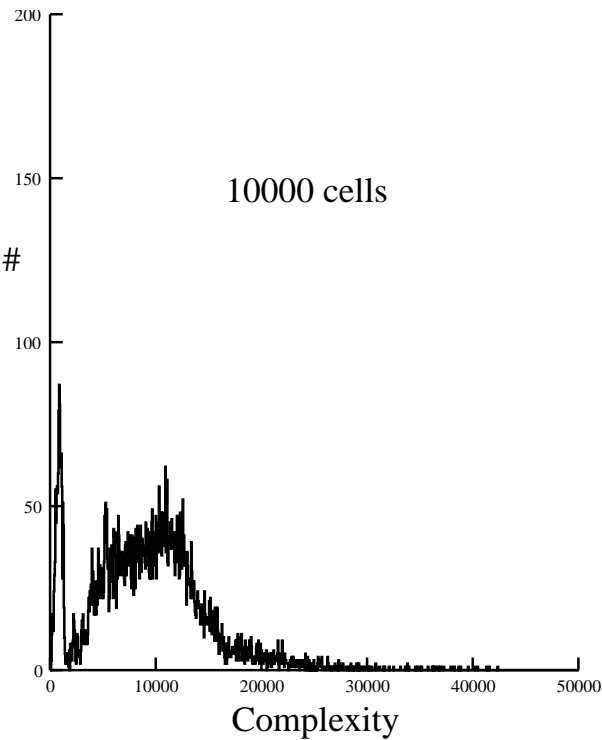
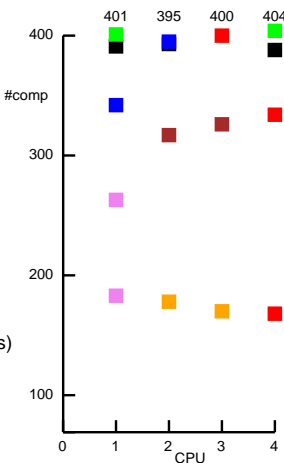
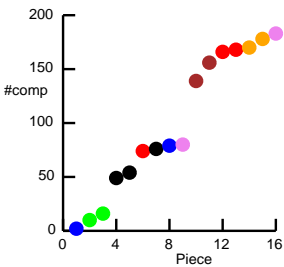
16 Pieces
4 CPU

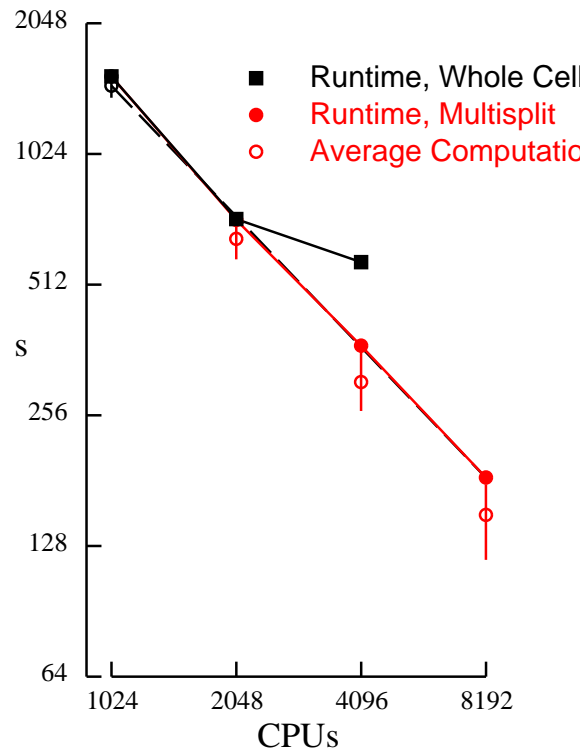


Time (s)			
CPU	Computation		
0	13.82	0.56	
1	13.35	1.03	16 pieces, 1 cpu
2	13.47	0.90	wholecell, 1 cpu
3	13.56	0.82	16 pieces, 4 cpu

Runtime(s)

55.0
56.2
14.4





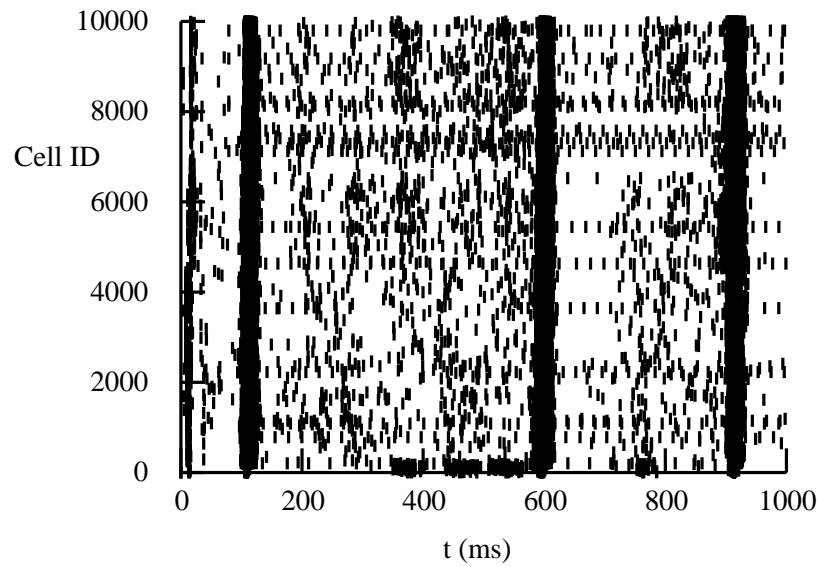
Results must be independent of
 Number of processors
 Distribution of cells

What about RANDOM?
 Reproducible
 Independent
 Restartable

Associate a random stream with a cell.

Use cryptographic transformation of several integers.
 run number
 stream number (cell gid)
 stream pick index

PatternStim



out.spk (58,858 lines)

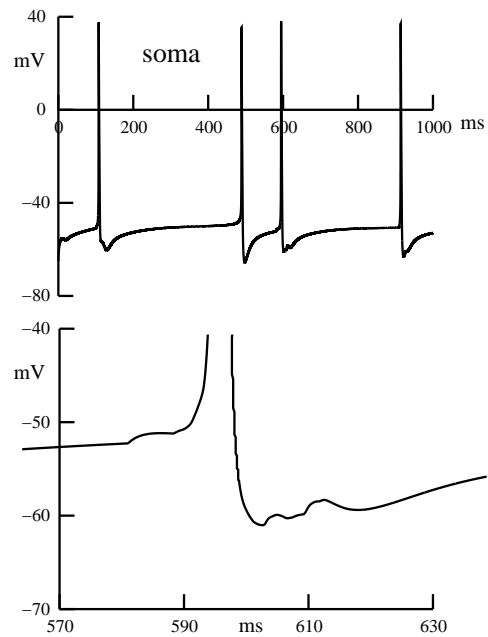
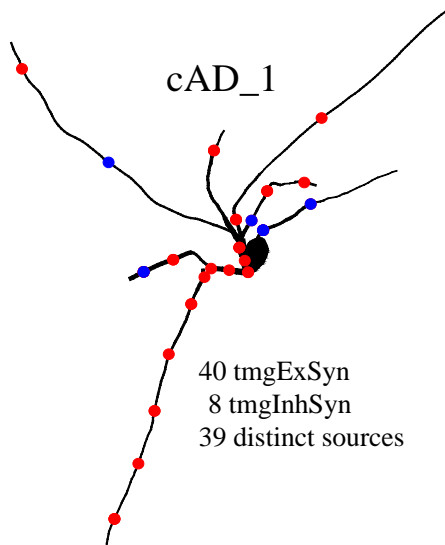
3.975 8050

3.975 8621

...

999.125 4632

Line#	spike(ms)	gid
8548:	107.25	1
18501:	488.625	1
27276:	594.25	1
51470:	913.475	1



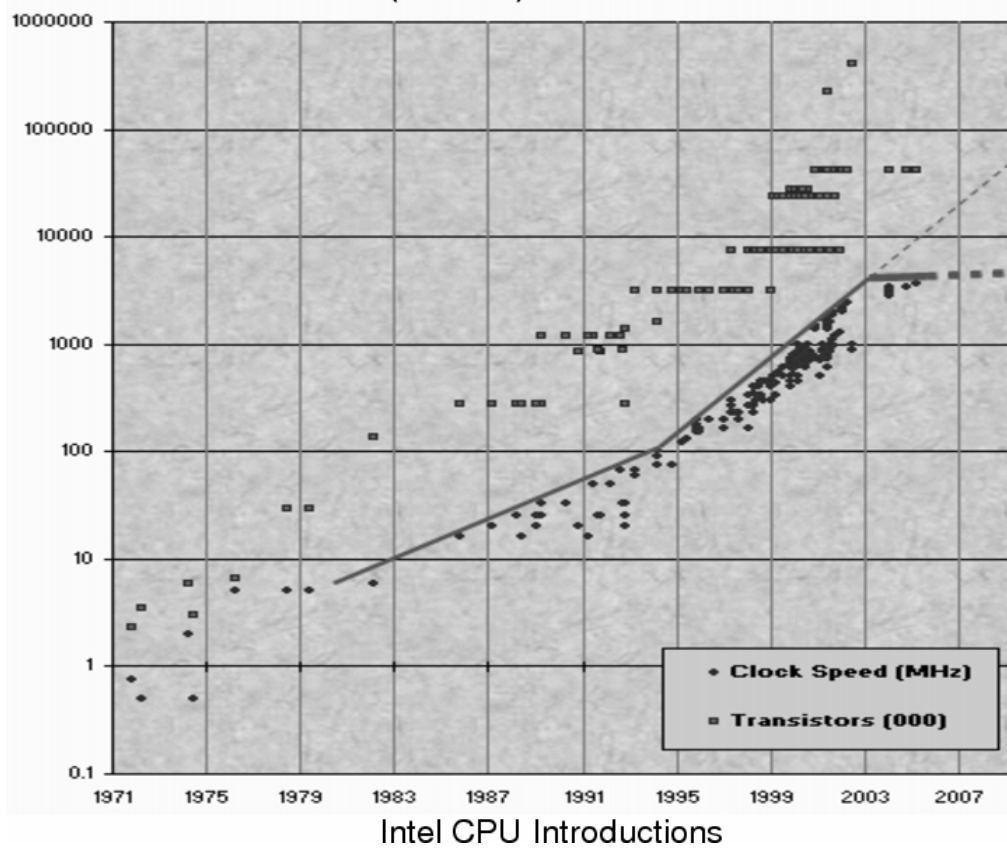
Debugging

- 1) GID and time of first spike difference.
- 2) All spikes delivered to synapses of that Cell?
- 3) When and what is the first state difference?

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 ...
}
```

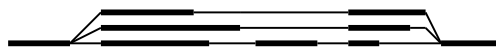


We never use.

Condition Wait

```
multithread_job(run)
```

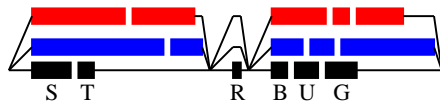
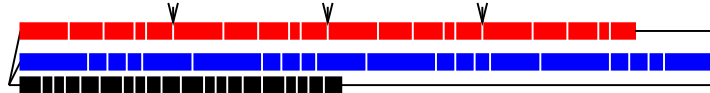
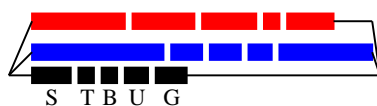
```
run(NrnThread* nt) {
  while(t < tstop) {
    step(nt)
    barrier()
    if (nt->id == 0) { plot() }
    barrier()
  }
}
```



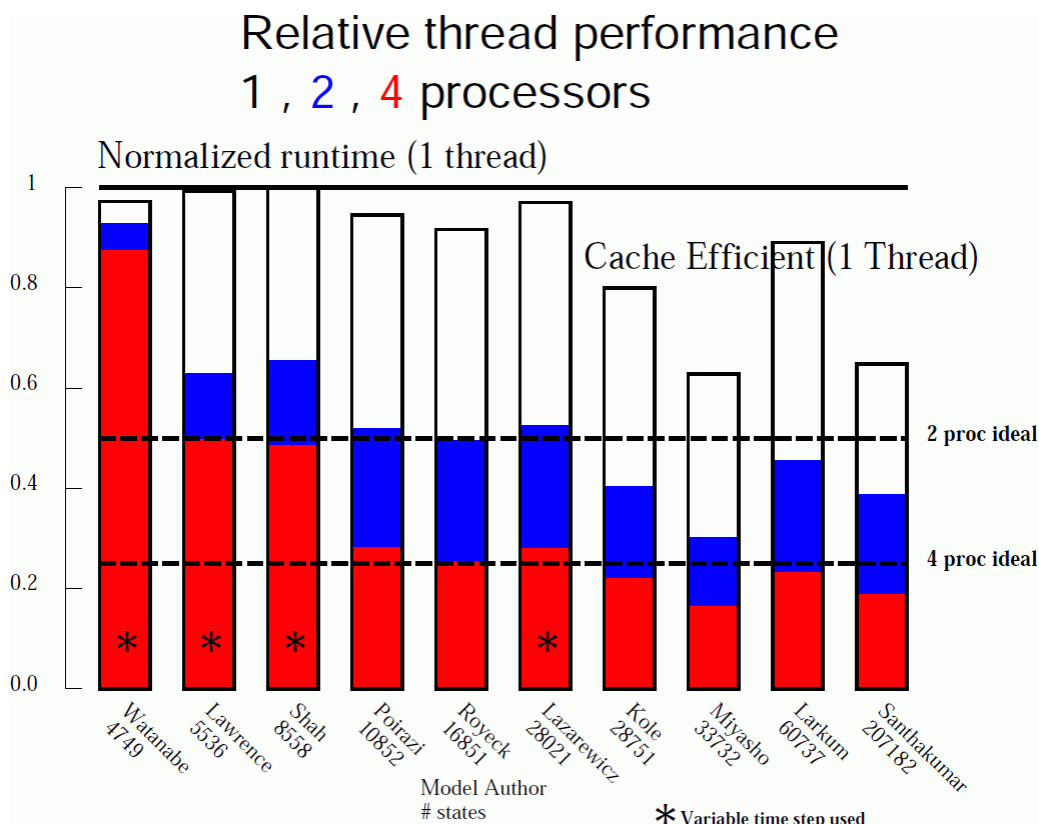
Reminiscent of MPI

Fixed step: $t \rightarrow t + dt$

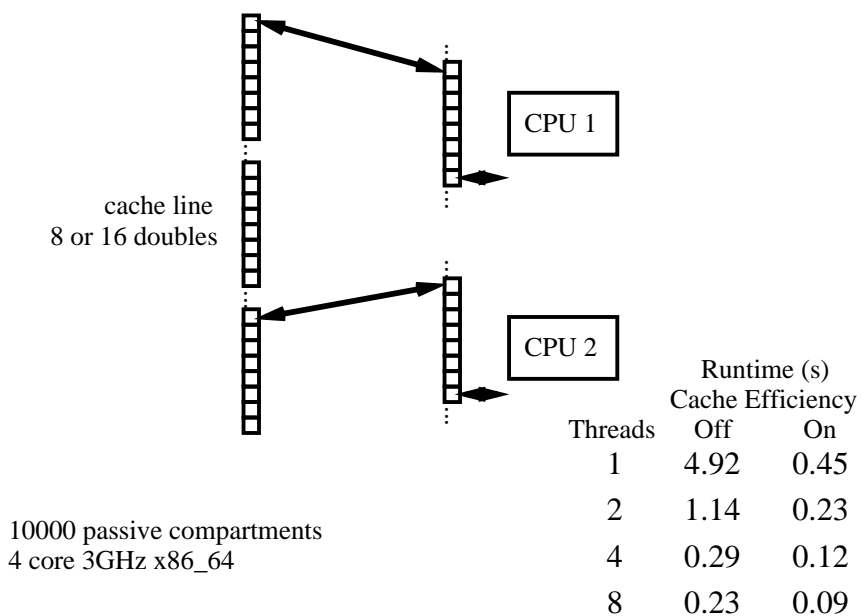
S	T	R	B	U	G
setup	triang	reduce solve	bksub	update	cond gates



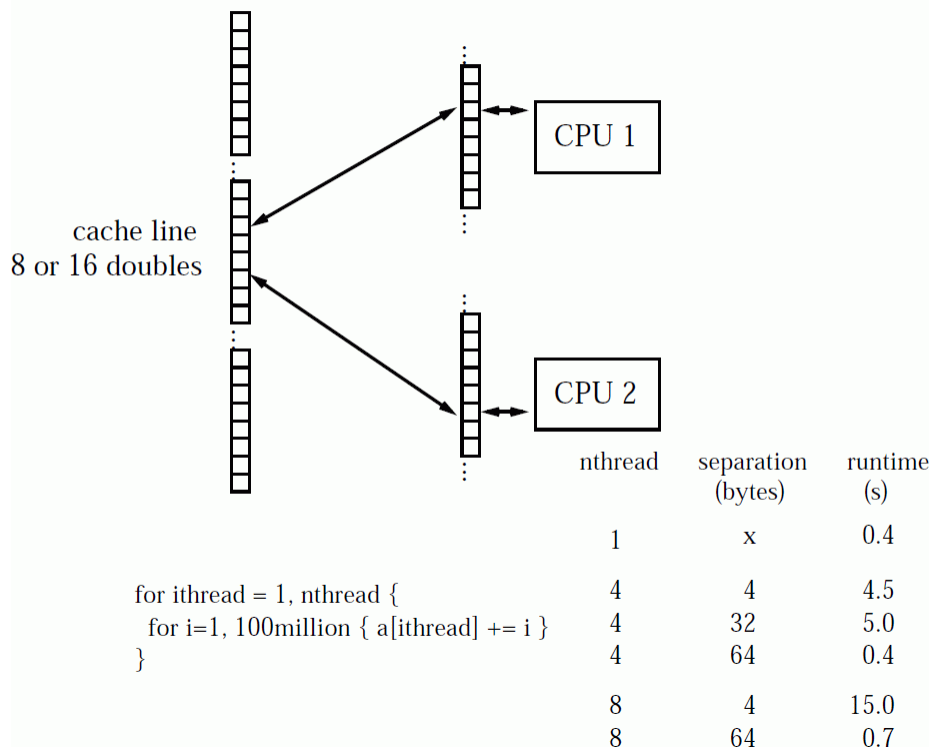
Global var dt $y' = f()$ dy'/dy
27 ||Vector operations



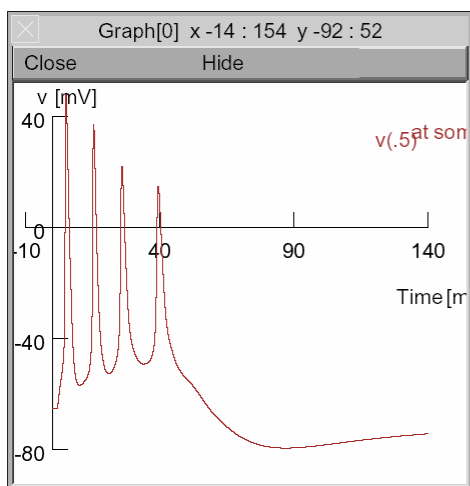
Ideal cache efficiency



False cache line sharing



Lazarewicz 2002, CA3 Pyramidal Neuron



RunControl

Close Hide

Init (mV) -65

Init & Run

Stop

Continue til (ms) 5

Continue for (ms) 1

SingleStep

t (ms) 140

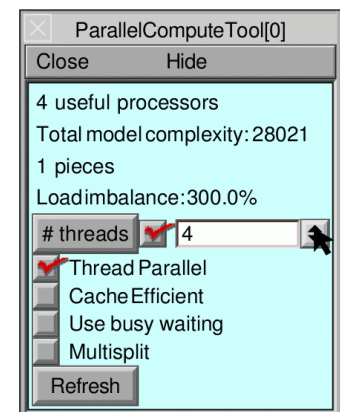
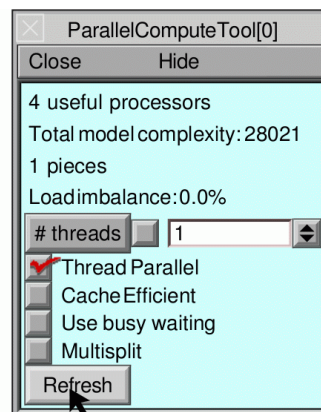
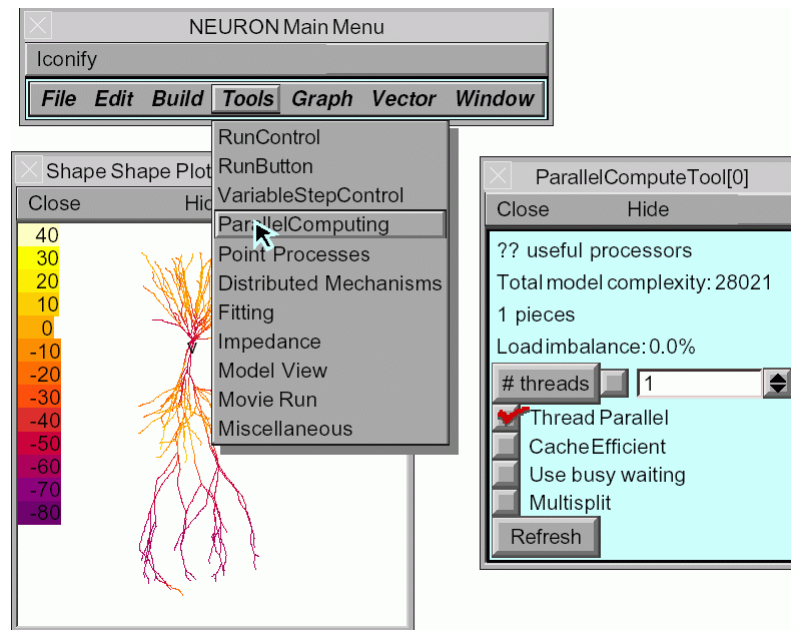
Tstop (ms) 140

dt (ms) ☒ 2.335

Points plotted/ms 40

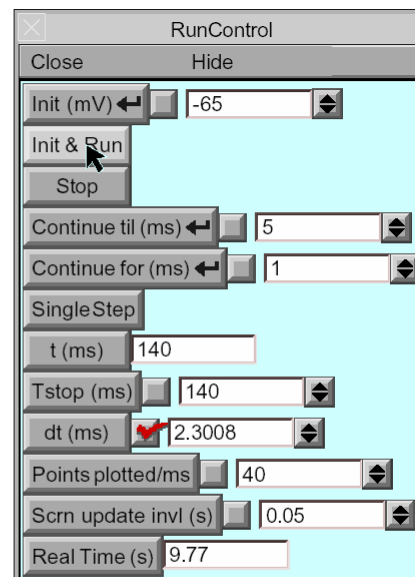
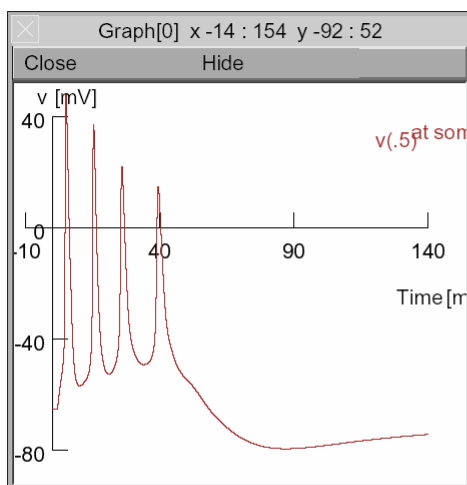
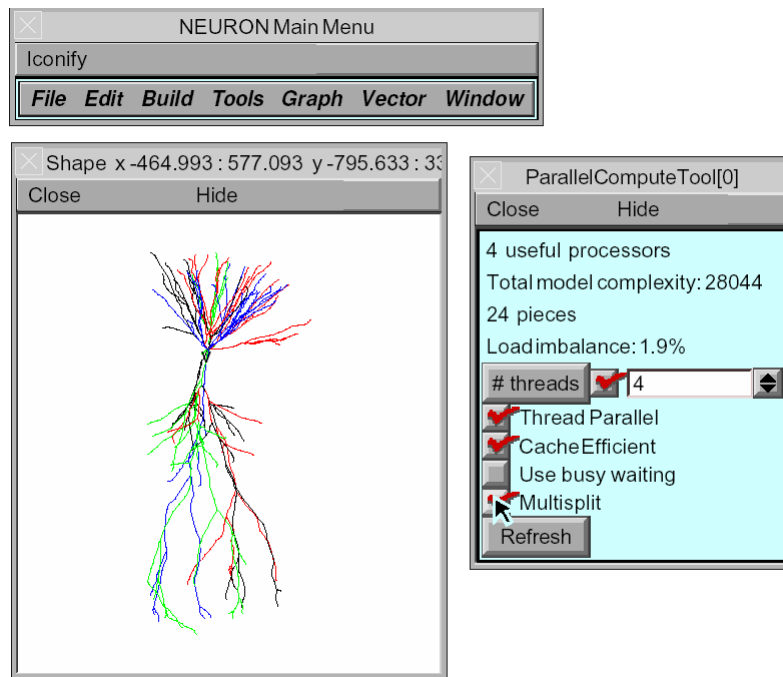
Scrn update invl (s) 0.05

Real Time (s) 35.4



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)
```


Initialization, broadly speaking:

We want to get the same result every time we click on
Init & Run, no matter what we did before

Note: this presentation explicitly omits details of initialization
of ionic concentrations and equilibrium potentials

Copyright © 1998–2003 N.T. Carnevale and M.L. Hines, all rights reserved

The NEURON Simulation Environment

Slide 2

Initialization should assign values at $t = 0$ for

- membrane potential
- gating states
- ionic concentrations
- chemical kinetic states
- voltage across capacitors in linear circuits
- internal states of op amps
- random number generators

and properly configure

- event queues
- vector record and play
- counters

Copyright © 1998–2003 N.T. Carnevale and M.L. Hines, all rights reserved

NEURON's `finitialize()`

- sets `t = 0`
- clears event queue
- sets up internal data structures that depend on topology and geometry
- initializes `Vector.play` controller
- delivers events whose delivery time is 0
- if `finitialize` was called with `v_init` argument, sets `v` in all compartments to `v_init`
- calls `INITIAL` block of every inserted mechanism in every segment
- if `extracellular` is used, sets `vext` to 0
- initializes ions; calculates equilibrium potentials if necessary
- initializes mechanisms that `WRITE` ion concentrations; recalcs equilib potentials as needed
- calls all other `INITIAL` blocks
- initializes `LinearMechanism` states
- calls `INITIAL` blocks inside `NET_RECEIVE` blocks; if this spawns network events, delivers any whose delay is 0 to their target `NET_RECEIVE` blocks
- if fixed time step integrator is used, calls all `BREAKPOINT` blocks
- initializes adaptive integrator (if being used)
- initializes any `ccode.record` and `vector.record` recordings

Copyright © 1998–2003 N.T. Carnevale and M.L. Hines, all rights reserved

Default initialization: the standard run library

`nrn/share/nrn/lib/hoc/stdrun.hoc`
 (MSWin: `c:\nrn\lib\hoc\stdrun.hoc`)

`stdinit()`

Called when you

click on `Init` or `Init & Run` in the `RunControl`

or

enter a new value for `v_init` in the `Init` button's field editor

```
proc stdinit() {
    realtime=0 // "run time" in seconds
    startsw()  // initialize run time stopwatch
    setdt()
    init()
    initPlot()
}
```

Copyright © 1998–2003 N.T. Carnevale and M.L. Hines, all rights reserved

init()

Most customizations are made here

```

proc init() {
  finitialize(v_init)
  // User-specified customizations go here.
  // If this invalidates the initialization of
  // variable dt integration and vector recording,
  // uncomment the following code.
  /*
  if (cnode.active()) {
    cnode.re_init()
  } else {
    fcurrent()
  }
  frecord_init()
  */
}

```

Copyright © 1998–2003 N.T. Carnevale and M.L. Hines, all rights reserved

INITIAL blocks in NMODL**HH-like mechanisms**

```

PROCEDURE rates(v(mv)) {
  minf = alpha(v)/(alpha(v) + beta(v))
  . . .
}
. . .

INITIAL {
  rates(v)
  m = minf
  . . .
}

```

Copyright © 1998–2003 N.T. Carnevale and M.L. Hines, all rights reserved

Kinetic schemes

```

INITIAL {
    SOLVE scheme METHOD steadystate
}
e.g.
NEURON {
    USEION k READ ek WRITE ik
}
STATE { c1 c2 o }
INITIAL {
    SOLVE scheme METHOD steadystate
}
BREAKPOINT {
    SOLVE scheme METHOD sparse
    ik = gbar*o*(v - ek)
}
KINETIC scheme {
    rates(v) : calculate the 4 k rates.
    ~ c1 <-> c2 (k12, k21)
    ~ c2 <-> o ( k2o, ko2)
}

```

Copyright © 1998–2003 N.T. Carnevale and M.L. Hines, all rights reserved

Default initialization of STATES

Use state0, e.g.

```

PARAMETER {
    state0 = 1
}

```

or alternative syntax

```

STATE {
    state START 1
}

```

It's best to be explicit

```

INITIAL {
    m = m0
    h = h0
}

```

To make them visible from hoc

```

NEURON {
    GLOBAL m0
    RANGE h0
}

```

Copyright © 1998–2003 N.T. Carnevale and M.L. Hines, all rights reserved

Typical custom initializations

Steady state

unperturbed system

system under constant voltage or current clamp

Defined starting point on a trajectory
of an oscillating or chaotic system

Adjust parameters to meet some condition

How?

Use a custom `init()` procedure.

Load after the standard library, so it won't be overwritten.

Copyright © 1998–2003 N.T. Carnevale and M.L. Hines, all rights reserved

Initializing to steady state

"Travel into the past," take large steps with implicit Euler, then return to the present.

```
proc init() { local dtsav, temp
  finitialize(v_init)
  t = -1e10
  dtsav = dt
  dt = 1e9
  // if ccode is on, turn it off to do large fixed step
  temp = ccode.active()
  if (temp!=0) { ccode.active(0) }
  while (t<-1e9) {
    fadvance()
  }
  // restore ccode if necessary
  if (temp!=0) { ccode.active(1) }
  dt = dtsav
  t = 0
  if (ccode.active()) {
    ccode.re_init()
  } else {
    fcurrent()
  }
  frecord_init()
}
```

Copyright © 1998–2003 N.T. Carnevale and M.L. Hines, all rights reserved

Initializing to a desired state

Especially useful for oscillating or chaotic models.

Run a "warmup simulation," then save all states

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

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

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

To read from a file

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

Copyright © 1998–2003 N.T. Carnevale and M.L. Hines, all rights reserved

Initializing to a desired state continued

A custom init() that restores saved states

```
proc init() {
  finitialize(v_init)
  svstate.restore()
  t = 0 // t is one of the "states"
  if (cnode.active()) {
    cnode.re_init()
  } else {
    fcurrent()
  }
  frecord_init()
}
```

Copyright © 1998–2003 N.T. Carnevale and M.L. Hines, all rights reserved

Initializing to a particular resting potential

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

Example: for a single compartment model with hh,

```
proc init() {
  finitialize(v_init)
  el_hh = (ina + ik + gl_hh*v)/gl_hh
  if (cnode.active()) {
    cnode.re_init()
  } else {
    fcurrent()
  }
  frecord_init()
}
```

Copyright © 1998–2003 N.T. Carnevale and M.L. Hines, all rights reserved

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

Example:

```
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()`

Copyright © 1998–2003 N.T. Carnevale and M.L. Hines, all rights reserved

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

```
proc init() {  
    finitialize(-65)  
    ic_constant = -(ina + ik + il_hh)  
    if (cnode.active()) {  
        cnode.re_init()  
    } else {  
        fcurrent()  
    }  
    frecord_init()  
}
```

Copyright © 1998–2003 N.T. Carnevale and M.L. Hines, all rights reserved

The NEURON Simulation Environment

Figure 1

NEURON's tools for Electrotonic Analysis

Input and transfer impedances

Voltage transfer ratio

$$V_{\text{downstream}} / V_{\text{upstream}}$$

Electrotonic transformation

$$\log(V_{\text{downstream}} / V_{\text{upstream}})$$

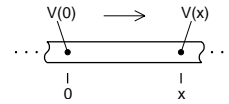
... all as functions of frequency and space

Copyright © 1998–2001 N.T. Carnevale and M.L. Hines, all rights reserved

The NEURON Simulation Environment

Figure 2

CLASSICAL CABLE THEORY



Infinite cylinder in the steady state

$$V(x) = V(0) e^{-x/\lambda}$$

$x \equiv$ physical distance

$\lambda \equiv$ length constant

Classical Definition of Electrotonic Distance:

$$X = \ln V(0) / V(x) = x / \lambda$$

$$\therefore \text{attenuation } A^V(x) = V(0) / V(x) = e^{x/\lambda}$$

Intuitively simple

Copyright © 1998–2001 N.T. Carnevale and M.L. Hines, all rights reserved

The NEURON Simulation Environment

Figure 3

BUT neurons \neq infinite cylinders

Attempted fix: reduce dendritic tree to an equivalent cylinder of finite length

Finite cylinder in the steady state

$$A^V(x) = \cosh L_{\text{classical}} / \cosh(L_{\text{classical}} - X)$$

$L_{\text{classical}} \equiv$ physical length of cylinder / λ

$$X \equiv x / \lambda$$

Copyright © 1998–2001 N.T. Carnevale and M.L. Hines, all rights reserved

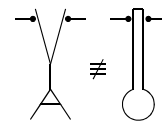
The NEURON Simulation Environment

Figure 4

The good and the bad news about the equivalent cylinder approximation

The bad news:

- ◆ Neither intuitive nor simple
- ◆ Destroys the spatial relationships among synaptic inputs



- ◆ Classical electrotonic distance $X \equiv x / \lambda$ fosters conceptual error because it obscures the direction-dependence of attenuation in finite structures

Copyright © 1998–2001 N.T. Carnevale and M.L. Hines, all rights reserved

The NEURON Simulation Environment

Figure 5

The good news: it's not valid either

Property	Assumption	Truth
Dendritic terminations	electrically equidistant from soma	varies widely
Diameters	cylindrical	irregular
Branch points	3/2 power criterion	no

$$d_p^{3/2} = \sum d_d^{3/2}$$

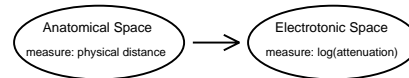
Copyright © 1998–2001 N.T. Carnevale and M.L. Hines, all rights reserved

The NEURON Simulation Environment

Figure 6

Alternative: a transformation from anatomical to electrotonic space that

- ✓ is intuitive
- ✓ is empirically-based
- ✓ makes no restrictive assumptions about anatomy



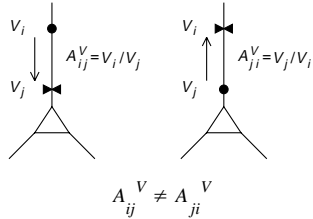
Copyright © 1998–2001 N.T. Carnevale and M.L. Hines, all rights reserved

The NEURON Simulation Environment

Figure 7

Foundation of this approach:
two-port analysis of electrotonus

How well do signals propagate?



Signal transfer is direction-dependent

Attenuation identities

$$A_{ij}^V = A_{ji}^I \quad A_{ij}^I = A_{ji}^Q$$

Copyright © 1998–2001 N.T. Carnevale and M.L. Hines, all rights reserved

The NEURON Simulation Environment

Figure 8

The Electrotonic Transformation

Functional definition of electrotonic distance

$$L = \log(\text{attenuation})$$

- ✓ simple, direct relationship to attenuation

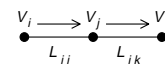
- ✓ direction-dependent: $L_{ij}^V = \ln(A_{ij}^V)$,

$$L_{ji}^V = \ln(A_{ji}^V), \text{ and in general } L_{ij}^V \neq L_{ji}^V$$

Each physical segment ij of a cell has **two** representations in electrotonic space—one for each direction of propagation!

- ✓ identical to classical electrotonic distance for an infinite cylinder

- ✓ additive over a path with a consistent direction of propagation



$$A_{ik} = \frac{V_i}{V_k} = \frac{V_i}{V_j} \frac{V_j}{V_k} = A_{ij} A_{jk}$$

$$\therefore L_{ik} = L_{ij} + L_{jk}$$

Copyright © 1998–2001 N.T. Carnevale and M.L. Hines, all rights reserved

The NEURON Simulation Environment

Figure 9

Using the Electrotonic Transformation

At each frequency of interest:

Step 1: Transform from anatomical to electrotonic space

- Compute attenuations between points of interest
- Map into electrotonic space (log)

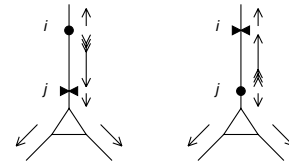
Copyright © 1998–2001 N.T. Carnevale and M.L. Hines, all rights reserved

The NEURON Simulation Environment

Figure 10

Step 2: Render graphically with respect to a reference point
(because attenuation is direction-dependent)

- A convenient reference: the soma
- Changing the reference point location alters only the direction of signal flow on the direct path between the old and new locations.



Therefore somatocentric renderings of the transform can be rearranged to generate the renderings for any other reference location.

- The attenuation identities give us the transform identities

$$V_{in} = I_{out} = Q_{out} \text{ and } V_{out} = I_{in} = Q_{in}$$

Copyright © 1998–2001 N.T. Carnevale and M.L. Hines, all rights reserved

The NEURON Simulation Environment

Figure 11

Q: How does somatic peak PSP amplitude depend on synaptic location?

A?: A_{in}^V (voltage attenuation) or $k_{syn \rightarrow soma}$ (synapse to soma voltage transfer ratio)?

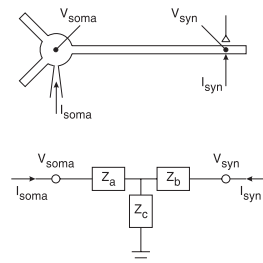
Time-honored and **wrong!**

- ◆ assumes synapses act like voltage sources.
- ◆ real synapses act more like current sources (Jaffe & Carnevale 1999).

Copyright © 1998–2001 N.T. Carnevale and M.L. Hines, all rights reserved

The NEURON Simulation Environment

Figure 12



Modified from Fig. 1 in Jaffe & Carnevale 1999.

If synapse \approx voltage source, then

- $V_{syn}(t) \approx$ independent of synaptic location

and

- Synapse to soma voltage transfer ratio

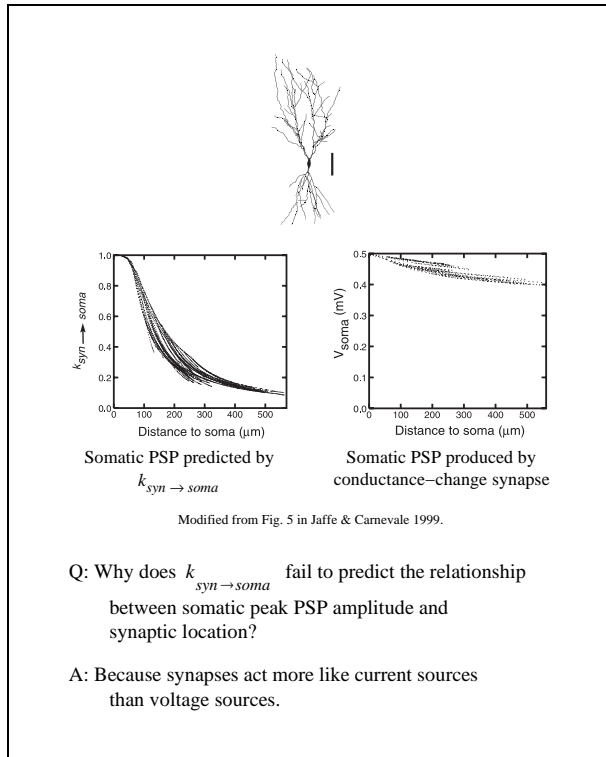
$$k_{syn \rightarrow soma} = 1/A_{in}^V = Z_c / (Z_b + Z_c)$$

predicts the variation of somatic PSP amplitude with synaptic location

Copyright © 1998–2001 N.T. Carnevale and M.L. Hines, all rights reserved

The NEURON Simulation Environment

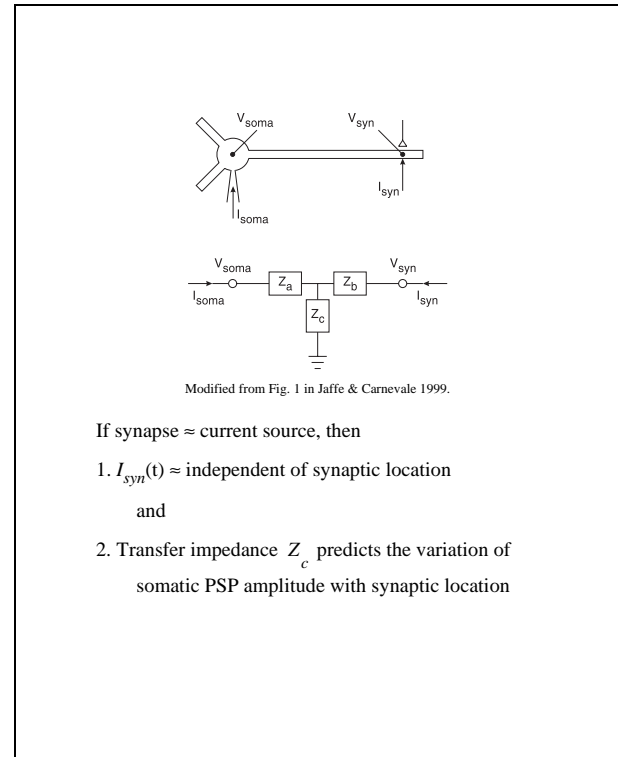
Figure 13



Copyright © 1998–2001 N.T. Carnevale and M.L. Hines, all rights reserved

The NEURON Simulation Environment

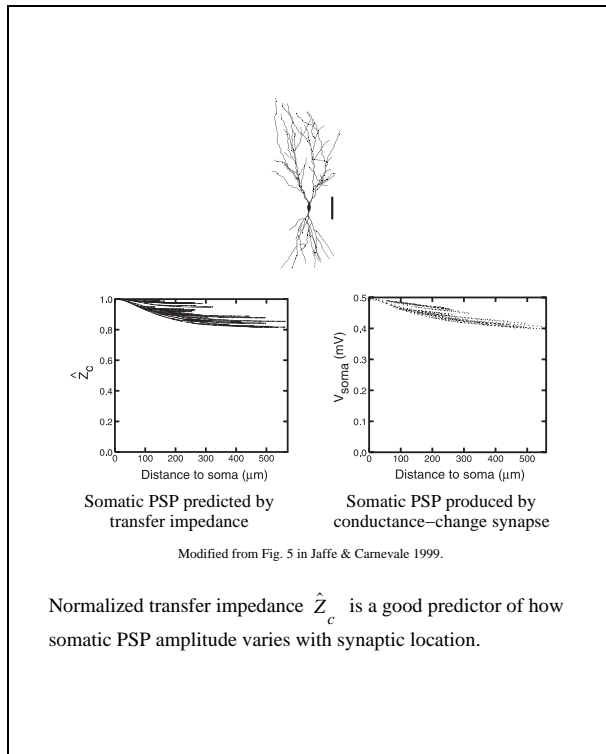
Figure 14



Copyright © 1998–2001 N.T. Carnevale and M.L. Hines, all rights reserved

The NEURON Simulation Environment

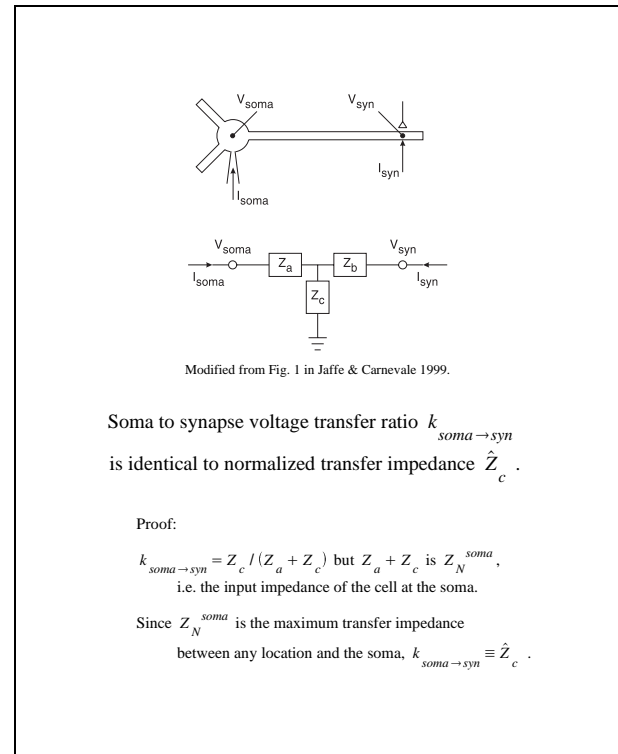
Figure 15



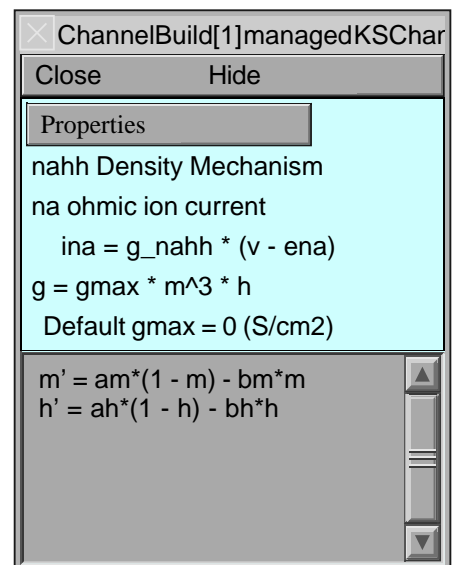
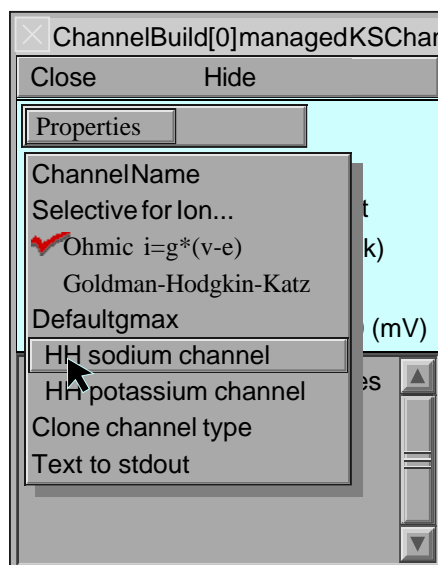
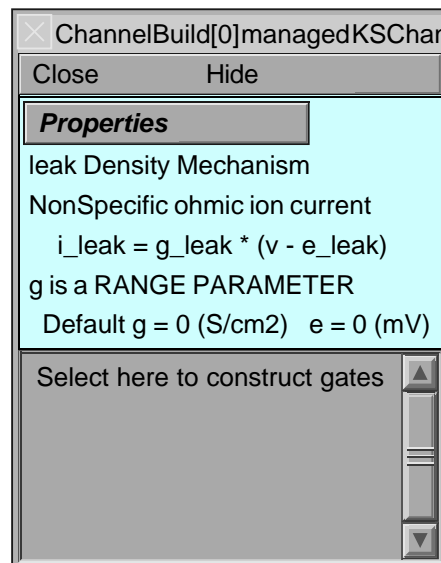
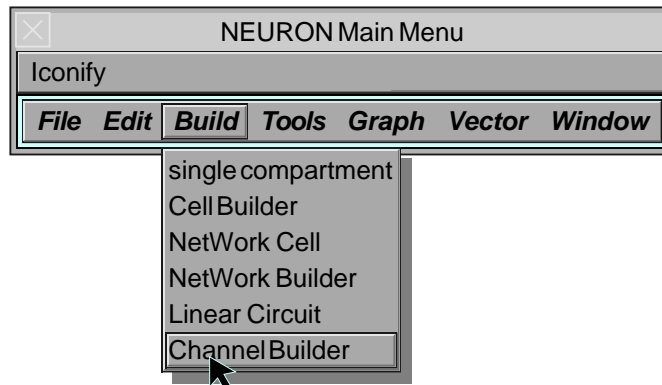
Copyright © 1998–2001 N.T. Carnevale and M.L. Hines, all rights reserved

The NEURON Simulation Environment

Figure 16



Copyright © 1998–2001 N.T. Carnevale and M.L. Hines, all rights reserved



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

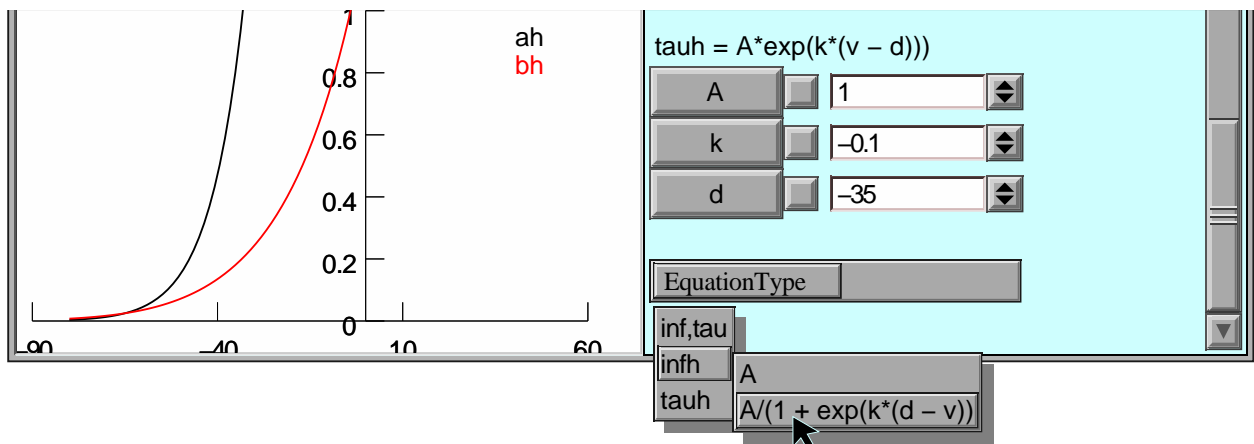
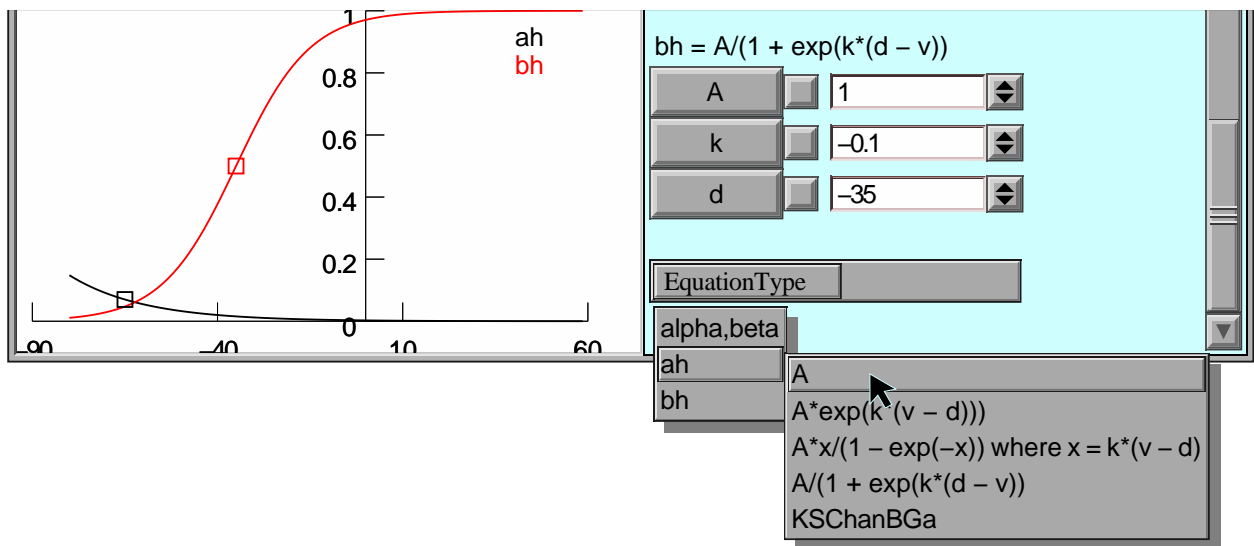
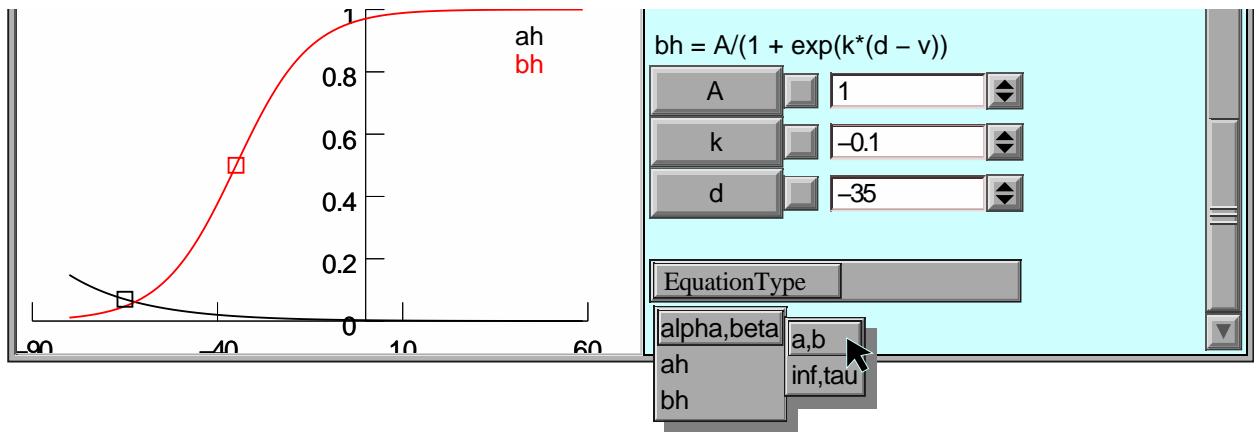
Defaultgmax

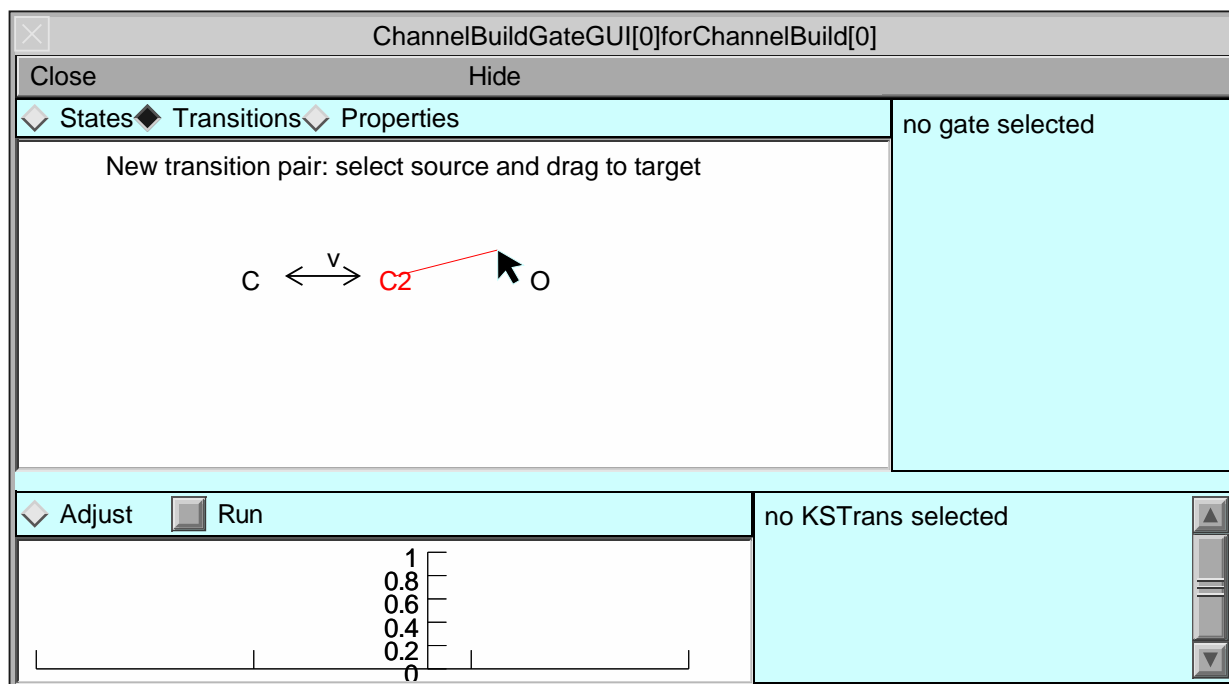
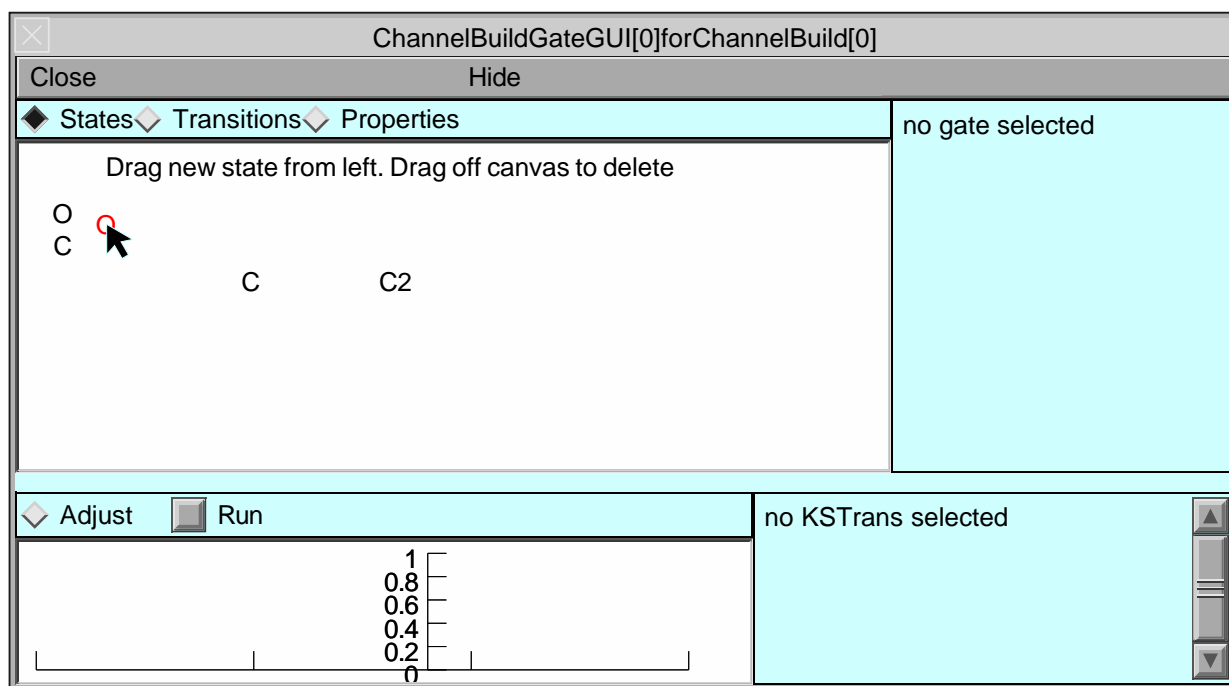
HH sodium channel

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

Fractional Conductance

C2 fraction

O fraction

Adjust Run

1
0.8
0.6
0.4
0.2
0

aC2O
bC2O

bC2O = A

A

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 displays two windows from the NEURON simulation environment. The top window, titled "ChannelBuild[0]managedKSChan[0]", shows the "Properties" tab with the following information:

- leak Density Mechanism
- NonSpecific ohmic ion current
- $i_{\text{leak}} = g_{\text{leak}} * (v - e_{\text{leak}})$
- $g = g_{\text{max}} * O * O2 * O3$
- Default $g_{\text{max}} = 0$ (S/cm2) $e = 0$ (mV)
- O: 3 state, 2 transitions
- O2: 2 state, 1 transitions
- $O3' = aO3 * (1 - O3) - bO3 * O3$

The bottom window, titled "ChannelBuildGateGUI[0]forChannelBuild[0]", shows the "States" tab. It displays a state transition diagram with states O, C, C2, O2, and C3. The transitions are labeled with voltage (v). The state O3 is highlighted in red. A dialog box titled "nrniv" is open, showing "Change state name" with "O3" in the input field and "Accept" and "Cancel" buttons. The bottom of the window has "Adjust" and "Run" buttons, and a status bar showing "no KSTrans selected".

The screenshot displays the "ChannelBuild[0]managedKSChar" window. The "Properties" tab is active, showing a list of channel types. The "Ohmic $i = g * (v - e)$ " option is selected with a red checkmark. Other options include "Goldman-Hodgkin-Katz", "HH sodium channel", "HH potassium channel", "Clone channel type", and "Text to stdout". A "Create new type" button is visible on the right side of the list.

