Schedule of Presentations

NTC	Ted Carnevale
RAM	Robert McDougal

Morning session

Time	Speaker	Title	Page
9:00 AM	NTC	Welcome	3
9:05	NTC	NEURON: a brief tour	5
	NTC	The basics	9
	NTC	Why the GUI?	19
	NTC	Construction and use of models	23
	NTC	Using the CellBuilder to make a stylized model	24
	NTC	Creating and using an interface for running simulations	36
10:15	NTC	The Linear Circuit Builder	47
10:30	Coffee Break		
10:45	NTC	Using NMODL to add new biophysical mechanisms	55
11:15	NTC	Numerical methods: accuracy, stability, speed	63
11:30 AM	NTC	Networks: spike-triggered synaptic transmission, events, and artificial spiking cells	69
12:14:59 PM	NTC	At last: how to start and stop NEURON	79
12:15 PM	Lunch		

Afternoon session

1:15 PM	RAM	Numerical methods: adaptive integration	81
1:30	RAM	NEURON with Python	87

Online references and supplementary materials:
NEURON's home page `https://www.neuron.yale.edu/neuron/` has links to:

- **Download page** `https://www.neuron.yale.edu/neuron/download`
- **Documentation page** `https://www.neuron.yale.edu/neuron/docs`
- **Programmer's Reference**
  Python `https://www.neuron.yale.edu/neuron/static/py_doc/index.html`
  hoc `https://www.neuron.yale.edu/neuron/static/new_doc/index.html`
- **NEURON Forum** `https://www.neuron.yale.edu/phpBB/`
- **Publications** `https://neuron.yale.edu/neuron/publications` which include articles that used NEURON and articles about NEURON

We value your opinions and suggestions for improving this course. Please take a moment to fill out and hand in the survey.

Satellite Symposium, Society for Neuroscience

USING NEURON TO MODEL CELLS AND NETWORKS

Washington, DC

Friday, November 10, 2017

Ted Carnevale
Robert McDougal

Supported by NINDS

NEURON http://neuron.yale.edu/

NEURON: a brief tour

A tool for empirically-based models of neurons and neural circuits

Open source project directed by Michael Hines

Active development and user support

- Documentation, tutorials, and forum at https://www.neuron.yale.edu/
- Courses
- SFN meetings
- Summer course at UCSD and elsewhere
- Other courses

The NEURON user community

Used by experimentalists, theoreticians, and educators for neuroscience research and teaching

As of October 2017

- more than 1930 publications
- more than 1800 subscribers to mailing list and forum http://www.neuron.yale.edu/phpBB/
- source code for almost 600 published models at ModelDB https://modeldb.yale.edu/

Specifying and using models with NEURON

Model specifications written in hoc and/or Python
- and/or

created with GUI tools (work via hoc)
- CellBuilder, Channel Builder,
- Network Builder, Linear Circuit Builder

Add new functionality with NMODL (compiled)
- ion channels, synaptic mechanisms
- signal sources
- accumulation, diffusion, transport, reactions
  - described by ODEs, kinetic schemes,
  - algebraic equations
- events, state machines, artificial spiking cells

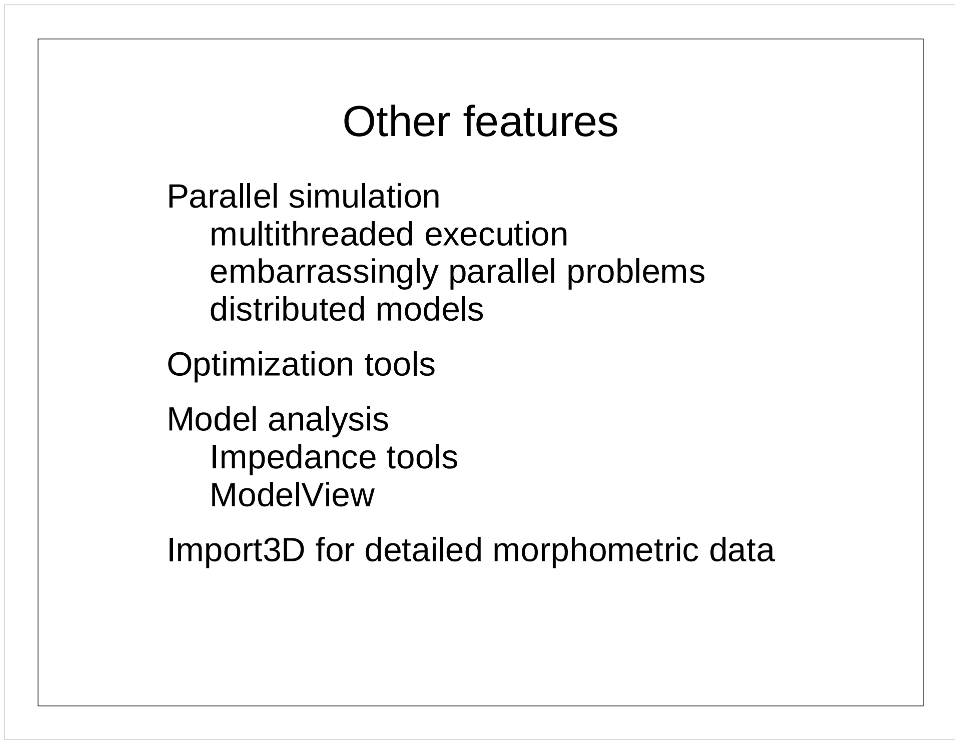
Add reactive diffusion (uses Python)

Not model specification, but necessary

Instrumentation
- stimulators, current or voltage clamps
- plotting and recording variables

Simulation control
- default and custom initializations
- integration methods
  - fixed time step
  - adaptive integration
- event system useful for implementing
  - "experimental protocols"

User interface



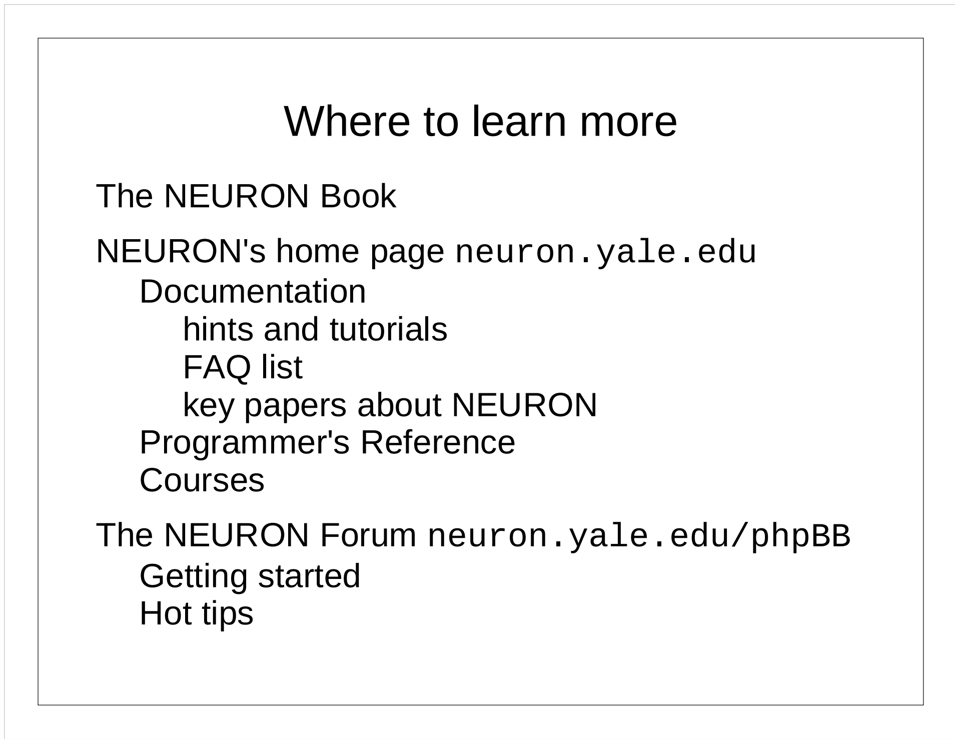
Other features

Parallel simulation
- multithreaded execution
- embarrassingly parallel problems
- distributed models

Optimization tools

Model analysis
- Impedance tools
- ModelView

Import3D for detailed morphometric data



Where to learn more

The NEURON Book

NEURON's home page `neuron.yale.edu`
- Documentation
  - hints and tutorials
  - FAQ list
  - key papers about NEURON
- Programmer's Reference
- Courses

The NEURON Forum `neuron.yale.edu/phpBB`
- Getting started
- Hot tips

The What and the Why of Neural Modeling

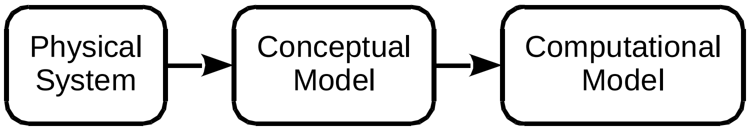
The moment-to-moment processing of information in the nervous system involves the propagation and interaction of electrical and chemical signals that are distributed in space and time.

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

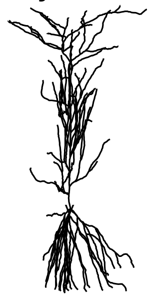
a simplified representation of the physical system

Computational model

an accurate representation of the conceptual model

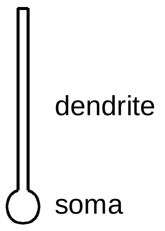
From Physical System to Computational Model

Physical system



Ca1 pyramidal cell

Conceptual model



ball and stick

Computational model

```
create soma, dendrite
connect dendrite(0), soma(1)
```

hoc code

Hierarchies of Complexity

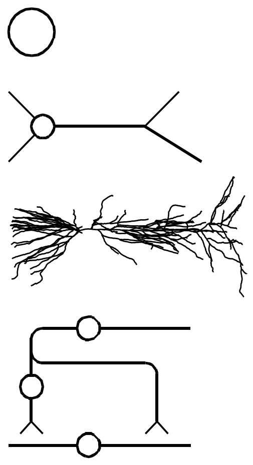
Structure

Single compartment

Stylized

Anatomically detailed

Network



Hierarchies of Complexity

Mechanism

Passive and Active currents
- HH-style
- kinetic scheme

Synaptic transmission
- continuous
- spike-triggered

Gap junctions

Extracellular fields, Linear circuits

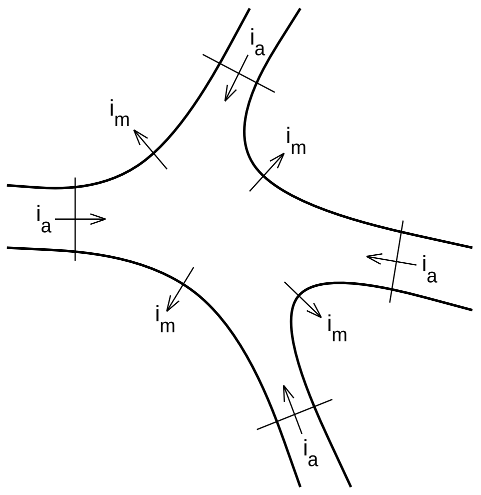
Diffusion, buffers, transport & exchange

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



$$C_m \frac{dV_m}{dt} + i_{ion} = \sum i_a$$

The Model Equations

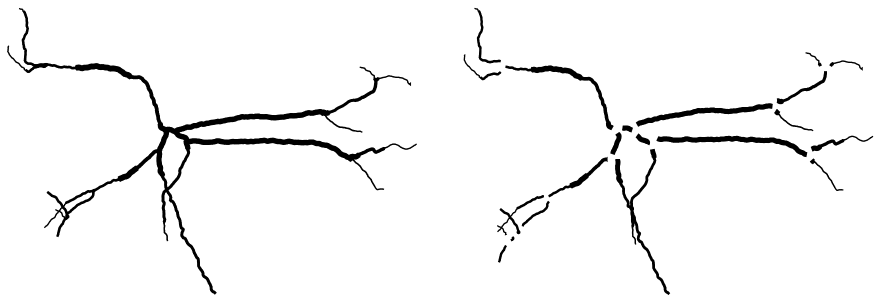
$$c_j \frac{dv_j}{dt} + i_{ion_j} = \sum_k \frac{v_k - v_j}{r_{jk}}$$

v_j	membrane potential in compartment j
i_{ion_j}	net transmembrane ionic current in compartment j
c_j	membrane capacitance of compartment j
r_{jk}	axial resistance between the centers of compartment j and adjacent compartment k

Separating Anatomy and Biophysics from Purely Numerical Issues

section

a continuous length of unbranched cable



Anatomical data from A.I. Gulyás

Mathematical description of a section

What we want:

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

What a new section gives us:

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

i.e. membrane capacitance and axial resistance,
but no ionic current.

How can we put ion channels in the membrane?

Adding mechanisms to sections

Density mechanisms
- distributed channels
- ion accumulation

Point processes
- electrodes, synapses

Described by
- differential equations
- kinetic schemes
- algebraic equations

Constructed with
- NMODL
- Channel Builder

```
hoc                                 Python

                                    from neuron import h

create soma, dend                   soma = h.Section()
                                    dend = h.Section()

connect dend(0), soma(1)            dend.connect(soma(1))

soma {
  L = 50 // [um] length             soma.L = 50 # [um] length
  diam = 50 // [um] diameter        soma.diam = 50
  nseg = 1                          soma.nseg = 1
  insert hh // HH mechanism         soma.insert('hh')
}

dend {
  L = 200                           dend.L = 200
  diam = 2                          dend.diam = 2
  nseg = 3                          dend.nseg = 3
  insert pas // passive channels    dend.insert('pas')
  e_pas = -65                       dend.e_pas = -65
}
```

Range Variables

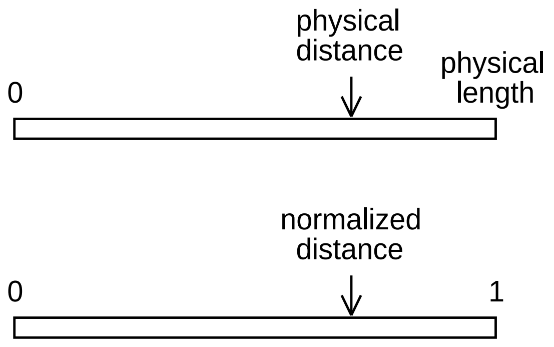
Name	Meaning	Units
`diam`	diameter	[µm]
`cm`	specific membrane capacitance	[µf/cm^2]
`g_pas`	specific conductance of the `pas` mechanism	[siemens/cm^2]
`v`	membrane potential	[mV]

range

normalized position along the length of a section

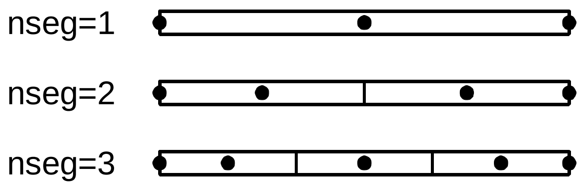
$0 \le \text{range} \le 1$

any variable name can be used for range, e.g. x



nseg

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



Example: `axon.nseg = 3`

To test spatial resolution

```
  for sec in h.allsec():
    sec.nseg = sec.nseg*3
```

and repeat the simulation

hoc: `forall nseg = nseg*3`

Syntax:

```
# access value of rangevar
# at location that corresponds to range
sectionname(range).rangevar
```

Examples:

```
dend(0.5).v # v at middle of dend
            # hoc: dend.v(0.5)

# print physical distance and v
# at every segment center in dend
for seg in dend:
  print seg.x, seg.x*dend.L, dend(seg.x).v
```

Category	**Variable**	**Units**
Time	`t`	[ms]
Distance	`diam, L`	[µm]
Voltage	`v`	[mV]
Current		
specific	`i`	[mA/cm^2] (density)
absolute		[nA] (point process)
Capacitance		
specific	`cm`	[µf/cm^2]
absolute		[nf] (point process)
Conductance		
specific	`g`	[S/cm^2] (density)
absolute		[µS] (point process)
Cytoplasmic resistivity	`Ra`	[Ω cm]
Resistance	`SEClamp.rs`	[10^6 Ω]
Concentration	`cai`, `nao`, etc.	[mM]

Why the GUI?

Improves productivity regardless of programming (in)experience by making it easier to

- develop, debug, and maintain models
- understand models developed by others
- visualize and understand simulation results
- use exploratory simulations to study model behavior
- optimize model parameters
- quickly create prototype models that can be mined for reusable code

Save time and avoid creating bugs--write less code!

Result: get more done faster and with less effort.

Using the GUI with Python

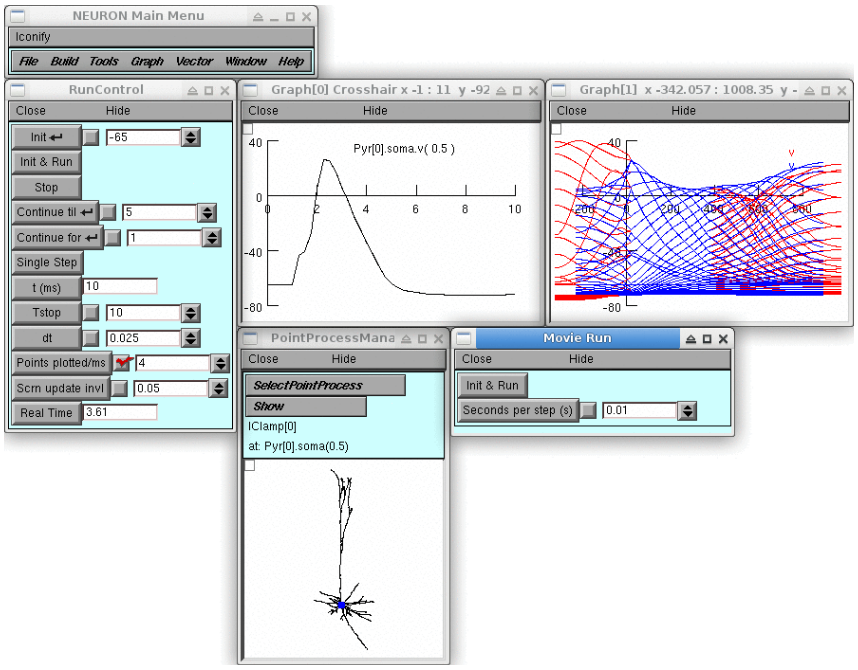
"I don't need to hear this. I won't use the GUI because I use Python, and the GUI doesn't work with Python."--Anonymous

Not so. The GUI works with Python, as long as the sections were created by hoc.

Example: pyrtest.py

```
from neuron import h,gui
h.load_file('Pyr.hoc') # defines Pyr class
# exported from CellBuilder
pyr = h.Pyr() # create instance of Pyr class
h.load_file('pyrtestrig.ses') # user interface
# built with NEURON's GUI tools
```

```
>>> python -i pyrtest.py
```



Init -65
Continue til 5
Continue for 1
t (ms) 10
Tstop 10
dt 0.025
Points plotted/ms 4
Scrn update invl 0.05
Real Time 3.61
IClamp[0]
at: Pyr[0].soma(0.5)
Seconds per step (s) 0.01

GUI tools

Many do things that would be very difficult, if not impossible, to accomplish with user-written code.

Import3d, Linear Circuit Builder, Multiple Run Fitter (optimizer), Impedance tools for analyzing electrical signaling in cells.

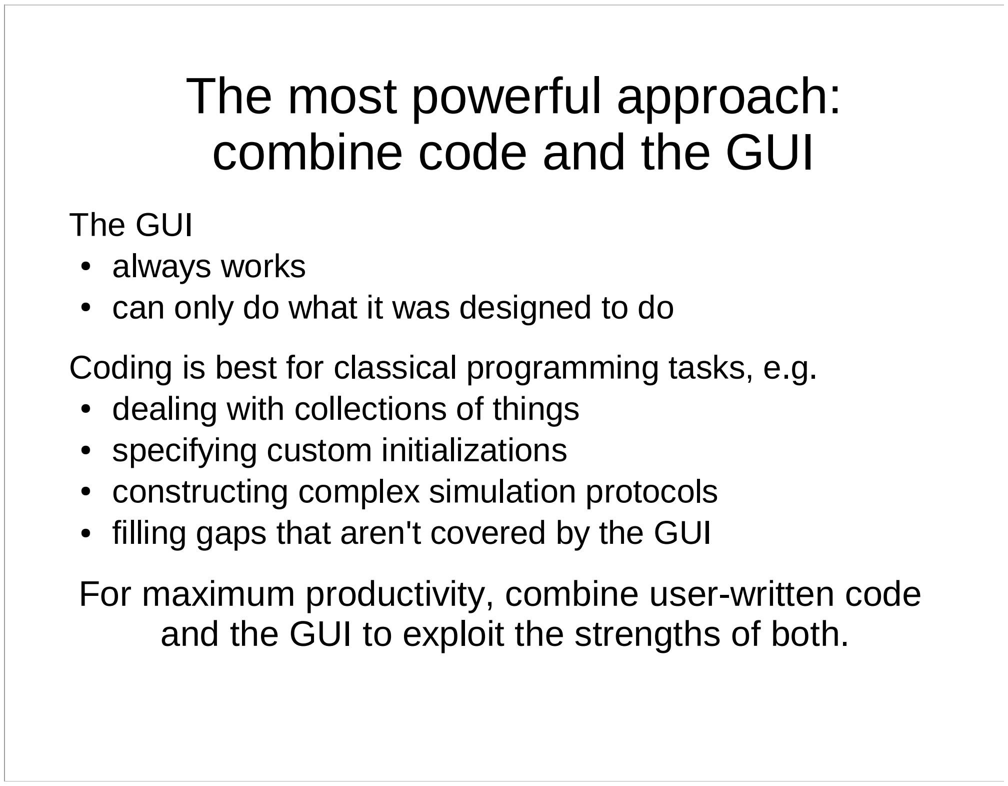
Some export code that can be reused with hoc and Python.

CellBuilder, Channel Builder, Linear Circuit Builder, Network Builder, Import3d, Model View (exports NeuroML)

Many can be saved directly to files for use by user-written hoc or Python script (example: pyrtest.py's custom interface)

Graphs, RunControl, any of the "Builders," Variable Step Control

See GUI tool tutorials on the Documentation page
https://neuron.yale.edu/neuron/docs



The most powerful approach: combine code and the GUI

The GUI
- always works
- can only do what it was designed to do

Coding is best for classical programming tasks, e.g.
- dealing with collections of things
- specifying custom initializations
- constructing complex simulation protocols
- filling gaps that aren't covered by the GUI

For maximum productivity, combine user-written code and the GUI to exploit the strengths of both.

Construction and Use of Models

Construction of cell models

- Specify topology: create and connect sections
- Specify geometry: stylized (L & diam)
  or 3D (x,y,z,diam)
- Specify biophysics: insert density mechanisms,
  attach "biological" point processes (synapses)

Construction of network models

- Define cell classes
- Create cells (instances of cell classes)
- Connect cells

Example: using the GUI to build and exercise a stylized model

1. How to use the CellBuilder to create and manage a model cell.
2. How to use NEURON's graphical tools to make an interface for running simulations.

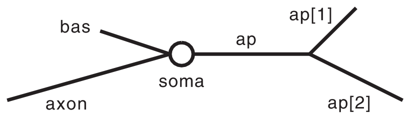
Step 0: Conceptualize the task

Shape
- stick figure / detailed

Channel distribution
- uniform / nonuniform
- whole cell / region / individual neurite

Creation
- single cell / use in a network

Step 1: using the CellBuilder to make a stylized model



Section	L	diam	Biophysics
soma	20 µm	20 µm	hh
ap[0]	400	2	reduced hh *
ap[1]	300	1	reduced hh *
ap[2]	500	1	reduced hh *
bas	200	3	pas §
axon	800	1	hh

* - gnabar_hh and gkbar_hh reduced to 10%, el_hh = - 64 mV

§ - e_pas = - 65 mV

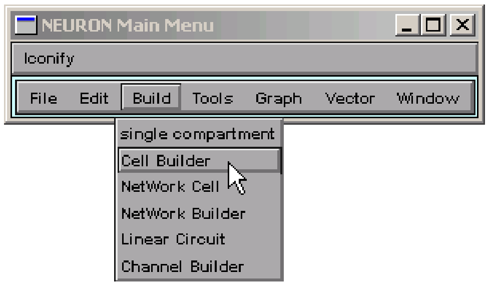
Throughout the cell Ra = 160 Ω cm, cm = 1 µf / cm^2

Launch NEURON with its library of graphical tools

UNIX/Linux `nrngui`

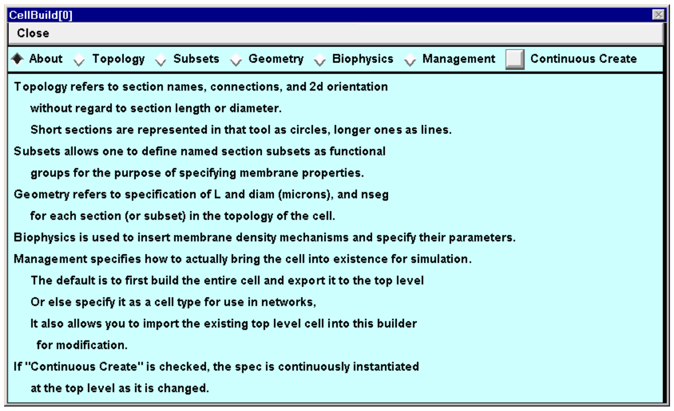
MSWin or OS X

Bring up a CellBuilder



NEURON Main Menu / Build / Cell Builder

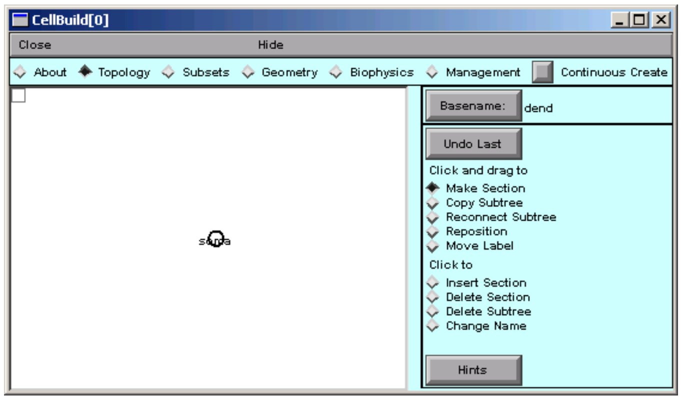
The CellBuilder



Topology refers to section names, connections, and 2d orientation
without regard to section length or diameter.
Short sections are represented in that tool as circles, longer ones as lines.
Subsets allows one to define named section subsets as functional
groups for the purpose of specifying membrane properties.
Geometry refers to specification of L and diam (microns), and nseg
for each section (or subset) in the topology of the cell.
Biophysics is used to insert membrane density mechanisms and specify their parameters.
Management specifies how to actually bring the cell into existence for simulation.
The default is to first build the entire cell and export it to the top level
Or else specify it as a cell type for use in networks,
It also allows you to import the existing top level cell into this builder
for modification.
If "Continuous Create" is checked, the spec is continuously instantiated
at the top level as it is changed.

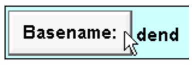
Use buttons from left to right.

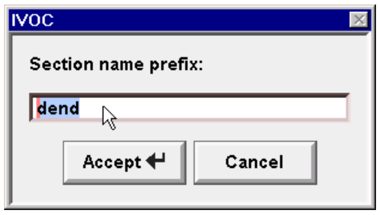
Topology

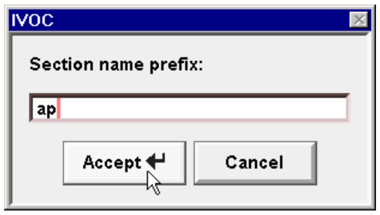


CB starts with a "soma" section.
We want to create new sections.

Specifying the "Basename"

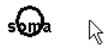




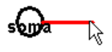


Making a new section

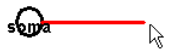
Place cursor near end of existing section



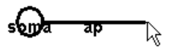
Click to start new section



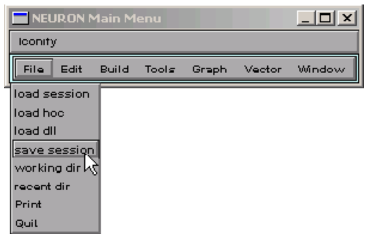
Drag to desired length



Release mouse button

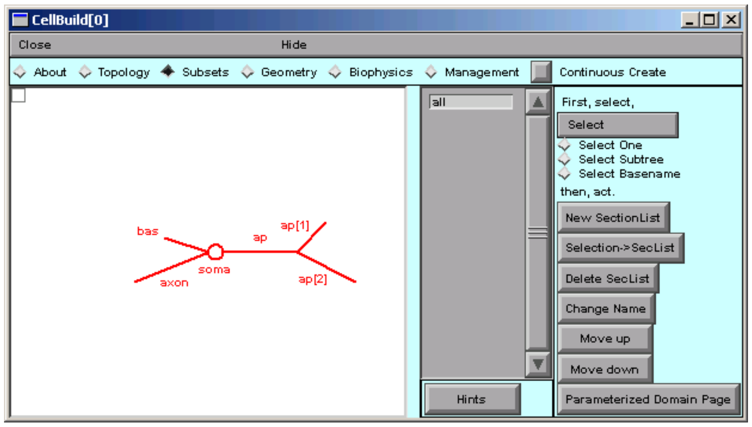


Save your work as you make progress!



NEURON Main Menu / File / save session

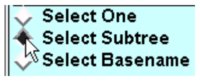
Subsets



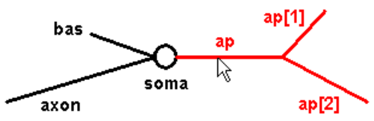
Group sections that have shared properties.
We want to make an "apicals" subset.

Making a new subset

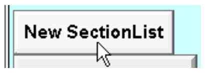
Click "Select Subtree"



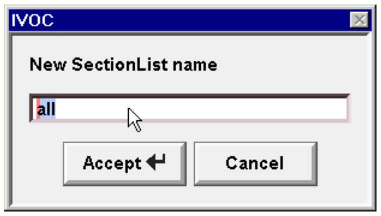
Click root of apical tree . . .



. . . then "New SectionList"

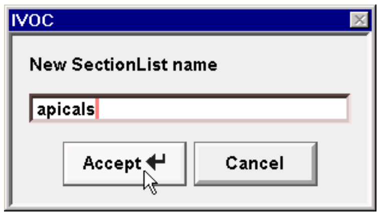


Making a new subset *continued*



New SectionList name

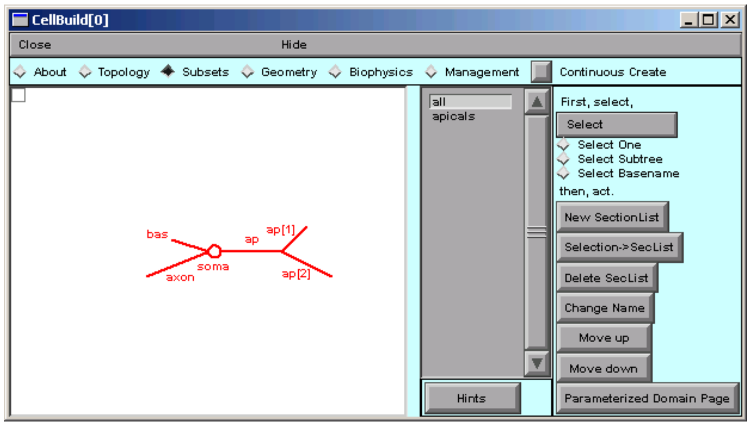
all



New SectionList name

apicals

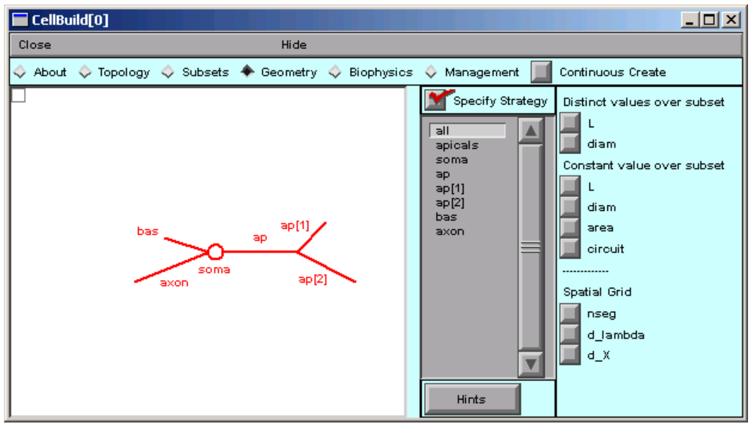
Subsets finished



Note "apicals".

Time to save a new session file.

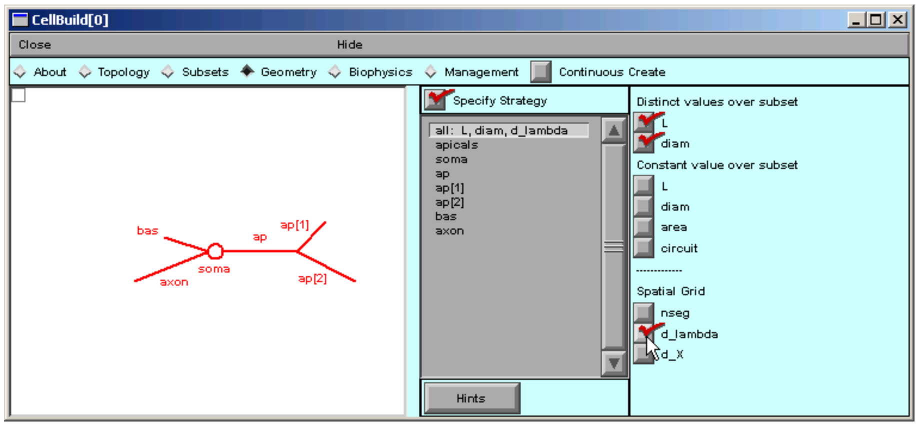
Geometry



"Specify Strategy" is ON.

A good strategy is a concise strategy.

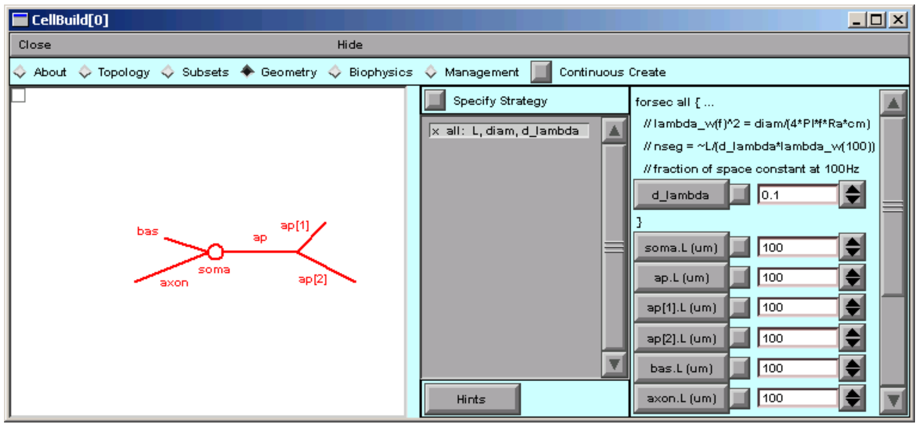
Geometry strategy



Each section has a different L and diam.

Compartmentalize according to $\lambda_{100\ \mathrm{Hz}}$ (d_lambda rule).

Implementing geometry strategy

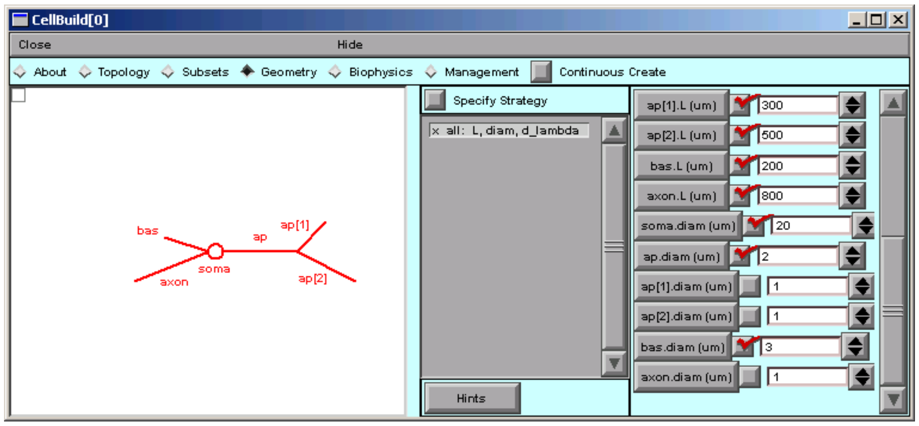


```
forsec all { ...
  // lambda_w(f)^2 = diam/(4*PI*f*Ra*cm)
  // nseg = ~L/(d_lambda*lambda_w(100))
  // fraction of space constant at 100Hz
  d_lambda        0.1
}
soma.L (um)      100
ap.L (um)        100
ap[1].L (um)     100
ap[2].L (um)     100
bas.L (um)       100
axon.L (um)      100
```

When strategy is complete, turn "Specify Strategy" OFF and start assigning values to parameters.

d_lambda = 0.1 at 100 Hz usually gives good spatial accuracy.

Implementing geometry *continued*



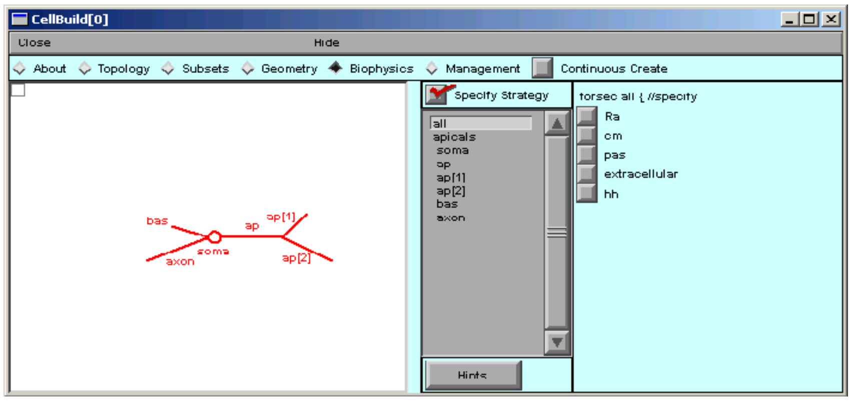
x all: L, diam, d_lambda

ap[1].L (um)	✓	300
ap[2].L (um)	✓	500
bas.L (um)	✓	200
axon.L (um)	✓	800
soma.diam (um)	✓	20
ap.diam (um)	✓	2
ap[1].diam (um)		1
ap[2].diam (um)		1
bas.diam (um)	✓	3
axon.diam (um)		1

Set L and diam for all sections.

Time to save to a session file!

Biophysics



✓ Specify Strategy

all
apicals
soma
ap
ap[1]
ap[2]
bas
axon

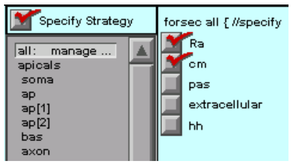
forsec all { //specify
Ra
cm
pas
extracellular
hh

"Specify Strategy" is ON.

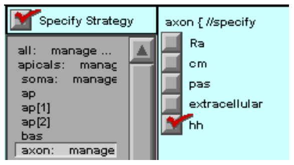
Base the plan on shared properties.

Biophysics strategy

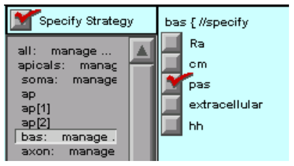
Ra and cm are homogeneous


```
[✓] Specify Strategy        forsec all { //specify
all:   manage ...           [✓] Ra
apicals                     [✓] cm
 soma                       [ ] pas
 ap                         [ ] extracellular
 ap[1]                      [ ] hh
 ap[2]
 bas
 axon
```

apicals, soma and axon have hh

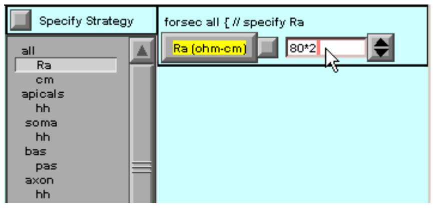

```
[✓] Specify Strategy        axon { //specify
all:   manage ...           [ ] Ra
apicals:   manag            [ ] cm
 soma:   manage             [ ] pas
 ap                         [ ] extracellular
 ap[1]                      [✓] hh
 ap[2]
 bas
 axon:   manage
```

bas has pas

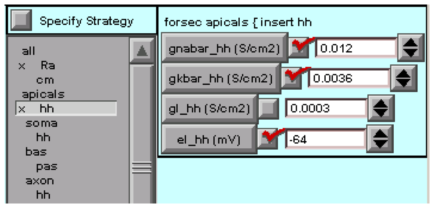

```
[✓] Specify Strategy        bas { //specify
all:   manage ...           [ ] Ra
apicals:   manag            [ ] cm
 soma:   manage             [✓] pas
 ap                         [ ] extracellular
 ap[1]                      [ ] hh
 ap[2]
 bas:   manage .
 axon:   manage
```

Implementing biophysics strategy

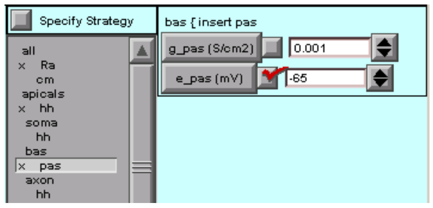
Double Ra


```
[ ] Specify Strategy        forsec all { // specify Ra
all                         Ra (ohm-cm) [ ] 80*2
   Ra
   cm
apicals
   hh
 soma
   hh
 bas
   pas
 axon
   hh
```

Fix apicals hh params


```
[ ] Specify Strategy        forsec apicals { insert hh
all                         gnabar_hh (S/cm2) [✓] 0.012
x  Ra                       gkbar_hh (S/cm2)  [✓] 0.0036
   cm                       gl_hh (S/cm2)     [ ] 0.0003
apicals                     el_hh (mV)        [✓] -64
x  hh
 soma
   hh
 bas
   pas
 axon
   hh
```

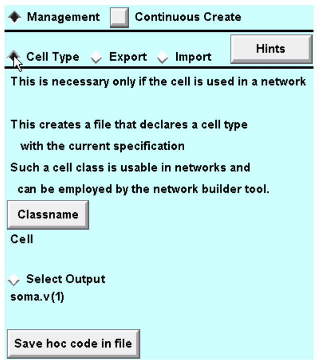
Shift e_pas in bas


```
[ ] Specify Strategy        bas { insert pas
all                         g_pas (S/cm2) [ ] 0.001
x  Ra                       e_pas (mV)    [✓] -65
   cm
apicals
x  hh
 soma
   hh
 bas
x  pas
 axon
   hh
```

Save another session file!!

Management

Option 1: save as a Cell Type
for use in a network



This is necessary only if the cell is used in a network

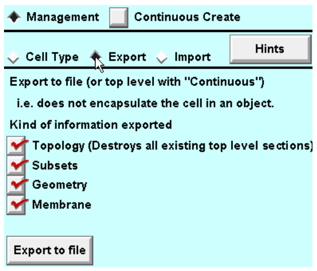
This creates a file that declares a cell type
with the current specification
Such a cell class is usable in networks and
can be employed by the network builder tool.

Cell

soma.v(1)

Management *continued*

Option 2: save as hoc file



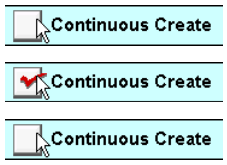
Export to file (or top level with "Continuous")
i.e. does not encapsulate the cell in an object.
Kind of information exported

- [✓] Topology (Destroys all existing top level sections)
- [✓] Subsets
- [✓] Geometry
- [✓] Membrane

Management *continued*

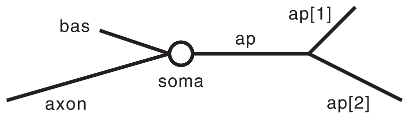
Option 3: export to interpreter

Toggle Continuous Create ON and OFF



or just leave it ON all the time.

Step 2: creating and using an interface for running simulations



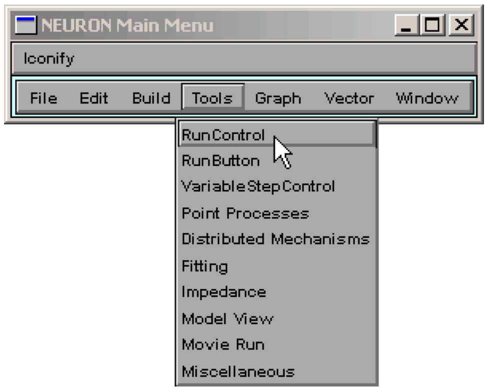
We want to

- attach a stimulating electrode
- evoke an action potential
- show time course of Vm at soma
- show Vm along a path from one end of the cell to the other

We need

- a "Run" button
- graphs to plot results
- a stimulator

Get a "Run" button



NEURON Main Menu / Tools / RunControl

RunControl panel

Init sets time to 0,
Vm to displayed value, and
conductances to steady-state

Init & Run does an Init,
then starts a simulation

Stop interrupts the simulation

Continue til runs until displayed time

Continue for runs for displayed
interval

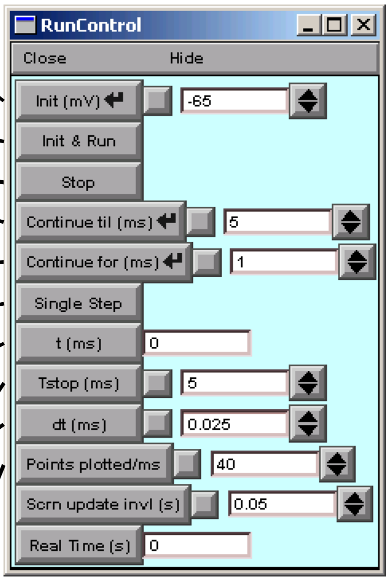
Single step advances by
1/(**Points plotted/ms**)

t numeric field shows model time

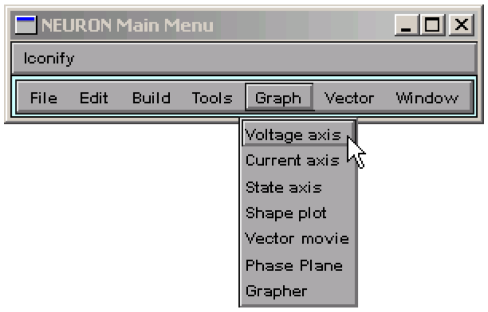
Tstop specifies when simulation ends

dt is integration time step;
must be integer fraction of
1/(**Points plotted/ms**)

Points plotted/ms is plotting interval

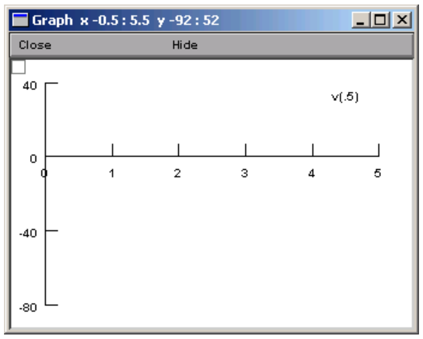

RunControl
Close Hide
Init (mV) -65
Init & Run
Stop
Continue til (ms) 5
Continue for (ms) 1
Single Step
t (ms) 0
Tstop (ms) 5
dt (ms) 0.025
Points plotted/ms 40
Scrn update invl (s) 0.05
Real Time (s) 0

We need to plot Vm(t) at soma



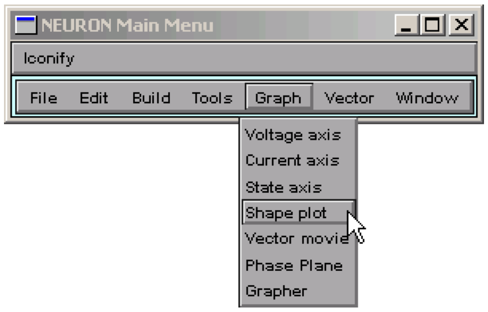
NEURON Main Menu / Graph / Voltage axis

Graph window



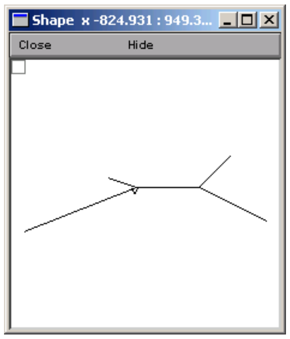
v(.5) is Vm at middle of default section
(soma in this example)

We need to plot Vm along a path



NEURON Main Menu / Graph / Shape plot

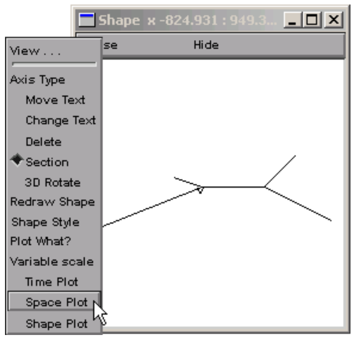
Bringing up a space plot



Use this "shape plot" to create a "space plot".

Click on its "menu box" . . .

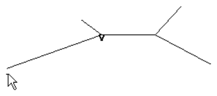
Bringing up a space plot *continued*



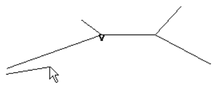
. . . and scroll down to "Space Plot".

Bringing up a space plot *continued*

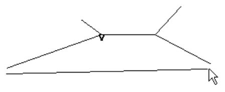
Click just left of the shape



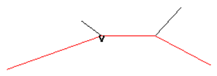
Hold button down while dragging from left . . .



. . . to right . . .

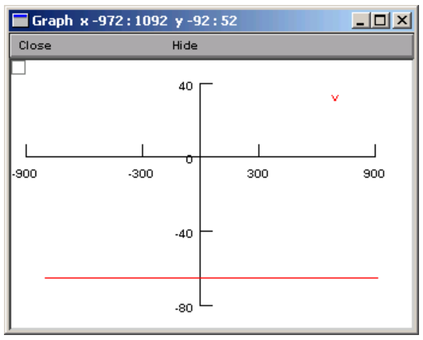


. . . then release button.



This pops up a . . .

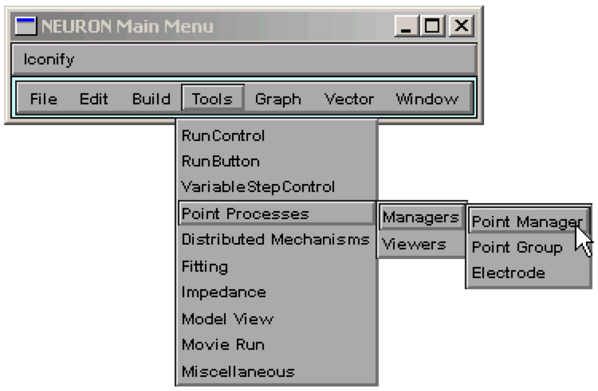
Space plot



A plot of Vm vs. distance along a path.

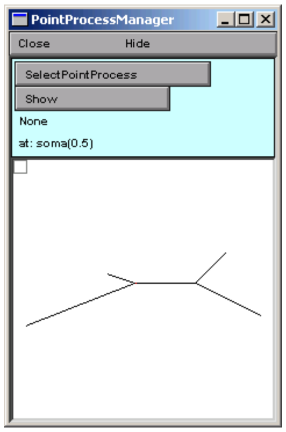
Better save a session file.

We need a stimulator



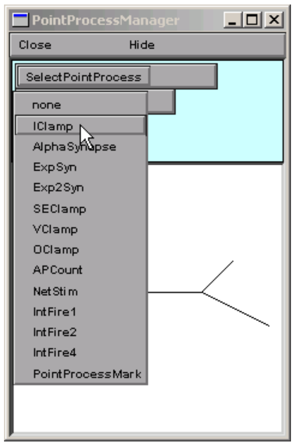
NEURON Main Menu / Tools / Point Processes / Managers / Point Manager

PointProcessManager window



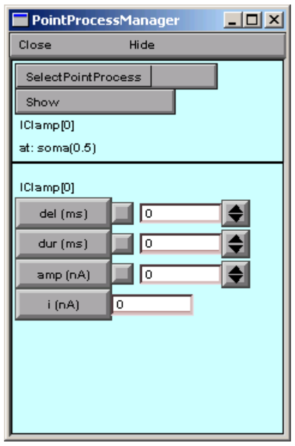
To make this an IClamp . . .

Creating an IClamp



. . . click on SelectPointProcess and scroll down to IClamp.

IClamp parameter panel



IClamp[0]
at: soma(0.5)

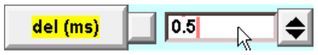
IClamp[0]

del (ms)	0
dur (ms)	0
amp (nA)	0
i (nA)	0

Next: set parameter values.

Entering values into numeric fields

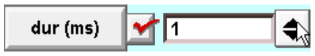
Direct entry



del (ms) 0.5

Note yellow highlight on button

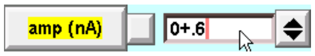
Spinner



dur (ms) 1

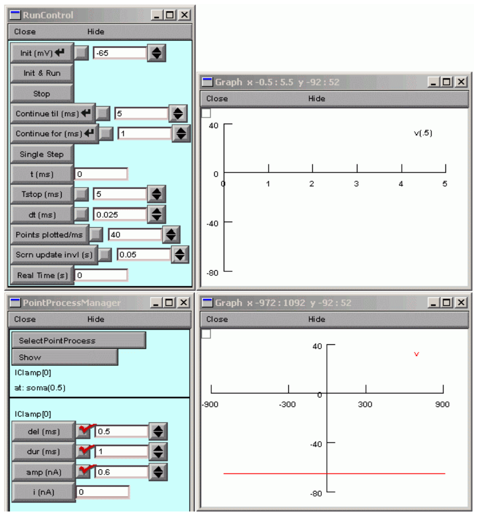
Red check means value has been changed from default

Mathematical expression



amp (nA) 0+.6

Our user interface



RunControl

Init (mV) -65

t (ms) 0

Continue til (ms) 5

Continue for (ms) 1

Tstop (ms) 5

dt (ms) 0.025

Points plotted/ms 40

Scrn update invl (s) 0.05

Real Time (s) 0

Graph x -0.5 : 5.5 y -92 : 52

PointProcessManager

IClamp[0]

at: soma(0.5)

IClamp[0]

del (ms) 0.5

dur (ms) 1

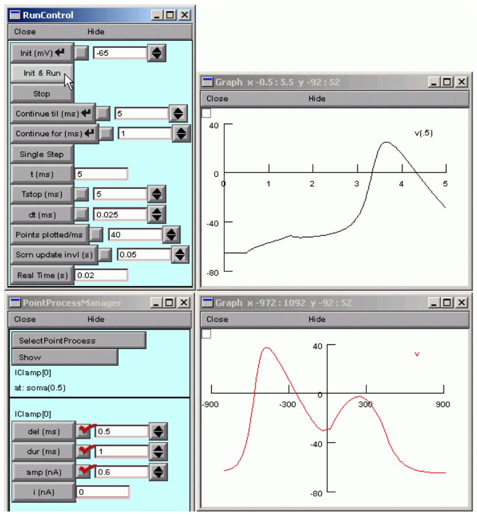
amp (nA) 0.6

i (nA) 0

Graph x -972 : 1092 y -92 : 52

Time to save to a new session file!

It works!

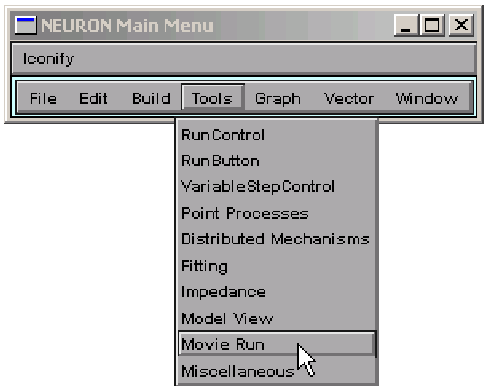


Init (mV) -65
Continue til (ms) 5
Continue for (ms) 1
t (ms) 5
Tstop (ms) 5
dt (ms) 0.025
Points plotted/ms 40
Scrn update invl (s) 0.05
Real Time (s) 0.02

IClamp[0]
at: soma(0.5)

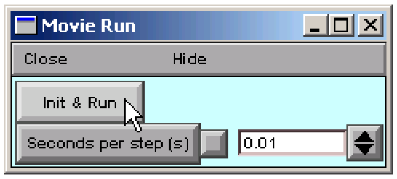
IClamp[0]
del (ms) ✓ 0.5
dur (ms) ✓ 1
amp (nA) ✓ 0.6
i (nA) 0

How to get nice space plot "movies"



NEURON Main Menu / Tools / Movie Run

Space plot "movies" *continued*



Movie Run / Init & Run

What if channel density in the apical tree varies systematically with position, e.g. distance from the soma?

See "Specifying parameterized variation of biophysical properties" in the CellBuilder tutorial at https://neuron.yale.edu/neuron/docs

The Linear Circuit Builder

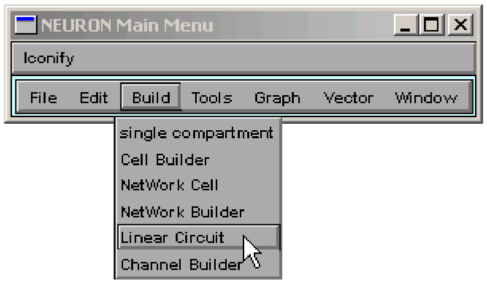
For building models that have linear circuit elements and may also involve neurons

Circuit elements include ground, current & voltage source, R, C, op amp

Potential applications include

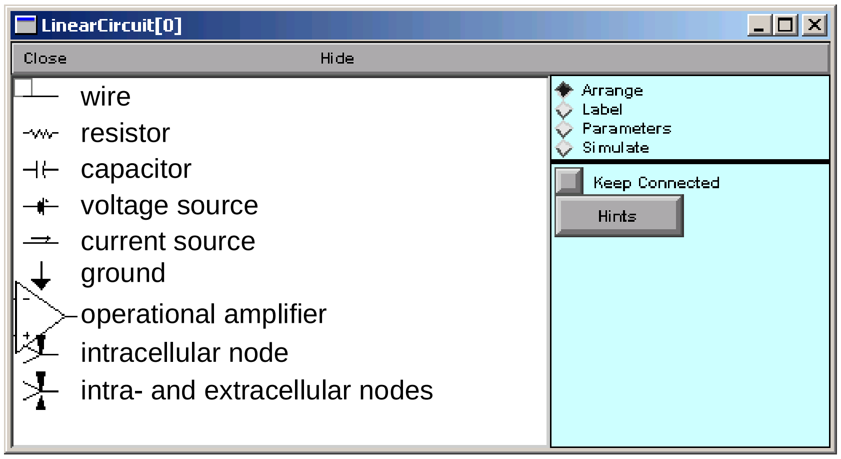
- effects and compensation of electrode R & C
- two-electrode voltage clamp
- ohmic and nonlinear gap junctions

1. Bring up a Linear Circuit Builder



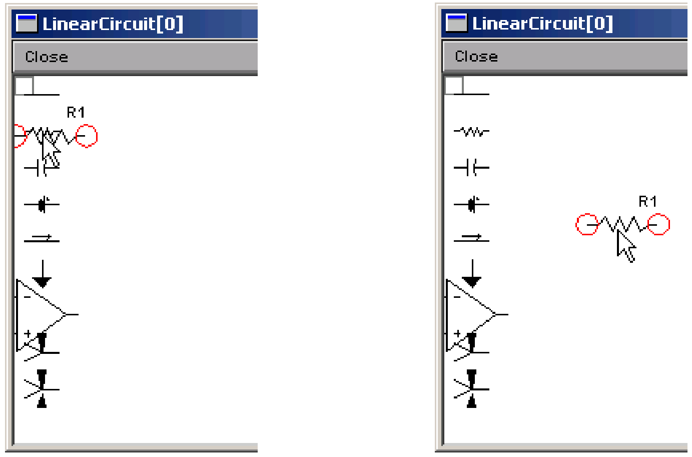
NEURON Main Menu / Build / Linear Circuit

The Linear Circuit Builder



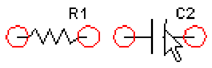
Arrange: spawn components

Click on palette and drag onto canvas

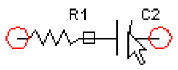


Arrange: connect components

Click and drag to overlap red circles



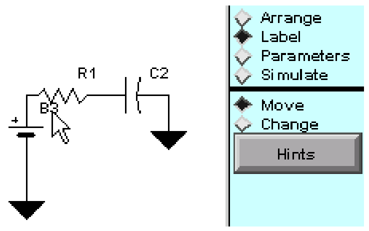
Black square is "solder joint"

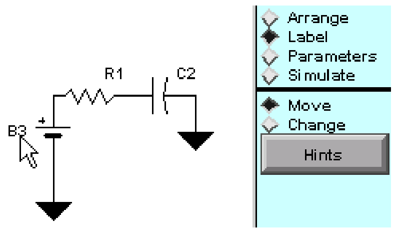


Pull apart to break connection

Label: move labels

Click and drag to new location

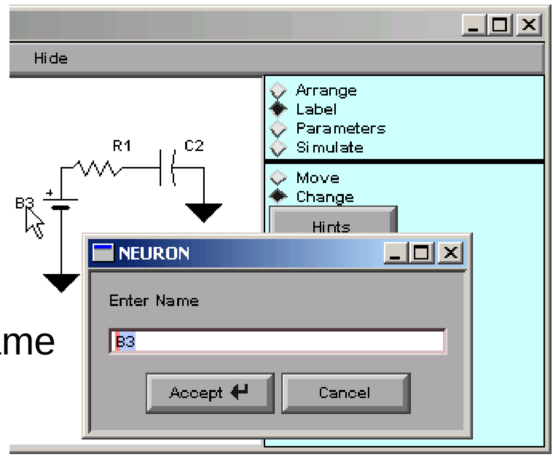




Label: change labels 1

Click on a label . . .

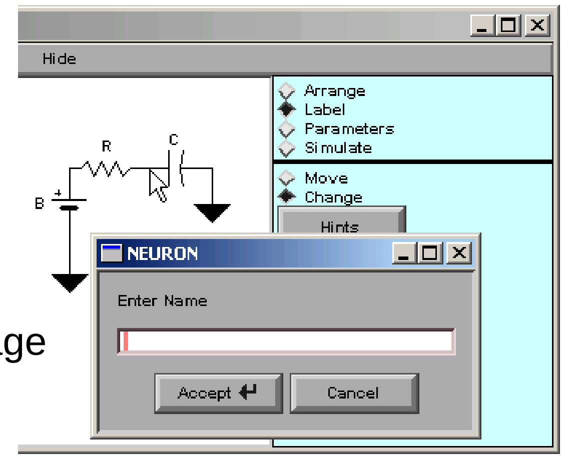
. . . to change its name



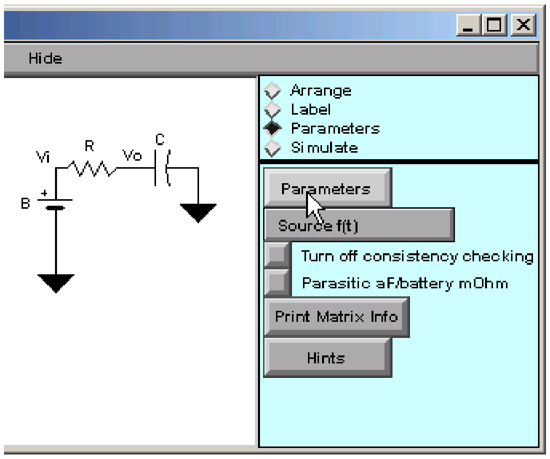
Label: change labels 2

Click on a node . . .

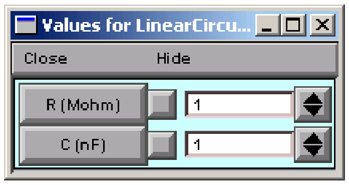
. . . to label a voltage



Parameters: non-source elements



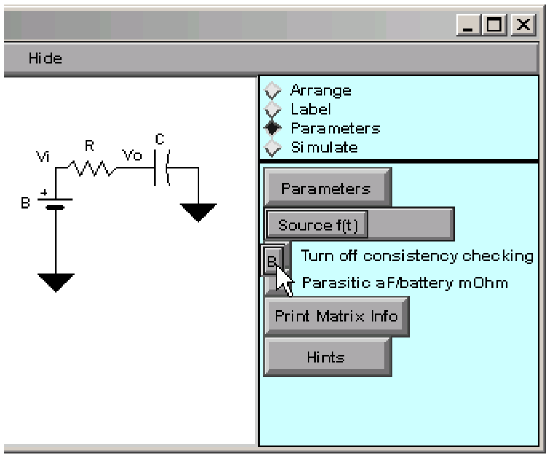
Click on "Parameters"



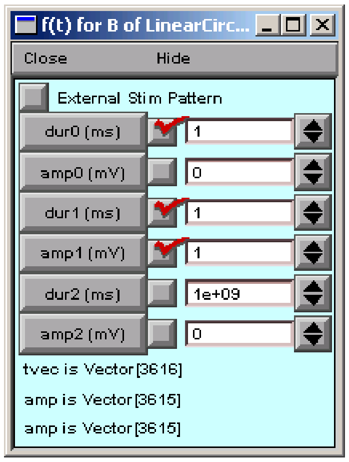
R (Mohm) 1
C (nF) 1

Parameters: signal sources

Source f(t) / B



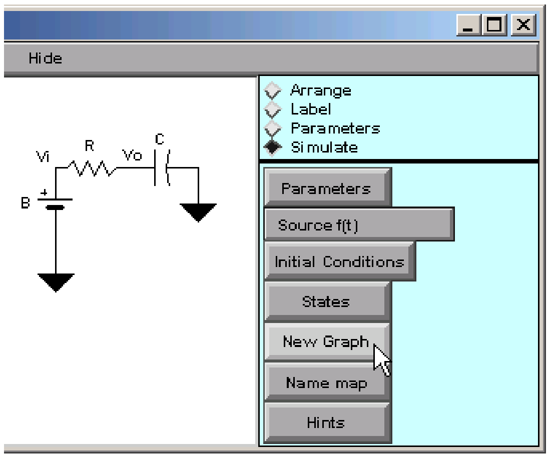
Parameters: signal sources *continued*



dur0 (ms) ✓ 1
amp0 (mV) 0
dur1 (ms) ✓ 1
amp1 (mV) ✓ 1
dur2 (ms) 1e+09
amp2 (mV) 0
tvec is Vector[3616]
amp is Vector[3615]
amp is Vector[3615]

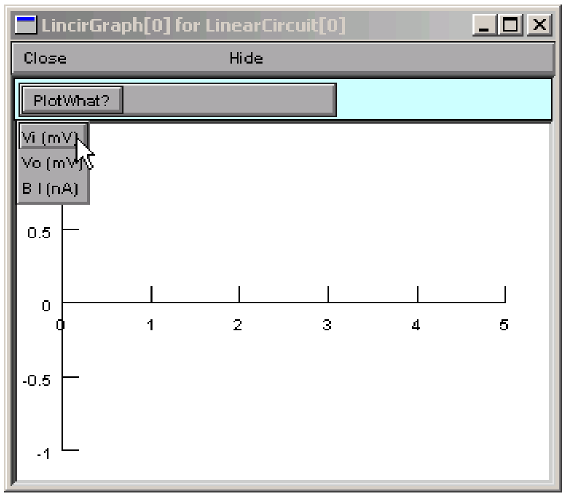
Configured

Simulate: creating a graph



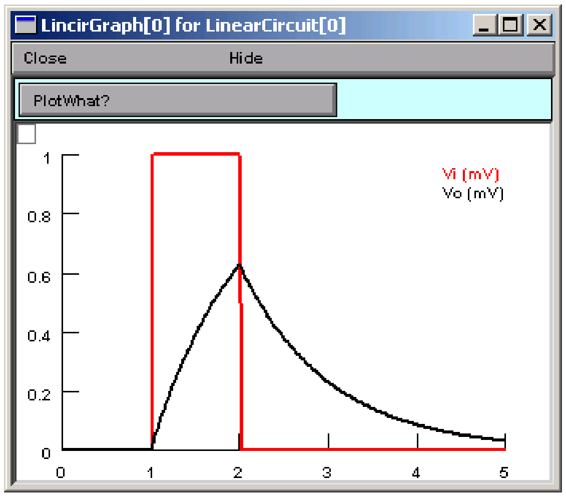
New Graph

Simulate: specifying what to plot



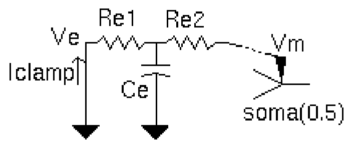
PlotWhat? / *variable_label*

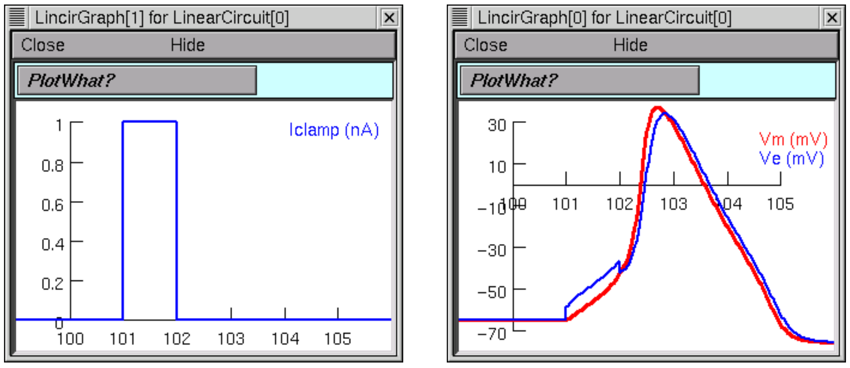
Simulate: simulation results



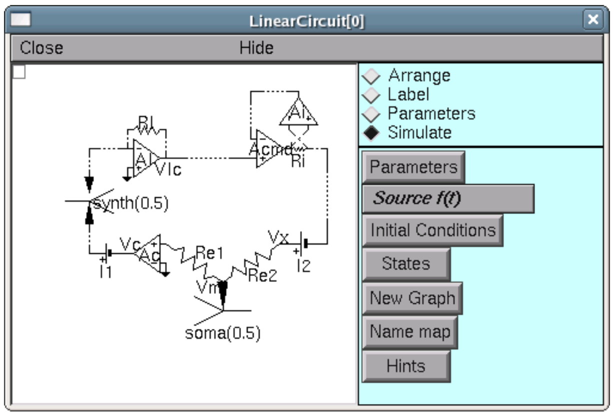
After minor cosmetic changes

Patch clamp with electrode R and C





NEURON demo: dynamic clamp



NMODL

NEURON Model Description Language

Add new membrane mechanisms to NEURON

Density mechanisms
- Distributed Channels
- Ion accumulation

Point Processes
- Electrodes
- Synapses

Described by
- Differential equations
- Kinetic schemes
- Algebraic equations

Benefits

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

NMODL general block structure

What the model looks like from outside

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

What names are manipulated by this model

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

Initial default values for states

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

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

(and specify how states are to be integrated)

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

State equations

```
DERIVATIVE deriv {
    rates(v)
    n′ = (ninf − n)/ntau
}
```

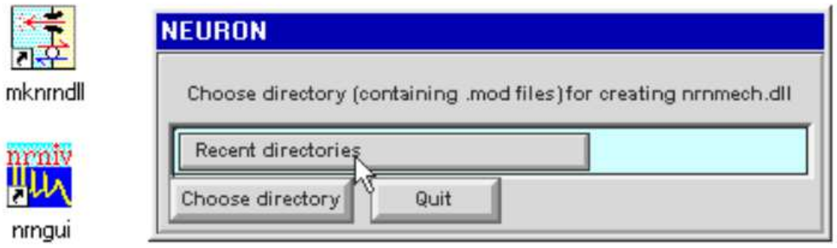
Functions and procedures

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

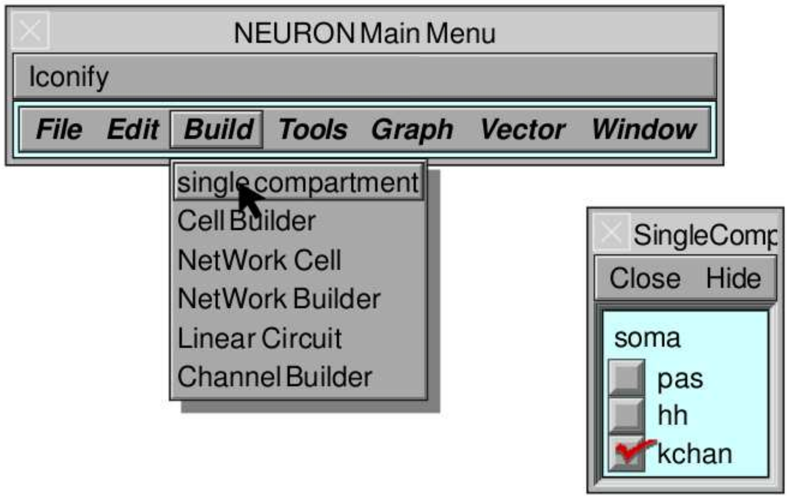
UNIX

```
nrnivmodl
nrngui
```

MSWIN


mknrndll
nrngui
NEURON
Choose directory (containing .mod files) for creating nrnmech.dll
Recent directories
Choose directory
Quit

Select NEURON Main Menu / Build / single compartment



Density mechanism

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

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

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

BREAKPOINT {
    i = g*(v − e)
}
```

Point Process

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

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

ASSIGNED {
    i (nanoamp)
    v (millivolt)
}

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

Density mechanism

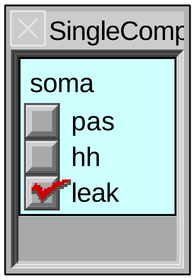
Point Process

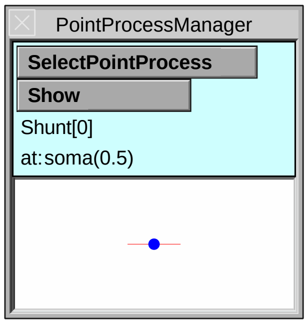
NMODL

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

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

GUI





Shunt[0]
at:soma(0.5)

Interpreter

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

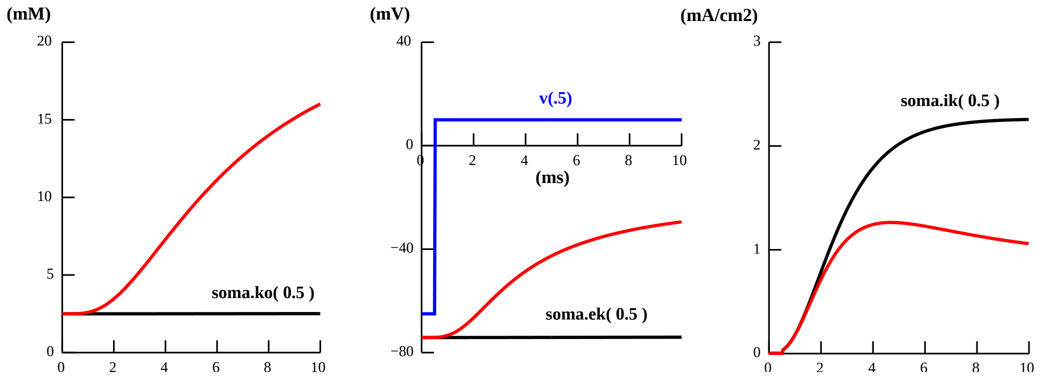
```
objref s
soma s = new Shunt(.5)
s.r = 2
```

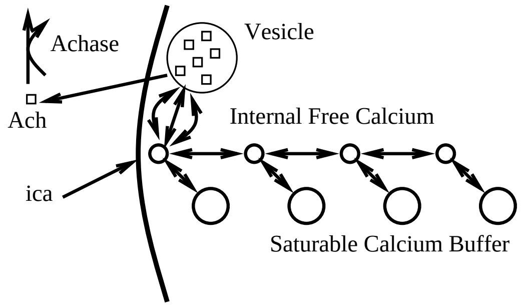
Ion Channel

```
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.

```
BREAKPOINT {
    i = (v − e)/r
}
```

The output from
modlunit shunt
is:

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

To fix the problem replace the line with:

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

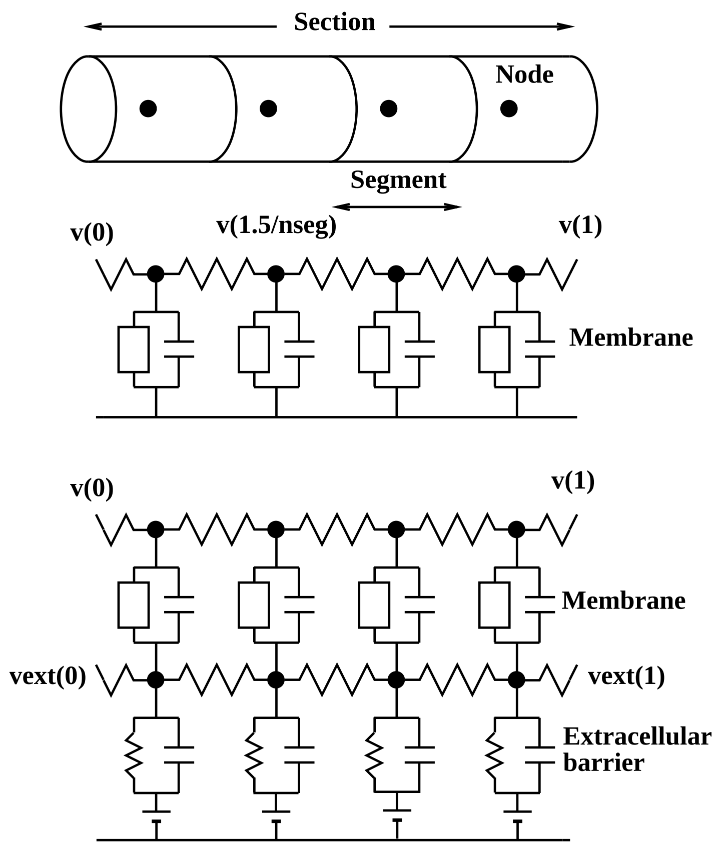
What conversion factor will make the following consistent?

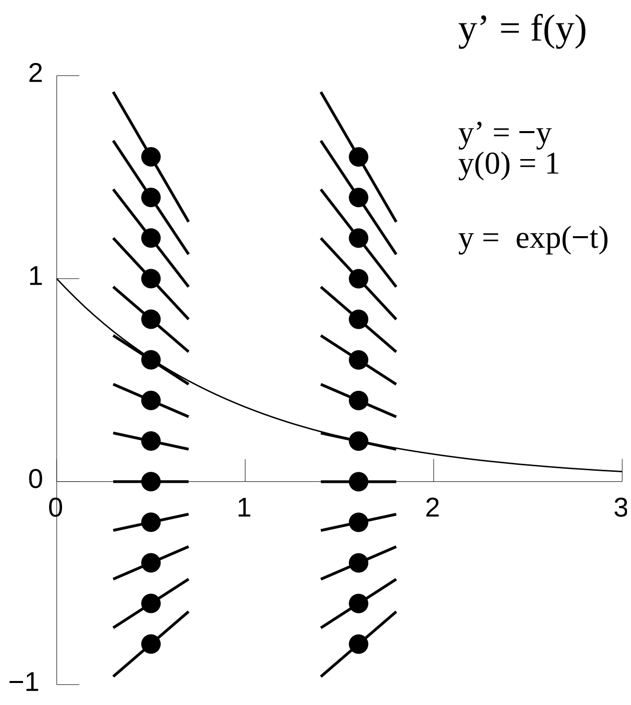
```
  nai’    =    ina      /     FARADAY     *     (c/radius)
(uM/ms)   (mA/cm2)  /  (coulomb/mole)           /  (um)
```

Compartmental Modeling

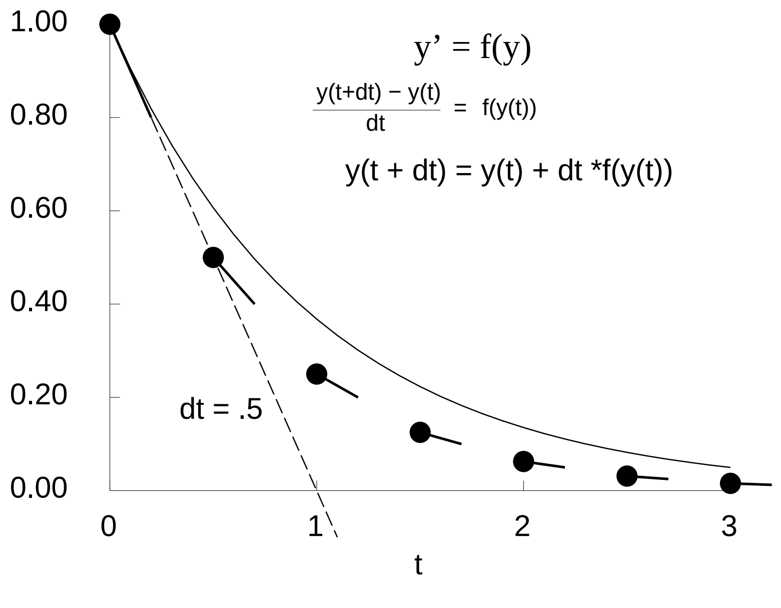
Not much mathematics required.

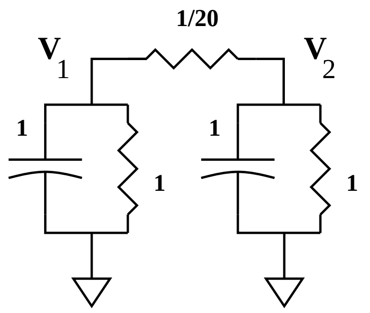
Good judgment essential!

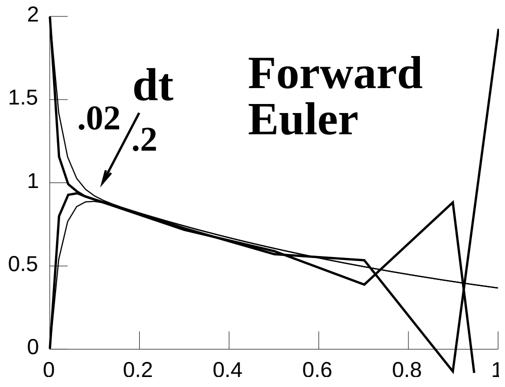


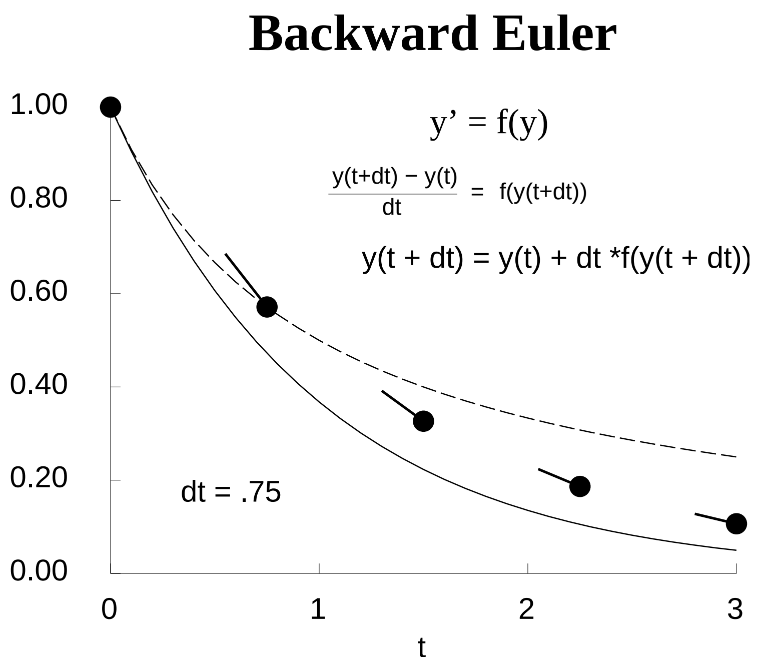


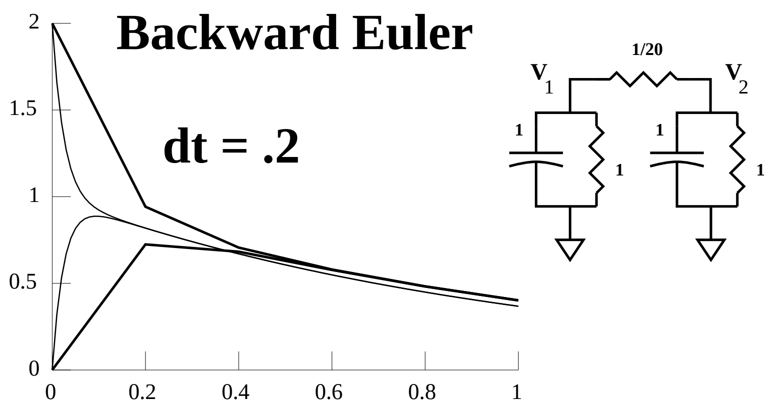
Forward Euler

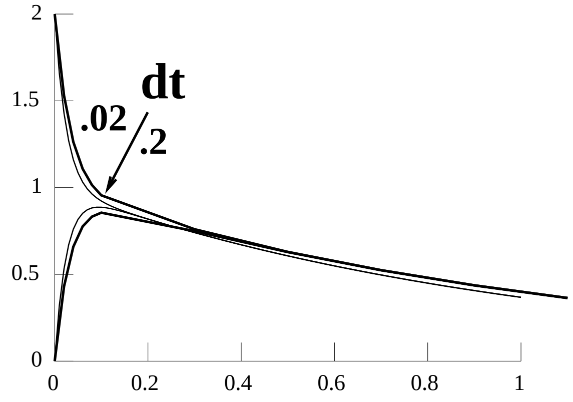


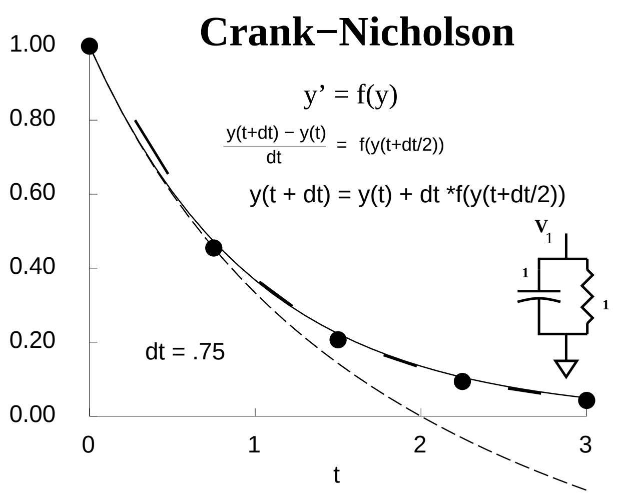

1/20
V_1
V_2
1
1
1
1

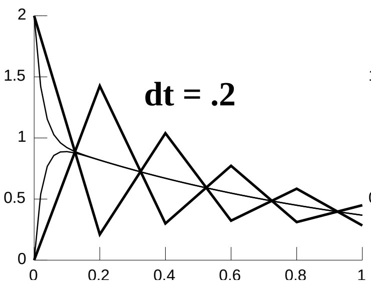

Forward Euler
dt
.02
.2
2
1.5
1
0.5
0
0
0.2
0.4
0.6
0.8
1

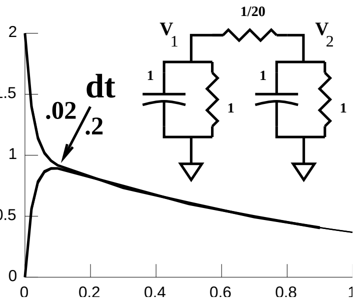

Backward Euler
y' = f(y)
$\frac{y(t+dt) - y(t)}{dt} = f(y(t+dt))$
y(t + dt) = y(t) + dt *f(y(t + dt))
dt = .75
1.00
0.80
0.60
0.40
0.20
0.00
0
1
2
3
t

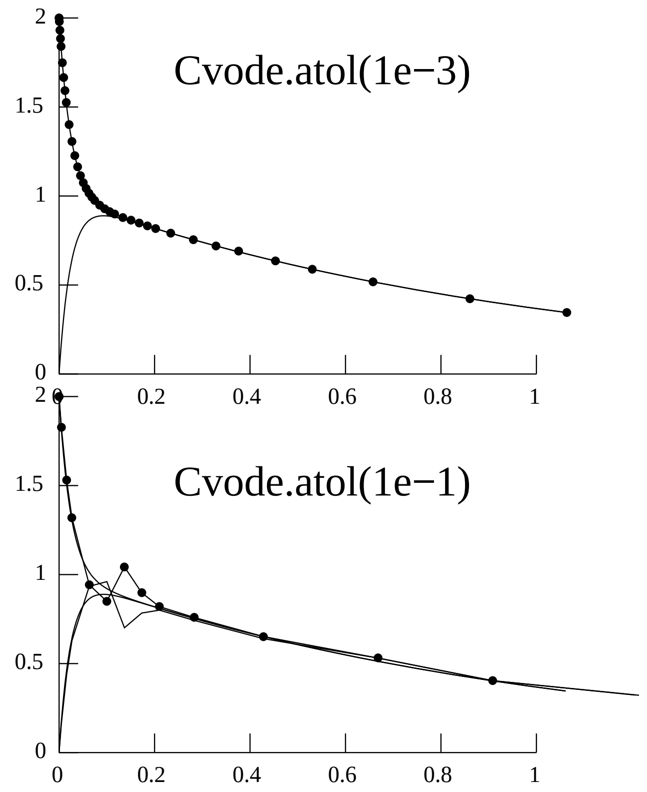

Backward Euler
dt = .2
1/20
V1
V2
1
1
1
1
0
0.5
1
1.5
2
0
0.2
0.4
0.6
0.8
1

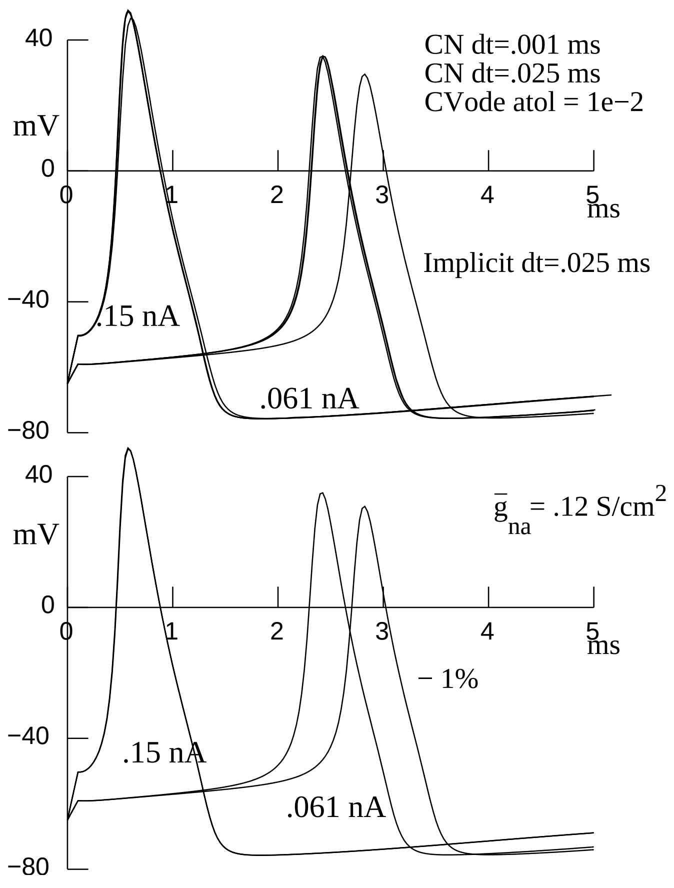

dt
.02
.2
0
0.5
1
1.5
2
0
0.2
0.4
0.6
0.8
1


Crank−Nicholson
y’ = f(y)
(y(t+dt) − y(t))/dt = f(y(t+dt/2))
y(t + dt) = y(t) + dt *f(y(t+dt/2))
V1
1
1
dt = .75
1.00
0.80
0.60
0.40
0.20
0.00
0
1
2
3
t


dt = .2
0
0.5
1
1.5
2
0
0.2
0.4
0.6
0.8
1


dt
.02
.2
1/20
V1
V2
1
1
1
1
0
0.5
1
1.5
2
0
0.2
0.4
0.6
0.8
1


Cvode.atol(1e−3)
Cvode.atol(1e−1)
2
1.5
1
0.5
0
0.2
0.4
0.6
0.8
1


CN dt=.001 ms
CN dt=.025 ms
CVode atol = 1e−2
Implicit dt=.025 ms
mV
ms
.15 nA
.061 nA
$\bar{g}_{na}$ = .12 S/cm^2
− 1%
40
0
−40
−80
1
2
3
4
5

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

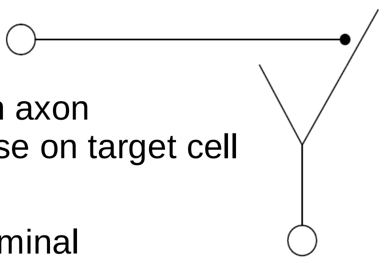
1. Define the types of cells
2. Create each cell in the network
3. Connect the cells

Communication between cells

Gap junctions

Synaptic transmission
- graded
- spike-triggered

Spike-triggered synaptic transmission

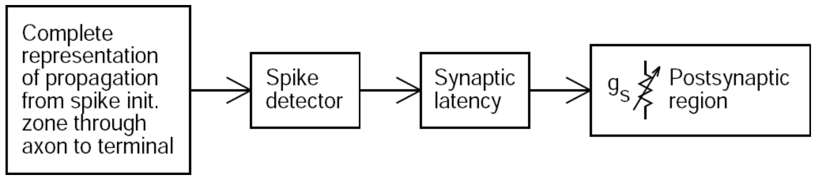


Physical system:
- Presynaptic neuron with axon that projects to synapse on target cell

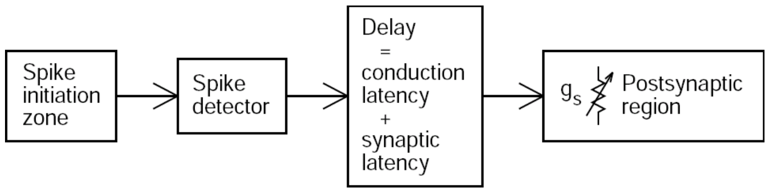
Conceptual model:
- Spike in presynaptic terminal triggers transmitter release; presynaptic details unimportant
- Postsynaptic effect described by DE or kinetic scheme that is perturbed by occurrence of a presynaptic spike

Spike-triggered transmission: computational implementation

Basic idea


Complete representation of propagation from spike init. zone through axon to terminal
Spike detector
Synaptic latency
g_s
Postsynaptic region

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)
```

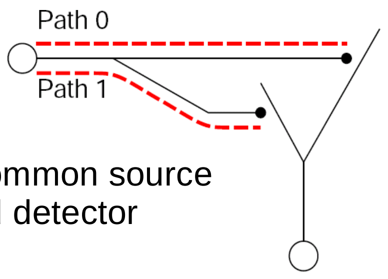
Defaults

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

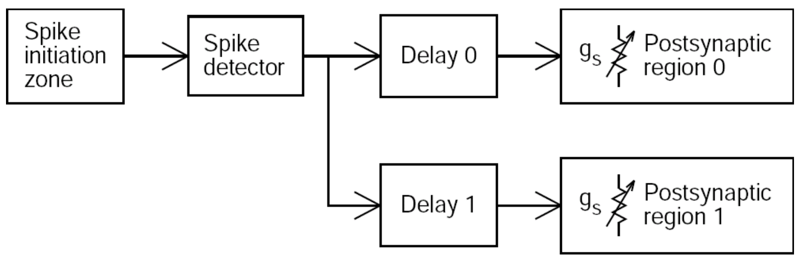
NMODL specification of synaptic mechanism

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

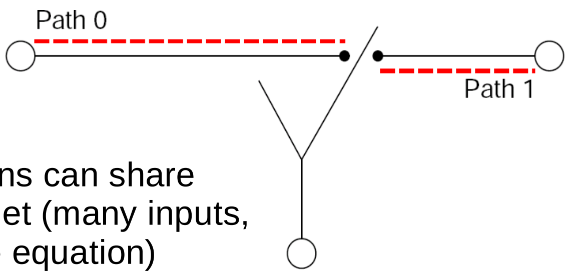
Efficient divergence



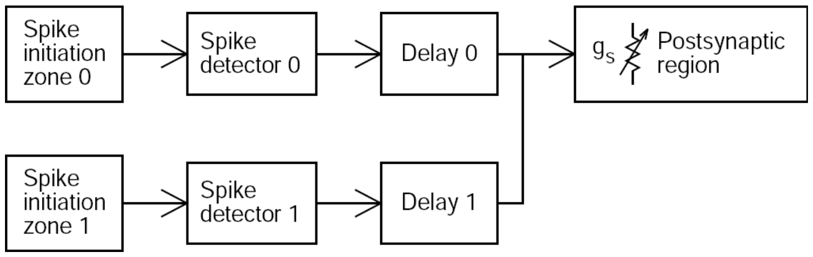
Multiple NetCons with a common source share a single threshold detector



Efficient convergence



Multiple NetCons can share
a single target (many inputs,
but only one equation)

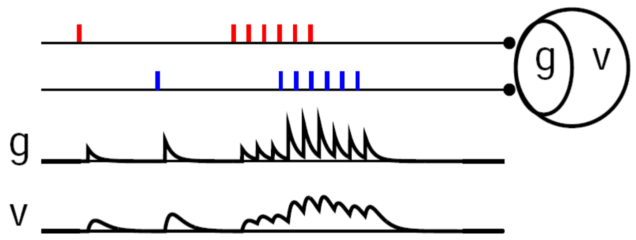


Example: g_s with fast rise and exponential decay

```
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 }
```

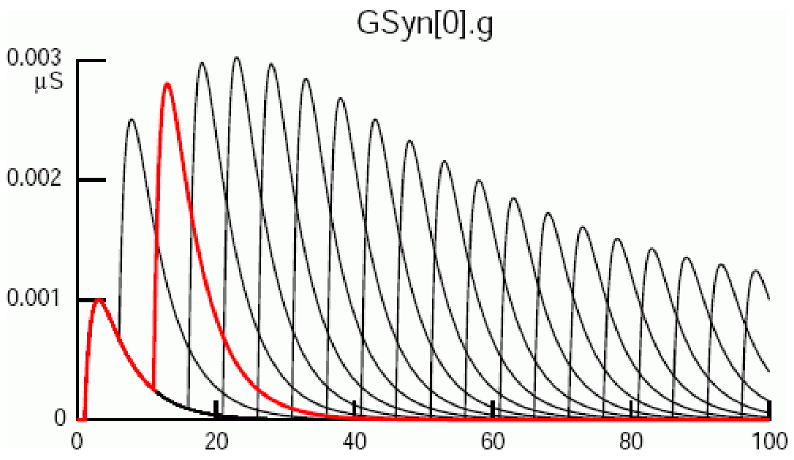
g_s with fast rise and exponential decay

continued



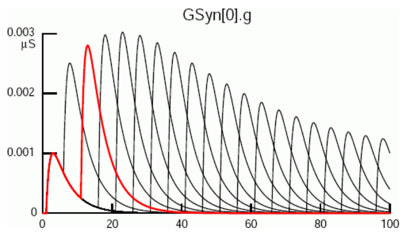
```
BREAKPOINT {
  SOLVE state METHOD cnexp
  i = g*(v-e)
}
DERIVATIVE state { g' = -g/tau }
NET_RECEIVE(w (uS)) { g = g + w }
```

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

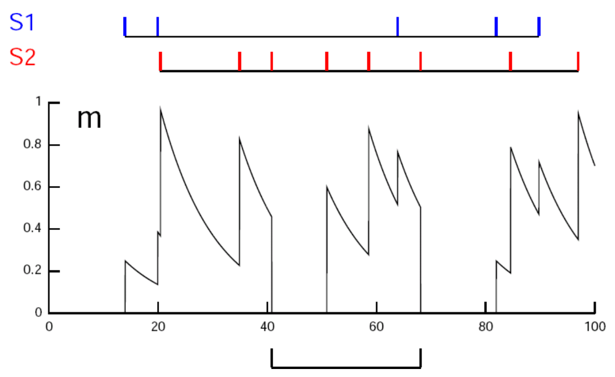
Prerequisite: all state variables must be analytically computable from a new initial condition

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

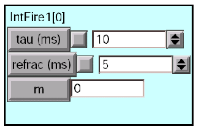
Example: leaky integrate and fire model



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
  }
}
```

IntFire1

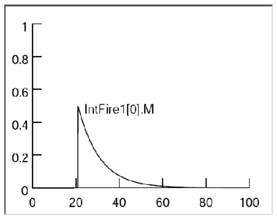


IntFire1[0]

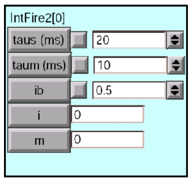
tau (ms) 10

refrac (ms) 5

m 0



IntFire2



IntFire2[0]

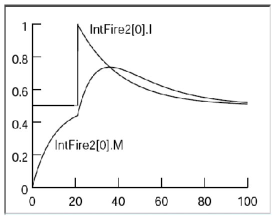
taus (ms) 20

taum (ms) 10

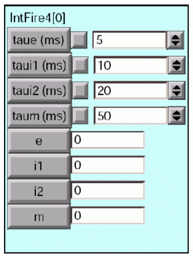
ib 0.5

i 0

m 0



IntFire4



IntFire4[0]

taue (ms) 5

taui1 (ms) 10

taui2 (ms) 20

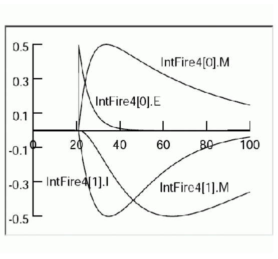
taum (ms) 50

e 0

i1 0

i2 0

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
}
```

Launch NEURON

via the GUI

- double click on nrngui icon
- double click on hoc file
- drag and drop hoc file onto nrngui icon

via the command line in a terminal
(*MS Win: double click on bash shell icon*)

- `nrngui # loads NEURON's GUI library`
- `nrniv  # omits GUI library`
- `nrngui bah.hoc # executes bah.hoc`
- `nrngui -python foo.py # executes foo.py`

Start Python, use NEURON as a module

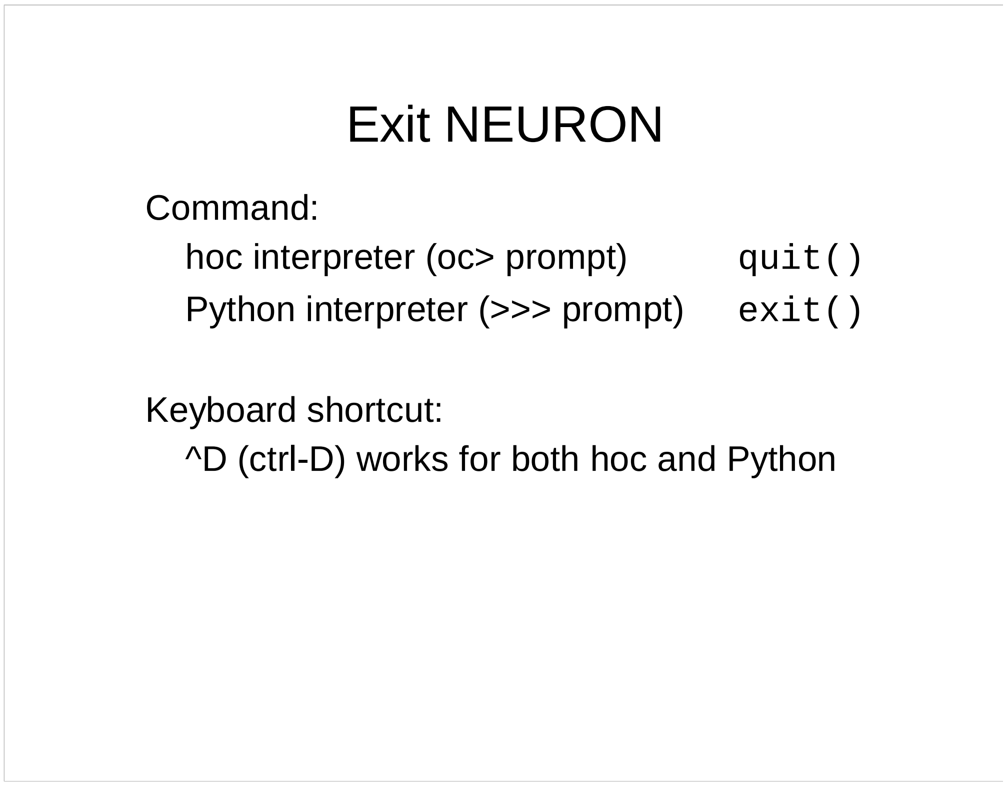
```
python foo.py
```

where foo.py contains

```
from neuron import h
# get NEURON's GUI with
# from neuron import h,gui
h.load_file('bah.hoc')
```

- may need PYTHONPATH
- prevent autoexit with -i switch, e.g.

```
python -i foo.py
```



Exit NEURON

Command:

hoc interpreter (oc> prompt)	`quit()`
Python interpreter (>>> prompt)	`exit()`

Keyboard shortcut:

^D (ctrl-D) works for both hoc and Python

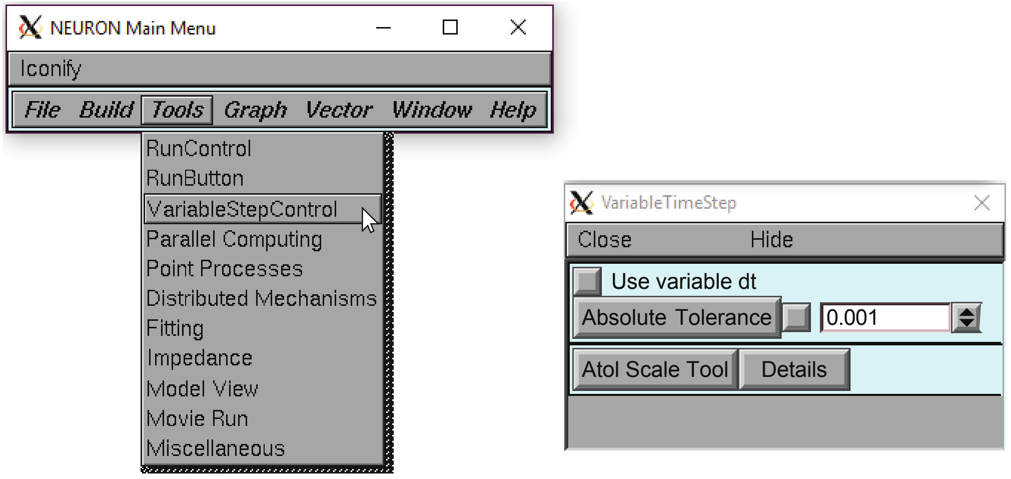
Numerical Methods: Adaptive Integration

Robert A. McDougal

Yale School of Medicine

11 November 2016

Enabling adaptive integration

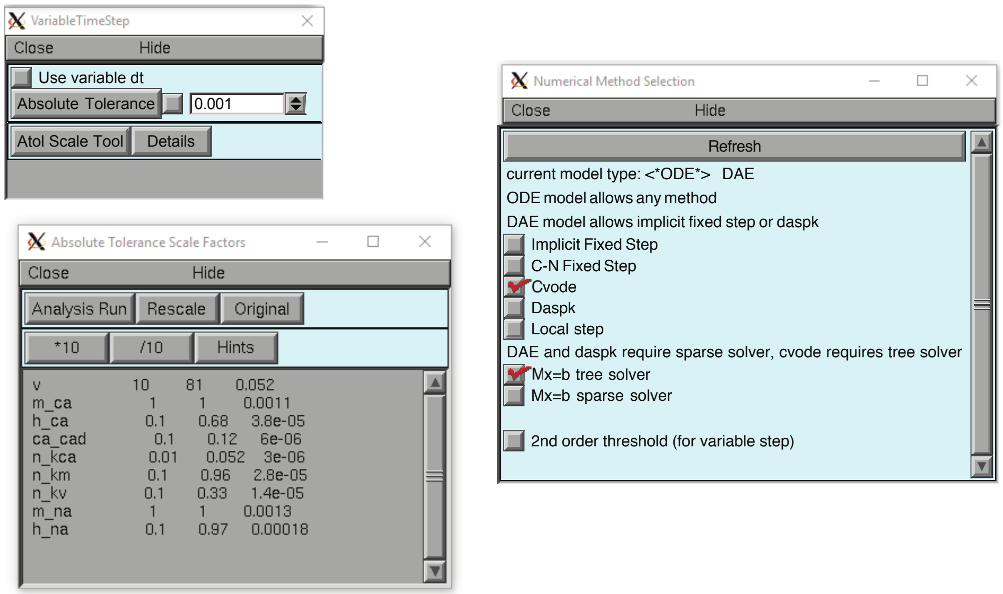


`h.cvode_active(1)`

h.cvode_active is defined in stdrun.hoc which is loaded automatically whenever the gui is imported.

This is a composite image, not a screenshot to allow combining the window decoration and vector-based window contents.

Options: per state variable tolerance, integration methods



VariableTimeStep
Close Hide
Use variable dt
Absolute Tolerance 0.001
Atol Scale Tool Details

Absolute Tolerance Scale Factors
Close Hide
Analysis Run Rescale Original
*10 /10 Hints

```
v          10    81    0.052
m_ca        1     1    0.0011
h_ca       0.1   0.68  3.8e-05
ca_cad     0.1   0.12  6e-06
n_kca      0.01  0.052 3e-06
n_km       0.1   0.96  2.8e-05
n_kv       0.1   0.33  1.4e-05
m_na        1     1    0.0013
h_na       0.1   0.97  0.00018
```

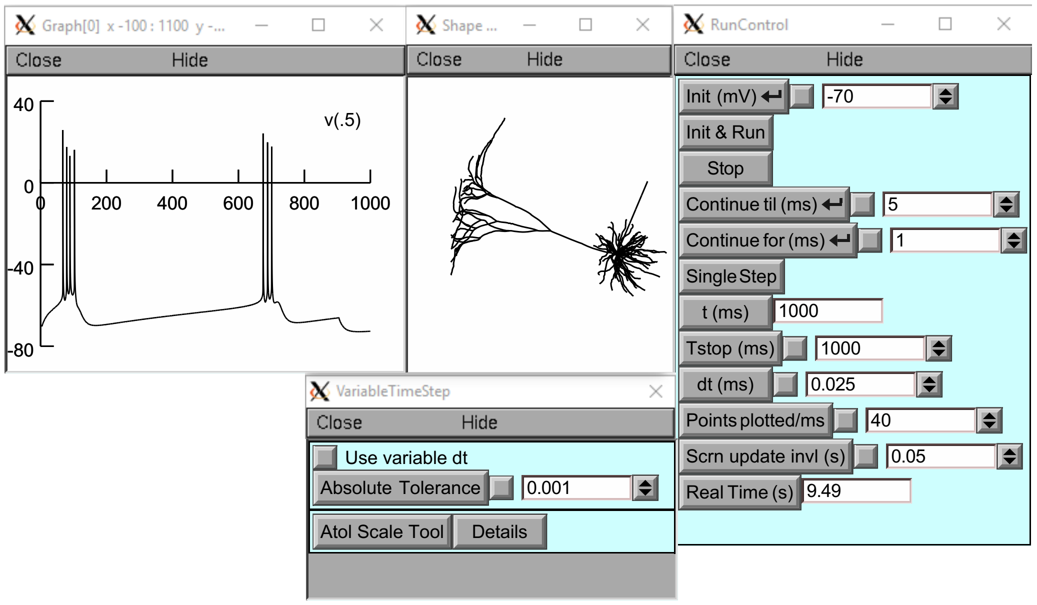
Numerical Method Selection
Close Hide
Refresh
current model type: <*ODE*> DAE
ODE model allows any method
DAE model allows implicit fixed step or daspk
- Implicit Fixed Step
- C-N Fixed Step
- [✓ Cvode]
- Daspk
- Local step

DAE and daspk require sparse solver, cvode requires tree solver
- [✓ Mx=b tree solver]
- Mx=b sparse solver

- 2nd order threshold (for variable step)

This is a composite image, not a screenshot to allow combining the window decoration and vector-based window contents.

Mainen & Sejnowski 1996, Figure 1D, fixed step: 9.49s



RunControl
Close Hide
Init (mV) -70
Init & Run
Stop
Continue til (ms) 5
Continue for (ms) 1
Single Step
t (ms) 1000
Tstop (ms) 1000
dt (ms) 0.025
Points plotted/ms 40
Scrn update invl (s) 0.05
Real Time (s) 9.49

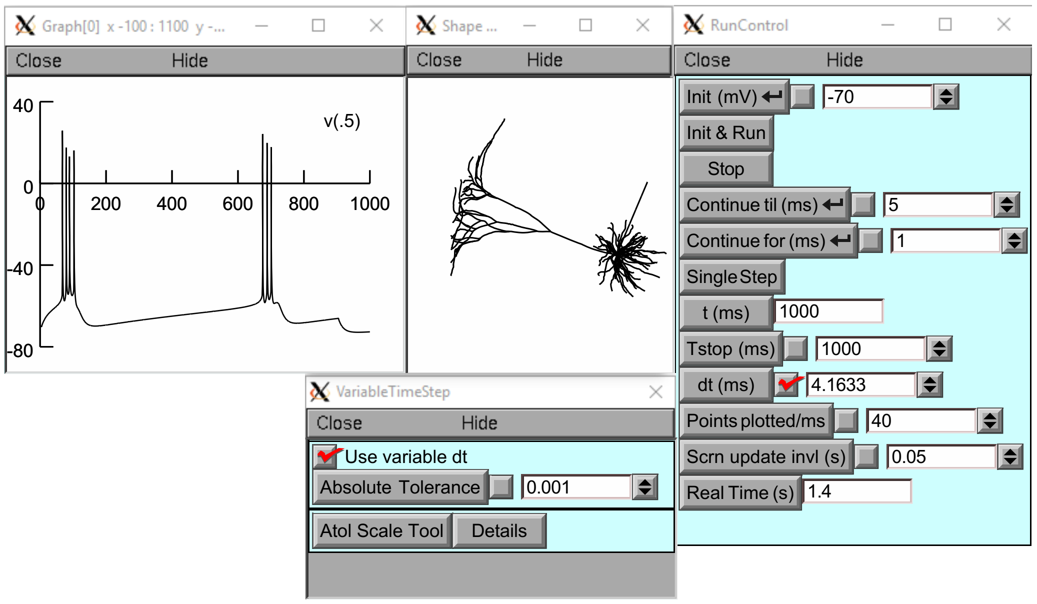
VariableTimeStep
Close Hide
Use variable dt
Absolute Tolerance 0.001
Atol Scale Tool Details

Code for this model is available at: http://modeldb.yale.edu/2488

Timings ran with NEURON 7.5 (cbd6261ecbad) on a 3.4 GHz i7-4770 with 24 GB RAM via the Windows Subsystem for Linux in Windows 10.

This is a composite image, not a screenshot to allow combining the window decoration and vector-based window contents.

Mainen & Sejnowski 1996, Figure 1D, variable step: 1.4s



Code for this model is available at: http://modeldb.yale.edu/2488

Timings ran with NEURON 7.5 (cbd6261ecbad) on a 3.4 GHz i7-4770 with 24 GB RAM via the Windows Subsystem for Linux in Windows 10.

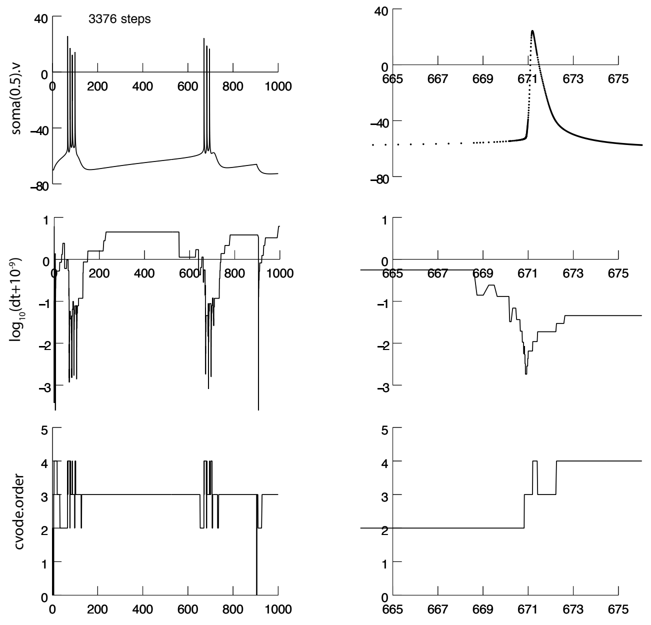
This is a composite image, not a screenshot. Due to pdf rendering problems, the original checkmarks have been replaced.

A closer look at change, time steps, and order

Membrane Potential

Timestep Size

Order of Integration Method



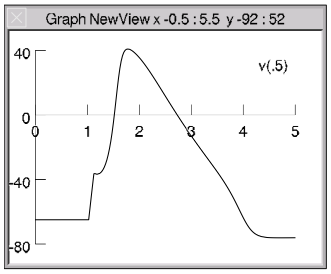
Results shown are for variable step method for Mainen & Sejnowski 1996, Figure 1D.

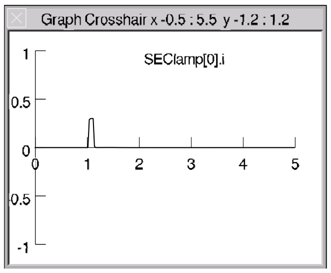
Question

Suppose we inject a current pulse to trigger an action potential that we record at a fixed rate. We then use this time series for a voltage clamp experiment on an identical cell.

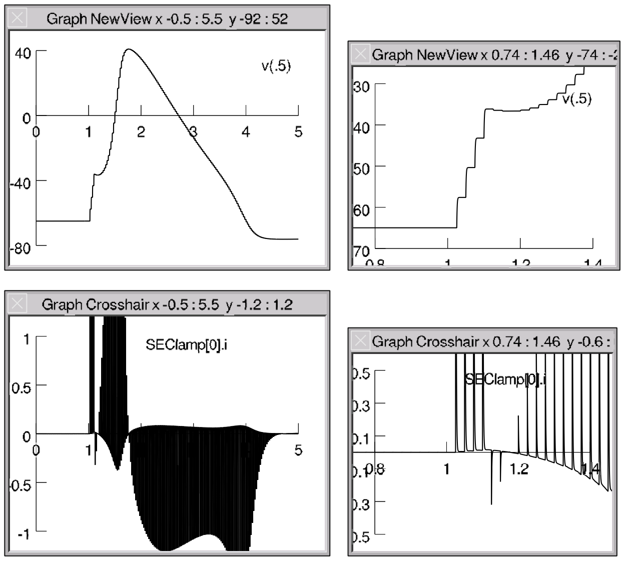
What are the dynamics of the current that must be injected through the voltage clamp?

Fixed step (same timestep)

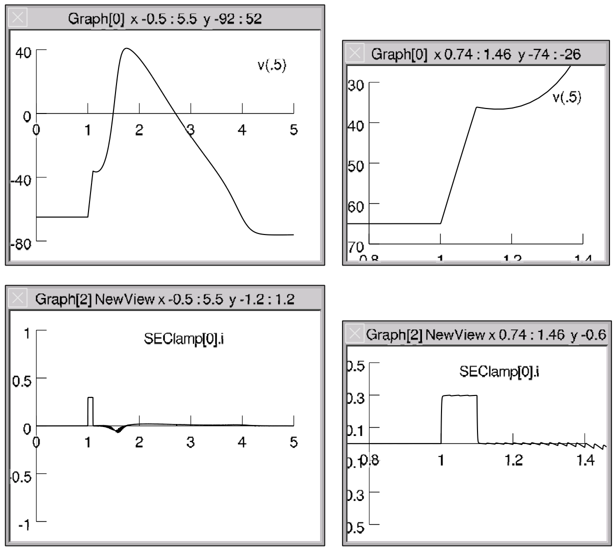




Variable step


Graph NewView x -0.5 : 5.5 y -92 : 52
v(.5)
Graph NewView x 0.74 : 1.46 y -74 : -2
v(.5)
Graph Crosshair x -0.5 : 5.5 y -1.2 : 1.2
SEClamp[0].i
Graph Crosshair x 0.74 : 1.46 y -0.6 :
SEClamp[0].i

Variable step with linear interpolation


Graph[0] x -0.5 : 5.5 y -92 : 52
v(.5)
Graph[0] x 0.74 : 1.46 y -74 : -26
v(.5)
Graph[2] NewView x -0.5 : 5.5 y -1.2 : 1.2
SEClamp[0].i
Graph[2] NewView x 0.74 : 1.46 y -0.6
SEClamp[0].i

```
vvec.play(h.SEClamp[0]._ref_amp1, tvec, 1)
```

The last argument of 1 indicates that the the values at intermediate time points should be estimated by linear interpolation.

Scripting NEURON

Robert A. McDougal

Yale School of Medicine

10 November 2017

What is a script?

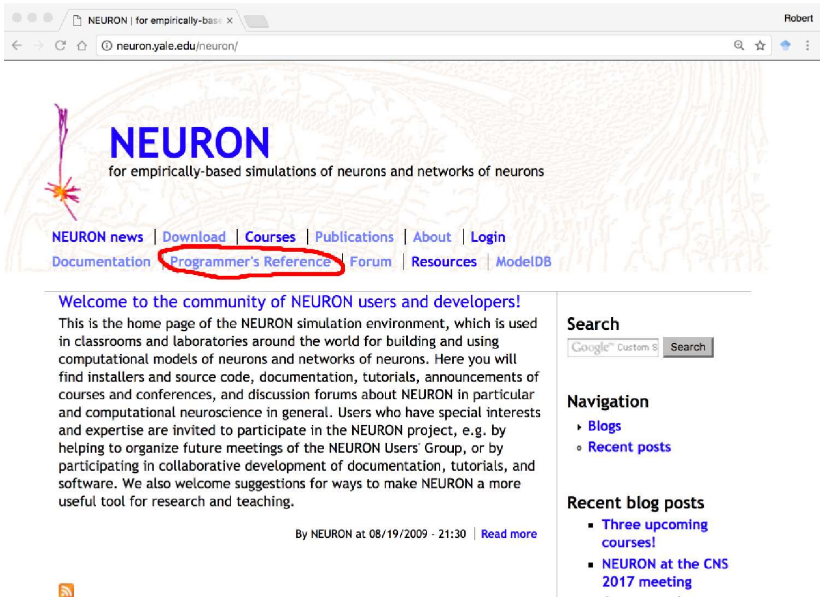
A **script** is a file with computer-readable instructions for performing a task.

In NEURON, scripts can: set-up a model, define and perform an experimental protocol, record data, . . .

Why write scripts for NEURON?

- Automation ensures consistency and reduces manual effort.
- Facilitates comparing the suitability of different models.
- Facilitates repeated experiments on the same model with different parameters (e.g. drug dosages).
- Facilitates recollecting data after change in experimental protocol.
- Provides a complete, reproducible version of the experimental protocol.

Documentation



NEURON | for empirically-base ×

neuron.yale.edu/neuron/

NEURON

for empirically-based simulations of neurons and networks of neurons

NEURON news | Download | Courses | Publications | About | Login
Documentation | Programmer's Reference | Forum | Resources | ModelDB

Welcome to the community of NEURON users and developers!

This is the home page of the NEURON simulation environment, which is used in classrooms and laboratories around the world for building and using computational models of neurons and networks of neurons. Here you will find installers and source code, documentation, tutorials, announcements of courses and conferences, and discussion forums about NEURON in particular and computational neuroscience in general. Users who have special interests and expertise are invited to participate in the NEURON project, e.g. by helping to organize future meetings of the NEURON Users' Group, or by participating in collaborative development of documentation, tutorials, and software. We also welcome suggestions for ways to make NEURON a more useful tool for research and teaching.

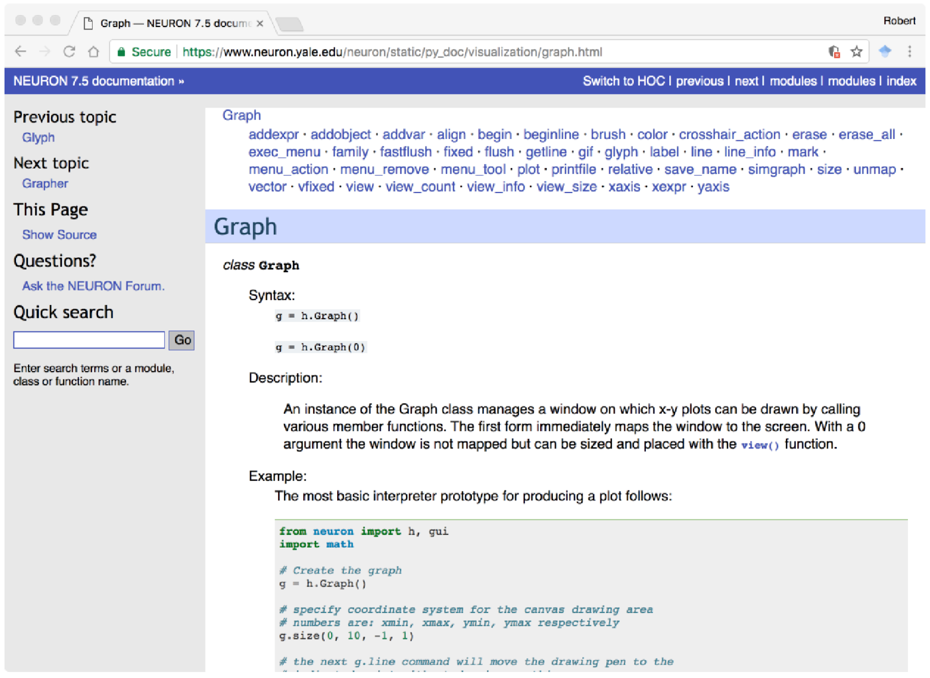
By NEURON at 08/19/2009 - 21:30 | Read more

Search

Navigation
- Blogs
- Recent posts

Recent blog posts
- Three upcoming courses!
- NEURON at the CNS 2017 meeting

neuron.yale.edu



Graph — NEURON 7.5 docume ×

Secure | https://www.neuron.yale.edu/neuron/static/py_doc/visualization/graph.html

NEURON 7.5 documentation » Switch to HOC | previous | next | modules | modules | index

Previous topic
Glyph
Next topic
Grapher
This Page
Show Source
Questions?
Ask the NEURON Forum.
Quick search
Go
Enter search terms or a module, class or function name.

Graph
addexpr · addobject · addvar · align · begin · beginline · brush · color · crosshair_action · erase · erase_all · exec_menu · family · fastflush · fixed · flush · getline · gif · glyph · label · line · line_info · mark · menu_action · menu_remove · menu_tool · plot · printfile · relative · save_name · simgraph · size · unmap · vector · vfixed · view · view_count · view_info · view_size · xaxis · xexpr · yaxis

Graph

class **Graph**

Syntax:

```
g = h.Graph()

g = h.Graph(0)
```

Description:

An instance of the Graph class manages a window on which x-y plots can be drawn by calling various member functions. The first form immediately maps the window to the screen. With a 0 argument the window is not mapped but can be sized and placed with the `view()` function.

Example:

The most basic interpreter prototype for producing a plot follows:

```
from neuron import h, gui
import math

# Create the graph
g = h.Graph()

# specify coordinate system for the canvas drawing area
# numbers are: xmin, xmax, ymin, ymax respectively
g.size(0, 10, -1, 1)

# the next g.line command will move the drawing pen to the
```

Use the "Switch to HOC" link in the upper-right corner of every page if you need documentation for HOC, NEURON's original programming language. HOC may be used in combination with Python: use h.load_file to load a HOC library; the functions and classes are then available with an h. prefix.

Introduction to Python

Displaying results

The print command is used to display non-graphical results.

It can display fixed text:

```
    print ('Hello everyone.')                        Hello everyone.
```

or the results of a calculation:

```
    print (5 * (3 + 2))                                           25
```

Storing results

Give values a name to be able to use them later.

```
     a = max([1.2, 5.2, 1.7, 3.6])
     print (a)                                                   5.2
```

In Python 2.x, print is a keyword and the parentheses are unnecessary. Using the parentheses allows your code to work with both Python 2.x and 3.x.

Don't repeat yourself

Lists and for loops

To do the same thing to several items, put the items in a list and use a `for` loop:

```
numbers = [1, 3, 5, 7, 9]
for number in numbers:
    print (number * number)                    1 9 25 49 81
```

Items can be accessed directly using the [] notation; e.g. `n = number[2]`

To check if an item is in a list, use `in`:

```
print (4 in [3, 1, 4, 1, 5, 9])                        True
print (7 in [3, 1, 4, 1, 5, 9])                        False
```

Dictionaries

If there is no natural order, specify your own keys using a dictionary.

```
data = {'soma': 42, 'dend': 14, 'axon': 'blue'}
print (data['dend'])                                     14
```

Don't repeat yourself

Functions

If there is a particularly complicated calculation that is used once or a simple one used at least twice, give it a name via `def` and refer to it by the name. Return the result of the calculation with the `return` keyword.

```
def area_of_cylinder(diameter, length):
    return 3.14 / 4 * diameter ** 2 * length

area1 = area_of_cylinder(2, 100)
area2 = area_of_cylinder(10, 10)
```

Using libraries

Libraries ("modules" in Python) provide features scripts can use.
To load a module, use `import`:

```
import math
```

Use dot notation to access a function from the module:

```
print (math.cos(math.pi / 3))                              0.5
```

One can also load specific items from a module.
For NEURON, we often want:

```
from neuron import h, gui
```

Other modules

Python ships with a large number of modules, and you can install more (like NEURON). Useful ones for neuroscience include: `math` (basic math functions), `numpy` (advanced math), `matplotlib` (2D graphics), `mayavi` (3D graphics), `pandas` (analysis and databasing), ...

Getting help

To get a list of functions, etc in a module (or class) use `dir`:

```
from neuron import h
print (dir(h))
```

Displays:

```
['APCount', 'AlphaSynapse', 'BBSaveState', 'CVode', 'DEG', 'Deck',
 'E', 'Exp2Syn', 'ExpSyn', 'FARADAY', 'FInitializeHandler',
 'File', 'GAMMA', 'GUIMath', 'Glyph', 'Graph', 'HBox', 'IClamp',
 'Impedance', 'IntFire1', 'IntFire2', 'IntFire4', 'KSChan', ...]
```

To see help information for a specific function, use `help`:

```
help(math.cosh)
```

Python is widely used, and there are many online resources available, including:

- docs.python.org – the official documentation
- Stack Overflow – a general-purpose programming forum
- the NEURON programmer's reference – NEURON documentation
- the NEURON forum – for NEURON-related programming questions

Basic NEURON scripting

Creating and naming sections

A `section` in NEURON is an unbranched stretch of e.g. dendrite.

To create a section, use `h.Section` and assign it to a variable:

```
    apical = h.Section(name='apical')
```

A section can have multiple references to it. If you set `a = apical`, there is still only one section. Use `==` to see if two variables refer to the same section:

```
    print (a == apical)                                    True
```

To access the name, use `.name()`:

```
    print (apical.name())                                apical
```

Also available: a `cell` attribute for grouping sections by cell.

In recent versions of NEURON, named Sections will print with their name; e.g. it suffices to say `print (apical)`.

Making NEURON GUI compatible sections

The NEURON GUI cannot read the names of sections created in Python, which imposes certain limitations to the mouse-based interface.

One work-around is to use the following function which creates a section in HOC and returns a Python Section object:

```
    def Section(name):
        h('create ' + name)
        return getattr(h, name)
```

To make multi-cell simulations fully manipulatable through the GUI, define each cell inside of a HOC Template and wrap that with a Python class.

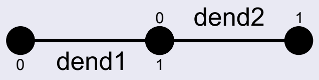
Controlling the GUI from the Python prompt has no such limitations. All graphical functions may be accessed through the command line.

Connecting sections

To reconstruct a neuron's full branching structure, individual sections must be connected using `.connect`:

```
    dend2.connect(dend1(1))
```

Each section is oriented and has a 0- and a 1-end. In NEURON, traditionally the 0-end of a section is attached to the 1-end of a section closer to the soma. In the example above, dend2's 0-end is attached to dend1's 1-end.



To print the topology of cells in the model, use `h.topology()`. The results will be clearer if the sections were assigned names.

```
    h.topology()
```

If no position is specified, then the 0-end will be connected to the 1-end as in the example.

Example

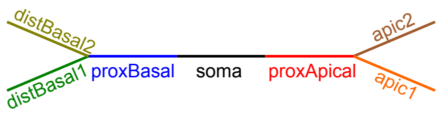
Python script:

```
from neuron import h

# define sections
soma = h.Section(name='soma')
papic = h.Section(name='proxApical')
apic1 = h.Section(name='apic1')
apic2 = h.Section(name='apic2')
pb = h.Section(name='proxBasal')
db1 = h.Section(name='distBasal1')
db2 = h.Section(name='distBasal2')

# connect them
papic.connect(soma)
pb.connect(soma(0))
apic1.connect(papic)
apic2.connect(papic)
db1.connect(pb)
db2.connect(pb)

# list topology
h.topology()
```

Output:

```
|-|       soma(0-1)
   `|       proxApical(0-1)
     `|       apic1(0-1)
     `|       apic2(0-1)
 `|       proxBasal(0-1)
   `|       distBasal1(0-1)
   `|       distBasal2(0-1)
```

Morphology:



Length, diameter, and position

Set a section's length (in μm) with `.L` and diameter (in μm) with `.diam`:

```
sec.L = 20
sec.diam = 2
```

Note: Diameter need not be constant; it can be set per segment.

To specify the $(x, y, z; d)$ coordinates that a section passes through, use e.g. `h.pt3dadd(x, y, z, d, sec=section)`. The section sec has `sec.n3d()` 3D points; their ith x-coordinate is `sec.x3d(i)`. The methods `.y3d`, `.z3d`, and `.diam3d` work similarly.

Warning: the default diameter is based on a squid giant axon and is not appropriate for modeling mammalian cells. Likewise, the temperature (`h.celsius`) is by default 6.3 degrees (appropriate for squid, but not for mammals).

Tip: Define a cell inside a class

Consider the code

```
class Pyramidal:
    def __init__(self):
        self.soma = h.Section(name='soma', cell=self)
```

The `__init__` method is run whenever a new `Pyramidal` cell is created, e.g. via

```
    pyr1 = Pyramidal()
```

The soma can be accessed using dot notation:

```
    print(pyr1.soma.L)
```

By defining a cell in a class, once we're happy with it, we can create multiple copies of the cell in a single line of code.

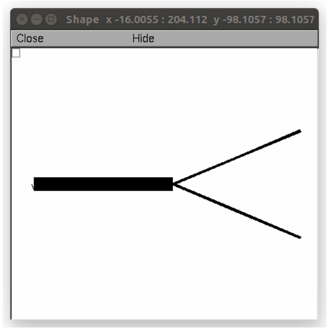
```
    pyr2 = Pyramidal()
```

or even

```
    pyrs = [Pyramidal() for i in range(1000)]
```

Viewing the morphology with h.PlotShape

```
from neuron import h, gui

class Cell:
  def __init__(self):
    main = h.Section(name='main', cell=self)
    dend1 = h.Section(name='dend1', cell=self)
    dend2 = h.Section(name='dend2', cell=self)

    dend1.connect(main)
    dend2.connect(main)

    main.diam = 10
    dend1.diam = 2
    dend2.diam = 2

    # Important: store the sections
    self.main = main; self.dend1 = dend1
    self.dend2 = dend2

my_cell = Cell()

ps = h.PlotShape()
# use 1 instead of 0 to hide diams
ps.show(0)
```



Note: `PlotShape` can also be used to see the distribution of a parameter or variable. To save the PlotShape ps use `ps.printfile('filename.eps')`.

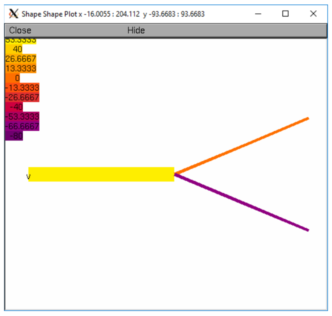
Viewing voltage, sodium, etc

Suppose we make the voltage (`'v'`) nonuniform, which we can do via:

```
my_cell.main.v = 50
my_cell.dend1.v = 0
my_cell.dend2.v = -65
```

We can create a PlotShape that color-codes the sections by voltage:

```
ps = h.PlotShape()
ps.variable('v')
ps.scale(-80, 80)
ps.exec_menu('Shape Plot')
ps.show(0)
```

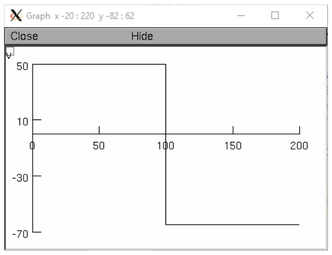


After increasing the spatial resolution:

```
for sec in h.allsec():  sec.nseg = 101
```

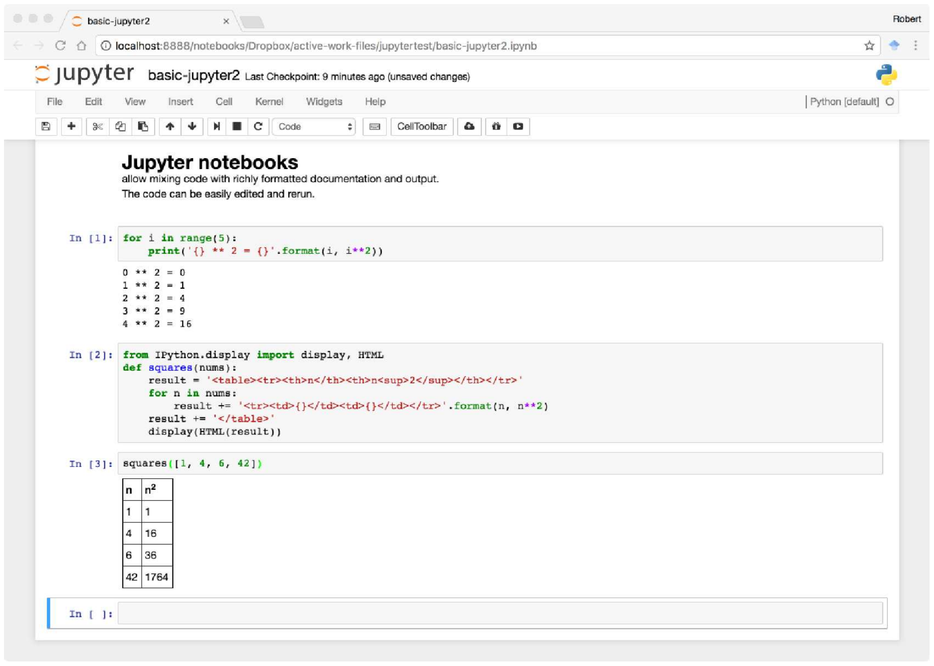
We can plot the voltage as a function of distance from `main(0)` to `dend2(1)`:

```
rvp = h.RangeVarPlot('v')
rvp.begin(0, sec=my_cell.main)
rvp.end(1, sec=my_cell.main)
g = h.Graph()
g.addobject(rvp)
g.exec_menu('View = plot')
```



Sodium concentration could be plotted with `'nai'` instead of `'v'`, etc.

Aside: Jupyter


basic-jupyter2
Robert
localhost:8888/notebooks/Dropbox/active-work-files/jupytertest/basic-jupyter2.ipynb
jupyter
basic-jupyter2 Last Checkpoint: 9 minutes ago (unsaved changes)
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Python [default]
Code
CellToolbar

Jupyter notebooks

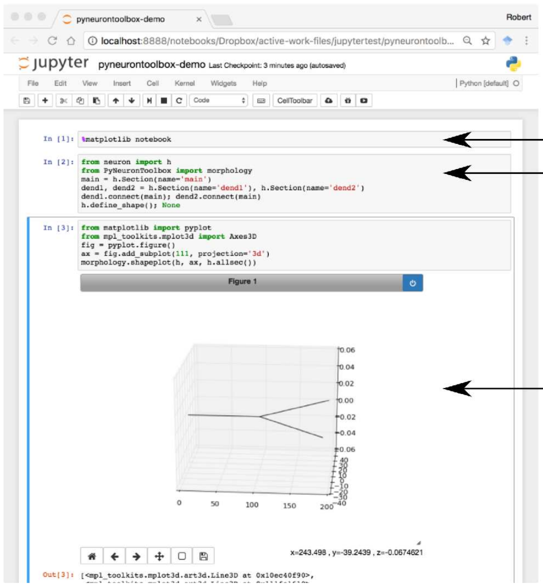
allow mixing code with richly formatted documentation and output.
The code can be easily edited and rerun.

```
In [1]: for i in range(5):
            print('{} ** 2 = {}'.format(i, i**2))

        0 ** 2 = 0
        1 ** 2 = 1
        2 ** 2 = 4
        3 ** 2 = 9
        4 ** 2 = 16

In [2]: from IPython.display import display, HTML
        def squares(nums):
            result = '<table><tr><th>n</th><th>n<sup>2</sup></th></tr>'
            for n in nums:
                result += '<tr><td>{}</td><td>{}</td></tr>'.format(n, n**2)
            result += '</table>'
            display(HTML(result))

In [3]: squares([1, 4, 6, 42])
```

n	n^2
1	1
4	16
6	36
42	1764

```
In [ ]:
```

Aside: Jupyter


pyneurontoolbox-demo
Robert
localhost:8888/notebooks/Dropbox/active-work-files/jupytertest/pyneurontoolb...
jupyter pyneurontoolbox-demo Last Checkpoint: 3 minutes ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Python [default]
Code
CellToolbar
Figure 1
x=243.498 , y=-39.2439 , z=-0.0674621

```
In [1]: %matplotlib notebook

In [2]: from neuron import h
        from PyNeuronToolbox import morphology
        main = h.Section(name='main')
        dend1, dend2 = h.Section(name='dend1'), h.Section(name='dend2')
        dend1.connect(main); dend2.connect(main)
        h.define_shape(); None

In [3]: from matplotlib import pyplot
        from mpl_toolkits.mplot3d import Axes3D
        fig = pyplot.figure()
        ax = fig.add_subplot(111, projection='3d')
        morphology.shapeplot(h, ax, h.allsec())

Out[3]: [<mpl_toolkits.mplot3d.art3d.Line3D at 0x10ec40f90>,
```

magic method that makes matplotlib graphs interactive

PyNeuronToolbox is Alex Williams' set of utility functions for working with NEURON models in Python.

(Additional options can be passed to colorize segments which can be used to encode information about e.g. membrane potential.)

interactive graph (rotate, zoom, save, etc)

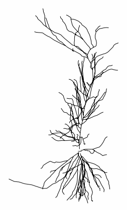
Cannot run new commands until interactive mode is turned off (blue button) at which point the graph becomes static.

https://github.com/ahwillia/PyNeuron-Toolbox

Loading morphology from an swc file

To create pyr, a Pyramidal cell with morphology from the file c91662.swc:

```
from neuron import h, gui
h.load_file('import3d.hoc')

class Pyramidal:
    def __init__(self):
        self.load_morphology()
        # do discretization, ion channels, etc
    def load_morphology(self):
        cell = h.Import3d_SWC_read()
        cell.input('c91662.swc')
        i3d = h.Import3d_GUI(cell, 0)
        i3d.instantiate(self)

pyr = Pyramidal()
```



pyr has lists of Sections: `pyr.apic`, `.axon`, `.soma`, and `.all`. Each Section has the appropriate `.name()` and `.cell()`.

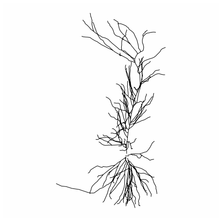
Only do this in code after you've already examined the cell with the Import3D GUI tool and fixed any issues in the SWC file.

Working with multiple cells

Suppose `Pyramidal` is defined as before and we create several copies:

```
mypyrs = [Pyramidal(i) for i in range(10)]
```

We then view these in a shape plot:



Where are the other 9 cells?

Working with multiple cells

To can create a method to reposition a cell and call it from `__init__`:

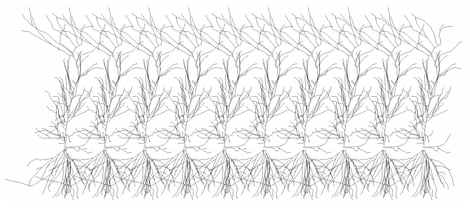
```
class Pyramidal:
  def _shift(self, x, y, z):
    soma = self.soma[0]
    n = soma.n3d()
    xs = [soma.x3d(i) for i in range(n)]
    ys = [soma.y3d(i) for i in range(n)]
    zs = [soma.z3d(i) for i in range(n)]
    ds = [soma.diam3d(i) for i in range(n)]
    for i, (a, b, c, d) in enumerate(zip(xs, ys, zs, ds)):
      h.pt3dchange(i, a + x, b + y, c + z, d, sec=soma)
```

```
def __init__(self, gid, x, y, z):
  self._gid = gid
  self.load_morphology()
  self._shift(x, y, z)

def load_morphology(self):
  cell = h.Import3d_SWC_read()
  cell.input('c91662.swc')
  i3d = h.Import3d_GUI(cell, 0)
  i3d.instantiate(self)
```

Now if we create ten, while specifying offsets,

```
mypyrs = [Pyramidal(i, i * 100, 0, 0) for i in range(10)]
```

The PlotShape will show all the cells separately:



Does position matter?

Sometimes.

Position matters with:

- Connections based on proximity of axon to dendrite.
- Connections based on cell-to-cell proximity.
- Extracellular diffusion.
- Communicating about your model to other humans.

Distributed mechanisms

Use `.insert` to insert a distributed mechanism into a section. e.g.

```
axon.insert('hh')
```

Point processes

To insert a point process, specify the segment when creating it, and save the return value. e.g.

```
pp = h.IClamp(soma(0.5))
```

To find the segment containing a point process pp, use

```
seg = pp.get_segment()
```

The section is then `seg.sec` and the normalized position is `seg.x`.

The point process is removed when no variables refer to it.

Use `List` to find out how many point processes of a given type have been defined:

```
all_iclamp = h.List('IClamp')
print ('Number of IClamps:')
print (all_iclamp.count())
```

Setting and reading parameters

In NEURON, each section has normalized coordinates from 0 to 1.
To read the value of a parameter defined by a range variable at a given normalized position use: section(x).MECHANISM.VARNAME
e.g.

```
gkbar = apical(0.2).hh.gkbar
```

Setting variables works the same way:

```
apical(0.2).hh.gkbar = 0.037
```

To specify how many evenly-sized pieces (segments) a section should be broken into (each potentially with their own value for range variables), use `section.nseg`:

```
apical.nseg = 11
```

To specify the temperature, use `h.celsius`:

```
h.celsius = 37
```

Setting and reading parameters

Often you will want to read or write values on all segments in a section. To do this, use a `for` loop over the Section:

```
for segment in apical:
    segment.hh.gkbar = 0.037
```

The above is equivalent to `apical.gkbar_hh = 0.037`, however the first version allows setting values nonuniformly.

A list comprehension can be used to create a Python list of all the values of a given property in a segment:

```
apical_gkbars = [segment.hh.gkbar for segment in apical]
```

Note: looping over a Section only returns true Segments. If you want to include the voltage-only nodes at 0 and 1, iterate over, e.g. `apical.allseg()` instead.

HOC's for (x,0) and for (x) are equivalent to looping over a section and looping over allseg, respectively.

Running simulations

Basics

To initialize a simulation to -65 mV:

```
h.finitialize(-65)
```

To run a simulation until $t = 50$ ms:

```
h.continuerun(50)
```

Additional `h.continuerun` calls will continue from the last time.

Ways to improve accuracy

Reduce time steps via, e.g. `h.dt = 0.01`
Enable variable step (allows error control): `h.CVode().active(True)`
Increase the discretization resolution: `sec.nseg = 11`

To increase nseg for all sections:

```
  for sec in h.allsec():  sec.nseg *= 3
```

Recording data

To see how a variable changes over time, create a `Vector` to store the time course:

```
    data = h.Vector()
```

and do a `.record` with the last part of the name prefixed by `_ref_`.

e.g. to record `soma(0.3).ina`, use

```
    data.record(soma(0.3)._ref_ina)
```

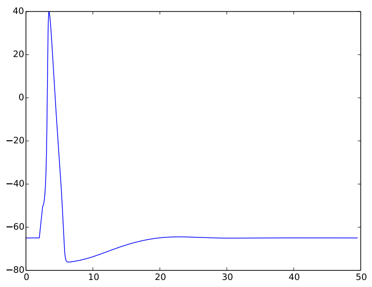
Tips

- Be sure to also record `h._ref_t` to know the corresponding times.
- `.record` must be called before `h.finitialize()`.

If `v` is a `Vector`, then `v.as_numpy()` provides the equivalent `numpy` array; that is, changing one changes the other.

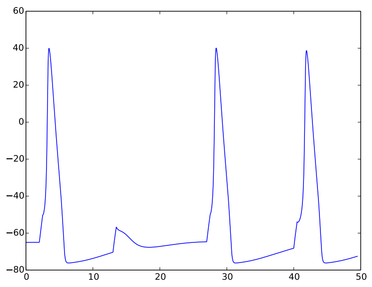
Example: Hodgkin-Huxley

```
from neuron import h, gui
from matplotlib import pyplot

# morphology and dynamics
soma = h.Section(name='soma')
soma.insert('hh')

# current clamp
i = h.IClamp(soma(0.5))
i.delay = 2 # ms
i.dur = 0.5 # ms
i.amp = 50

# recording
t = h.Vector()
v = h.Vector()
t.record(h._ref_t)
v.record(soma(0.5)._ref_v)

# simulation
h.finitialize(-65)
h.continuerun(49.5)

# plotting
pyplot.plot(t, v)
pyplot.show()
```



A spike occurs whenever V_m crosses some threshold (e.g. 0 mV).
Python can easily find all spike times. Only changes from the previous example are highlighted.

```
from neuron import h, gui
from matplotlib import pyplot
soma = h.Section(name='soma')
soma.insert('hh')
# current clamps
iclamps = []
for t in [2, 13, 27, 40]:
    i = h.IClamp(soma(0.5))
    i.delay = t # ms
    i.dur = 0.5 # ms
    i.amp = 50
    iclamps.append(i)
# recording
t = h.Vector()
v = h.Vector()
t.record(h._ref_t)
v.record(soma(0.5)._ref_v)
# simulation
h.finitialize(-65)
h.continuerun(49.5)
# compute spike times
st = [t[j] for j in range(len(v) - 1)
      if v[j] <= 0 and v[j + 1] > 0]
print ('spike times:')
print (st)
# plotting
pyplot.plot(t, v)
pyplot.show()
```



The console displays:

```
spike times:
[3.1750000000000114, 28.149999999998936,
41.6250000000009]
```

That is, the cell spiked at: 3.175 ms, 28.150 ms, and 41.625 ms.

Interspike intervals (ISIs) are the delays between spikes; that is, they are the differences between consecutive spike times.

To display ISIs for the previous example, we add the lines:

```
isis = [next - last for next, last in zip(st[1:], st[:-1])]
print ('ISIs:'); print (isis)
```

The result:

```
[24.974999999998925, 13.475000000001966]
```

That is, the delays between spikes were 24.975 ms and 13.475 ms.

Networks of neurons

Suppose we have the simple neuron model:

```
from neuron import h, gui

class Cell:
    def __init__(self):
        self.soma = h.Section(name='soma', cell=self)
        self.soma.insert('hh')
```

and two cells:

```
neuron1 = Cell()
neuron2 = Cell()
```

one of which is stimulated by a current clamp:

```
ic = h.IClamp(neuron1.soma(0.5))
ic.amp = 50
ic.delay = 2 # ms
ic.dur = 0.5 # ms
```

A synapse from that cell to the other may cause the second cell to fire when the first cell is stimulated. In NEURON, the post-synaptic side of the synapse is a point process; presynaptic threshold detection is done with an `h.NetCon`.

Networks of neurons

Setup the post-synaptic side:

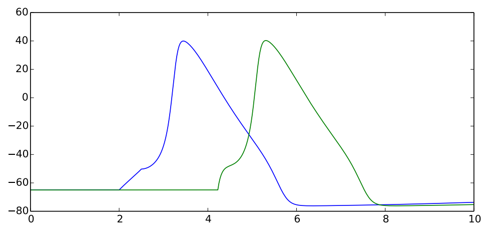
```
postsyn = h.ExpSyn(neuron2.soma(0.5))
postsyn.e = 0  # reversal potential
```

Setup the presynaptic side, transmission delay, and synaptic weight:

```
syn = h.NetCon(neuron1.soma(0.5)._ref_v, postsyn, sec=neuron1.soma)
syn.delay = 1
syn.weight[0] = 5
```

Then we can setup recording, run, and plot as usual:

```
t, v1, v2 = h.Vector(), h.Vector(), h.Vector()
t.record(h._ref_t)
v1.record(neuron1.soma(0.5)._ref_v)
v2.record(neuron2.soma(0.5)._ref_v)

h.finitialize(-65)
h.continuerun(10)

from matplotlib import pyplot
pyplot.plot(t, v1, t, v2)
pyplot.xlim((0, 10))
pyplot.show()
```



h.ExpSyn is one of several general synapse types distributed with NEURON; additional ones may be specified in NMODL or downloaded from ModelDB.

The use of h.NetCon must be modified slightly to support parallel simulation; this is discussed in a different presentation.

Storing data to CSV to share with other tools

The CSV format is widely supported by mathematics, statistics, and spreadsheet programs and offers an easy way to pass data back-and-forth between them and NEURON.

In Python, we can use the csv module to read and write csv files.

Adding the following code after the continuerun in the example will create a file data.csv containing the course data.

```
import csv
with open('data.csv', 'wb') as f:
    csv.writer(f).writerows(zip(t, v))
```

Each row in the file corresponds to one time point. The first column contains t values; the second contains v values. Additional columns can be stored by adding them after the t, v.

For more complicated data storage needs, consider the pandas or h5py modules. Unlike csv, these must be installed separately.

Version control

Version control: git

Why use version control?

- **Protects against losing working code**: if something used to work but no longer does, you can test previous versions to identify what change caused the error.
- **Provides a record of script history**: authorship, changes, . . .
- **Promotes collaboration**: provides tools to combine changes made independently on different copies of the code.

Version control: git basics

Setup

```
git init
```

Declare files to be tracked

```
git add FILENAME
```

Commit a version (so can return to it later)

```
git commit -a
```

Return to the version of FILENAME from 2 commits ago

```
git checkout HEAD~2 FILENAME
```

Version control: git

View list of changes

```
git log
```

Remove a file from tracking

```
git rm FILENAME
```

Rename a tracked file

```
git mv OLDNAME NEWNAME
```

Version control: git and remote servers

`git` (and mercurial) is a distributed version control system, designed to allow you to collaborate with others. You can use your own server or a public one like github or bitbucket.

Download from a server

```
git clone http://URL.git
```

Get changes from server and merge with local changes

```
git pull
```

Sync local, committed changes to the server

```
git push
```

Version control: syncing data with code

One simple way to ensure you always know what version of the code generated your data is to include the git hash in the filename. The following function can help:

```
def git_hash():
    import subprocess
    suffix = ''
    if subprocess.check_output(['git', 'diff']):
        suffix = '+'
    return '%s%s' % (subprocess.check_output([
        'git', 'log', '-1', '--pretty=format:%h']),
        suffix)
```

Then, for example, save matplotlib graphics with:

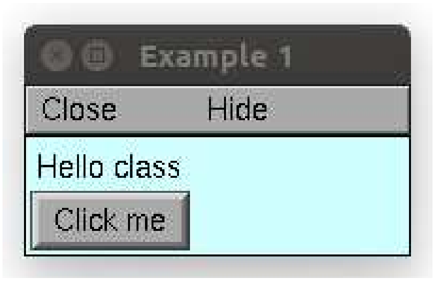
```
    pyplot.savefig('filename_' + git_hash() + '.pdf')
```

GUI development

Making your own graphical interface

- To ensure your GUI responds to user input, be sure to: `from neuron import gui`
- Place basic widgets (text, buttons, checkboxes, . . .) in an `h.xpanel`.

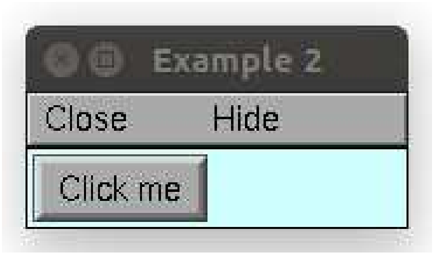
```
from neuron import h, gui

h.xpanel('Example 1')
h.xlabel('Hello class')
h.xbutton('Click me')
h.xpanel()
```



Button actions

To perform an action when a button is pressed, write it as a function, and then pass the function to h.xbutton.

```
from neuron import h, gui

def say_hello():
    print 'hello!'

h.xpanel('Example 2')
h.xbutton('Click me',
          say_hello)
h.xpanel()
```



Pressing the button displays:

```
hello!
```

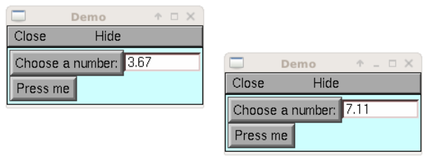
Pressing the button twice:

```
hello!
hello!
```

Number fields and classes

Place your GUI commands in a class to allow independent reuse.

```
from neuron import h, gui
class Demo:
    def __init__(self):
        self.value = 7.18
        h.xpanel('Demo')
        h.xvalue('Choose a number:',
            (self, 'value'))
        h.xbutton('Press me',
            self.print_value)
        h.xpanel()
    def print_value(self):
        print ('You chose:')
        print (self.value)

# make two demos
d1 = Demo()
d2 = Demo()
```



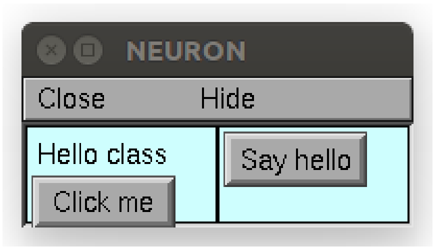
Clicking "Press me" on the left window and then on the right window displays:

```
You chose:
3.67
You chose:
7.11
```

Layout: HBox and VBox

Combine windows horizontally with HBox and vertically with VBox.

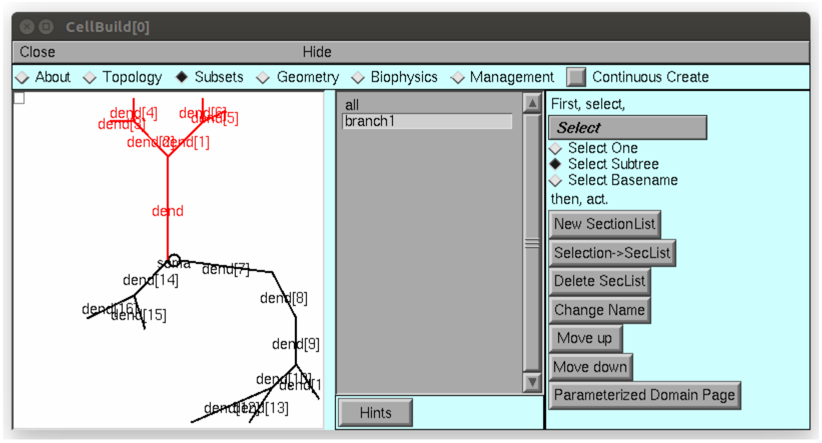
```
from neuron import h, gui
hbox = h.HBox()
hbox.intercept(1)
h.xpanel('Example 1')
h.xlabel('Hello class')
h.xbutton('Click me')
h.xpanel()
h.xpanel('Example 3')
h.xbutton('Say hello')
h.xpanel()
h.xpanel()
hbox.intercept(0)
hbox.map()
```



Note: HBox and VBox can contain: H/VBox, Deck, xpanel, Graph, . . .

Layout: HBox and VBox

Complicated layouts can be constructed using nested VBox and HBox objects:



For more information

For more background and a step-by-step guide to creating a network model, see the NEURON + Python tutorial at:

http://neuron.yale.edu/neuron/static/docs/neuronpython/index.html

The NEURON Python programmer's reference is available at:

http://neuron.yale.edu/neuron/static/py_doc/index.html

Ask questions on the NEURON forum:

http://neuron.yale.edu/phpbb

Building, Running, and Visualizing Parallel NEURON Models

Robert A. McDougal

Yale School of Medicine

10 November 2017

Why use parallel computation?

Three reasons:

- Get the results for a simulation in less real time.
- Run a larger simulation in the same amount of time.
- Run models needing more memory than is available on one machine.

What are the downsides?

Parallel models introduce:

- Greater programming complexity.
- New kinds of bugs.

You have to decide if the time spent parallelizing your model can be recovered.

Other considerations

The 16384 core EPFL IBM BlueGene/P can theoretically do as many calculations in 1 hour at 850 MHz as a 3 GHz desktop computer can do in 6 months.

Building a parallelizable model typically requires little extra effort from building a serial model; converting a serial model to a parallel model is often more difficult.

Three main classes of parallel problems

Parameter sweeps

Running the same (typically fast) simulation 1000s of times with different parameters is an example of an *embarrassingly parallel* problem. NEURON supports this natively with bulletin boards; Calin-Jageman and Katz (2006) developed a screen saver solution.

Distributing networks across processors

Cells can communicate by

- logical spike events with significant axonal, synaptic delay.
- postsynaptic conductance depending continuously on presynaptic voltage.
- gap junctions.

Distributing single cells across processors

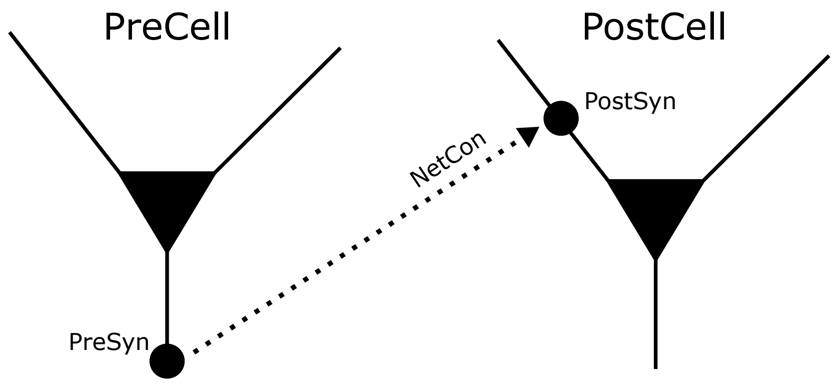
The *multisplit* method distributes portions of the tree cable equation across different machines.

A parallel model can fall in 1, 2, or 3 of these classes.

Some parallel philosophy

- A network of neurons is composed of many individual neurons of potentially many cell types. As much as possible, design and debug each cell type separately before building the network.
- A simulation should give the same results regardless of the number of processors used to run it.
- When possible, parameterize your network so you can run a small test first.

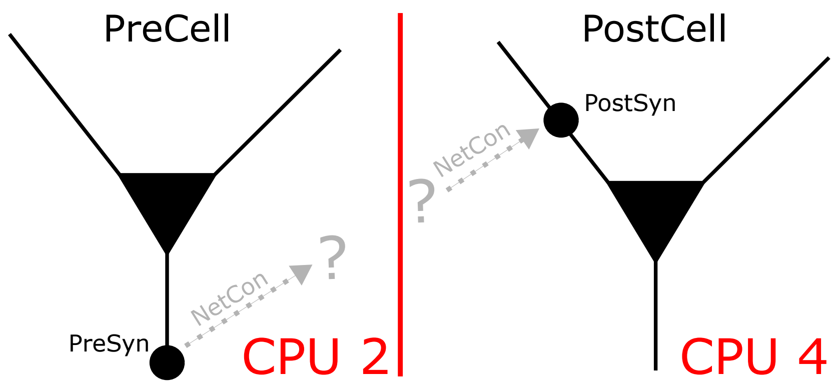
Synaptic connections with one processor



```
nc = h.NetCon(PreSyn, PostSyn, sec=presyn_section)
nc.delay = 1
```

Delay is measured in ms.
We can also set: `nc.weight` and `nc.threshold[]`.
PreSyn is a pointer, e.g. `soma(0.5)._ref_v`; PostSyn is a point process e.g. an instance of `h.ExpSyn`.

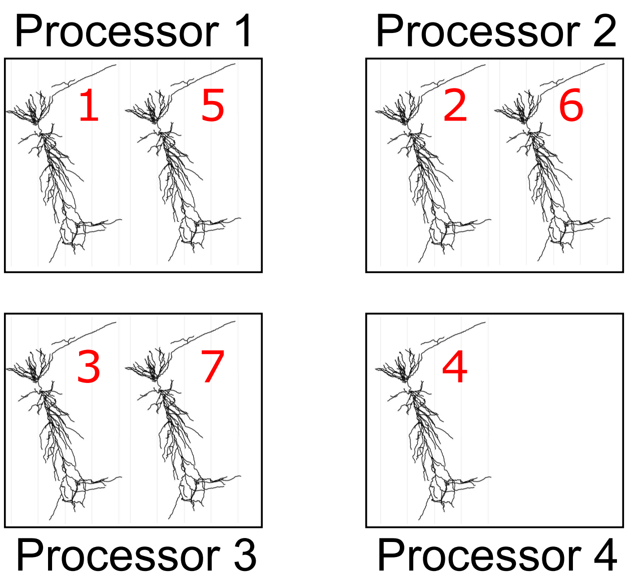
If cells in different processes, a different approach is needed



The `ParallelContext` object facilitates building parallel models.

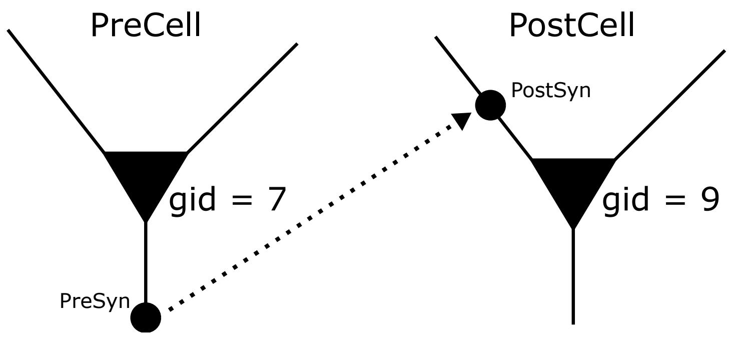
```
pc = h.ParallelContext()
```

Every spike source **must** have a GID.

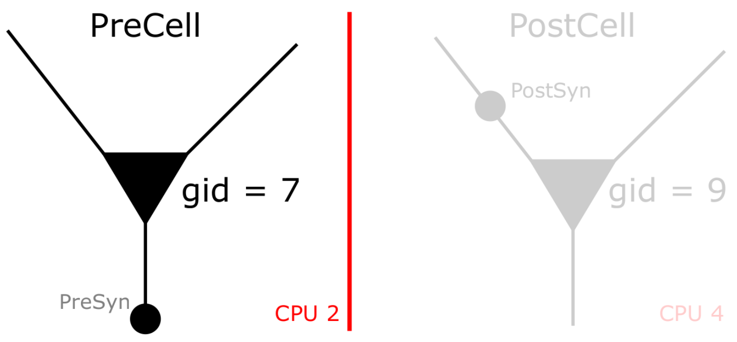


Note: to ensure the model produces identical results regardless of the number of processors, also use GIDs to selecting random streams (e.g. `Random123`).

Building synapses



Configuring the presynaptic connection site



Create cell only where the gid exists:

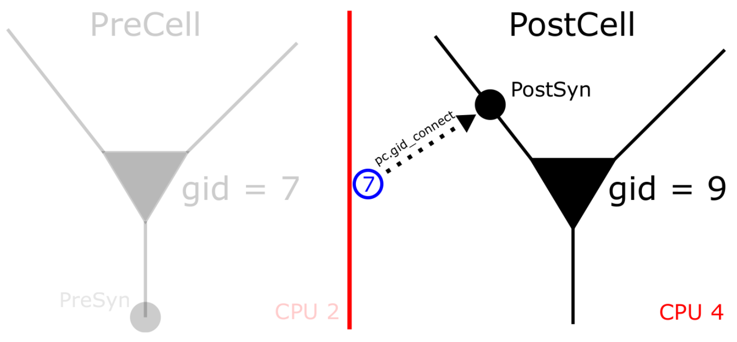
```
if pc.gid_exists(7):
    PreCell = Cell()
```

Associate gid with spike source:

```
nc = h.NetCon(PreSyn, None, sec=presec)
pc.cell(7, nc)
```

PreSyn here is a **pointer**, e.g. `PreCell.soma(0.5)._ref_v`

Configuring the postsynaptic connection site

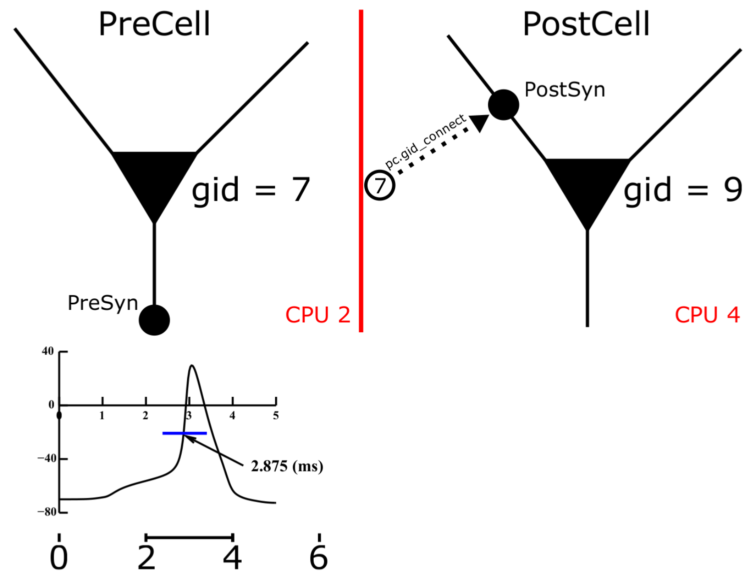


Create NetCon on node where target exists:

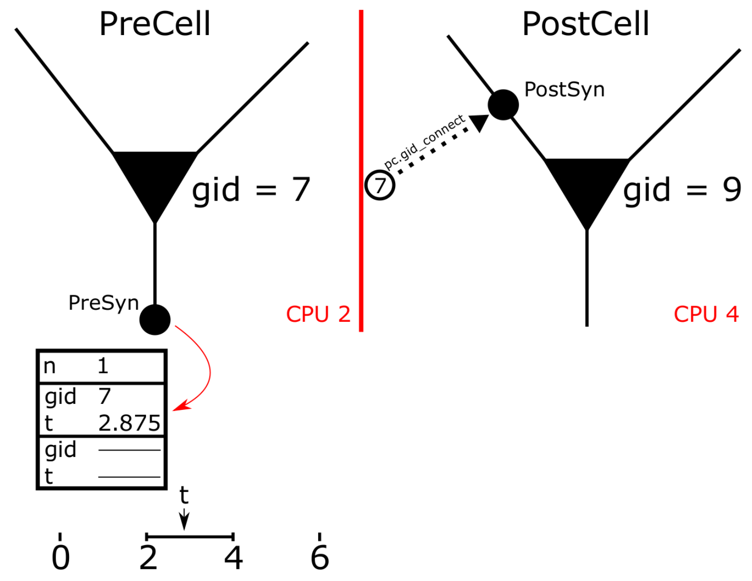
```
nc = pc.gid_connect(7, PostSyn)
```

PostSyn here is a Point Process, e.g. an ExpSyn.

Spike exchange method

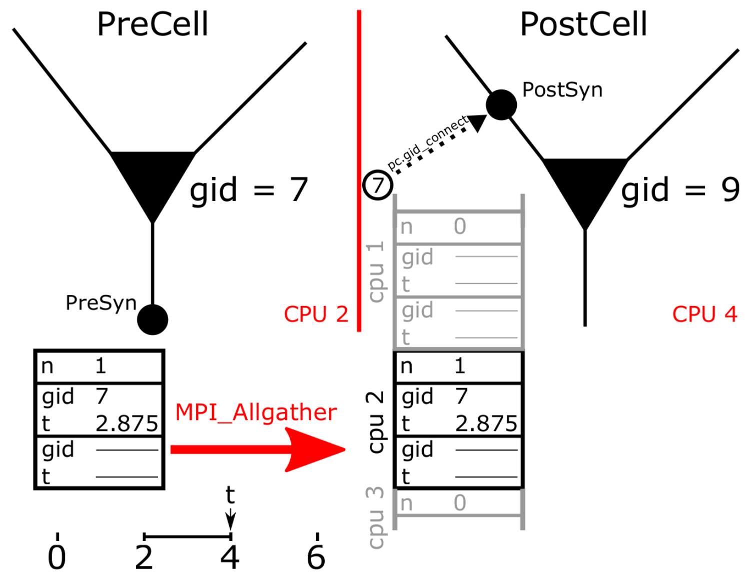


Spike exchange method

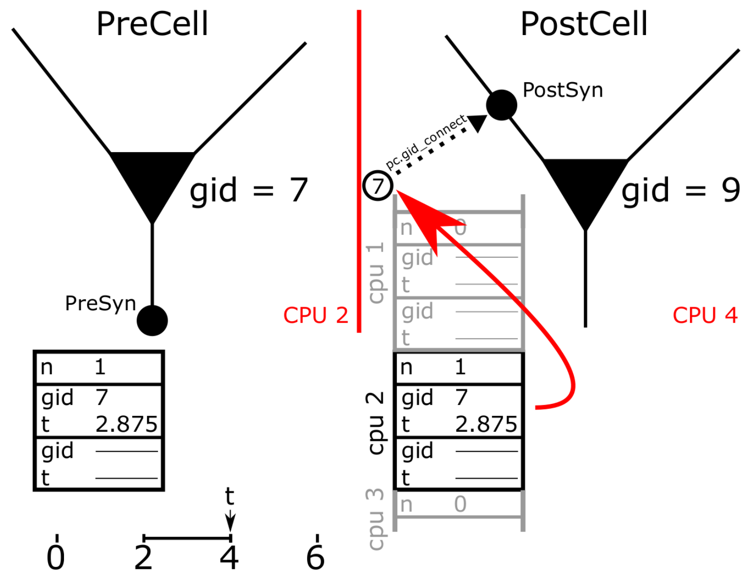


n	1
gid	7
t	2.875
gid	———
t	———

Spike exchange method



Spike exchange method

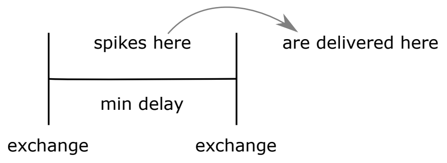


Exploit transmission delays: using pc.set_maxstep

Run using the idiom:

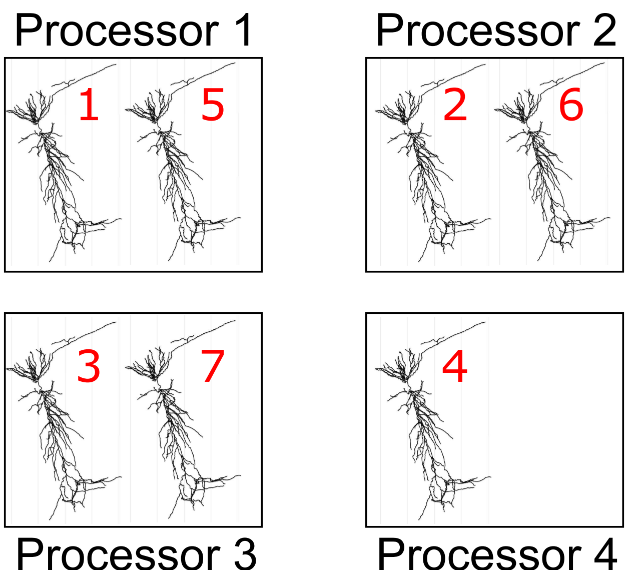
```
pc.set_maxstep(10)
h.stdinit()
pc.psolve(tstop)
```

NEURON will pick an event exchange interval equal to the smaller of all the NetCon delays and of the argument to pc.set_maxstep. In general, larger intervals are better because they reduce communication overhead.



pc.set_maxstep must be called on each node; it uses MPI_Allreduce to determine the minimum delay.

Simple load-balancing strategy: round-robin.



Simple load-balancing strategy: round-robin.

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

An efficient way to distribute, especially if all cells similar:

```
for gid in range(int(pc.id()), ncell, int(pc.nhost())):
    pc.set_gid2node(gid, pc.id())
    ...
```

(Note: the body is executed at most $\lceil$`ncell/nhost`$\rceil$ times, not `ncell`.)

Advanced load-balancing: balance work not number of cells

Strategy:

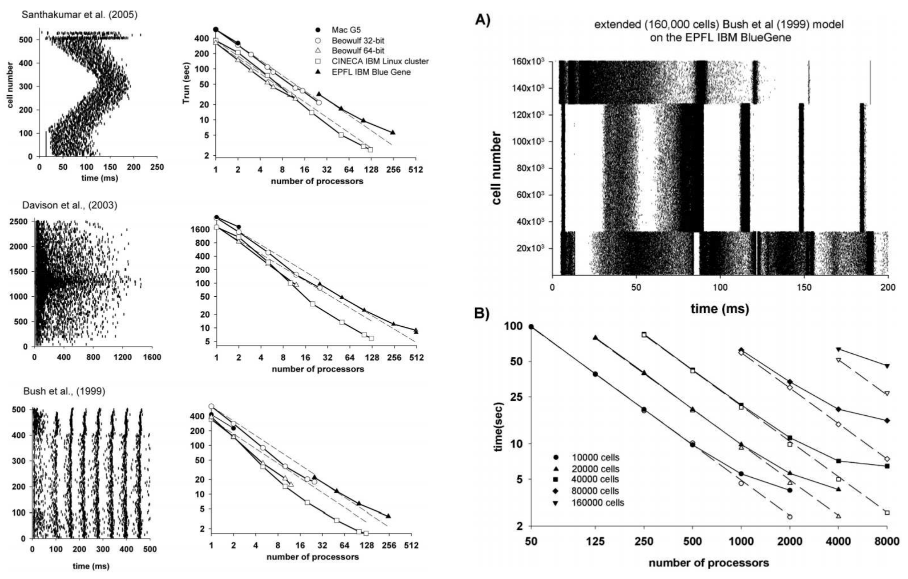
- Distribute cells round-robin to all processors, instantiate them.
- Compute an estimate of the computational complexity:

```
def complexity(self):
    h.load_file('loadbal.hoc')
    lb = h.LoadBalance()
    return lb.cell_complexity(sec=self.all[0])
```

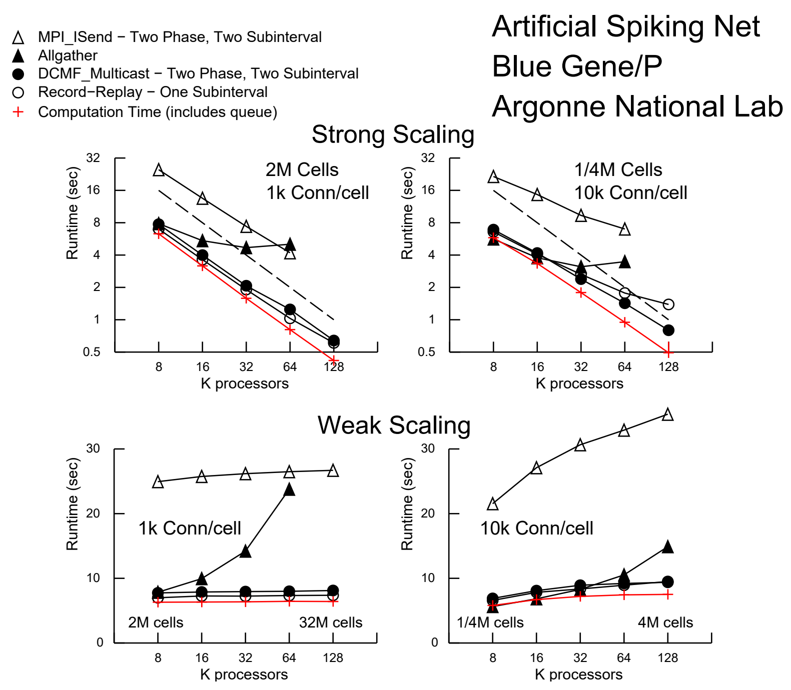
- Destroy the cells, send the gid-complexity data to node 0.
- (On node 0): distribute gids such that the next gid goes to the node with the least amount of complexity.
- Send the gids to the nodes; instantiate the cells.

For a more accurate (but computationally more intensive) estimate of complexity, use lb.ExperimentalMechComplex and lb.read_complex.

Performance: MPI scaling



Performance: Spike exchange strategies

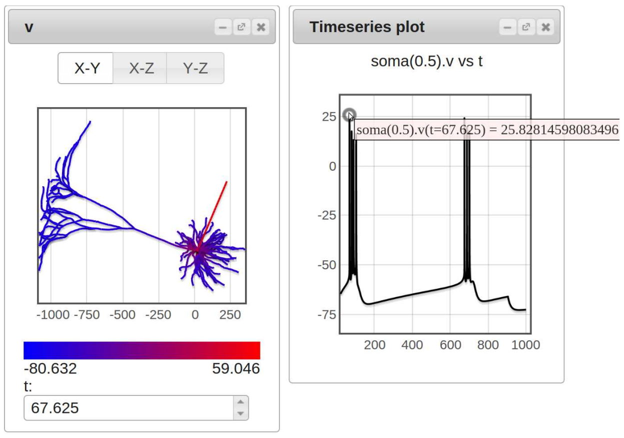


Performance Tip

Tip: For network models, use a fixed step solver and not a variable step solver.

Tip: Store synaptic events; recreate single cells as needed

$$\begin{tikzcd}\begin{array}{c}\text{initial conditions}\\ +\\ \text{synaptic events}\end{array} \arrow[r] & \text{neuron dynamics}\end{tikzcd}$$



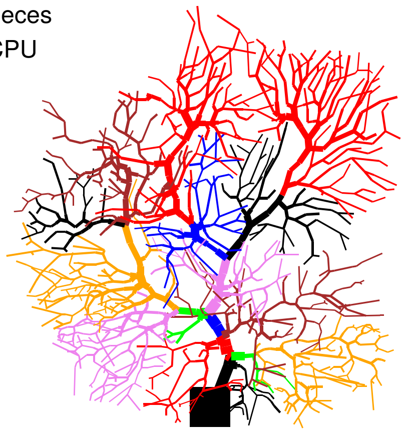
Use `NetCon.record` method to store spike times. Save them as e.g. JSON. Play them back into a single cell simulation using `VecStim`.

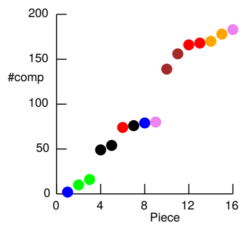
VecStim is defined in `vecevent.mod` which is available at https://github.com/nrnhines/nrn/blob/master/share/examples/nrniv/netcon/vecevent.mod

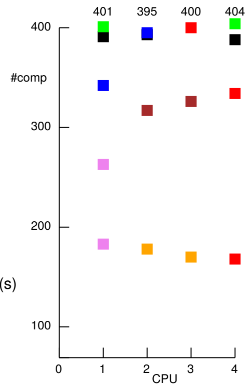
Multisplit

Improve load balancing with multisplit

16 Pieces
4 CPU







	Time (s)	
CPU	Computation	Exchange
0	13.82	0.56
1	13.35	1.03
2	13.47	0.90
3	13.56	0.82

	Runtime(s)
16 pieces, 1 cpu	55.0
wholecell, 1 cpu	56.2
16 pieces, 4 cpu	14.4

Multisplit algorithm described in Hines et al 2008. DOI: 10.1007/s10827-008-0087-5

Using multisplit (MPI)

For process-based multisplit (with MPI), use `pc.multisplit` to declare split nodes:

```
pc.multisplit(x, subtreeid, sec=sec)
```

After all split nodes are declared, **every** process must execute:

```
pc.multisplit()
```

If created, destroy any parts of the cell that do not belong on the processor.

Rules:

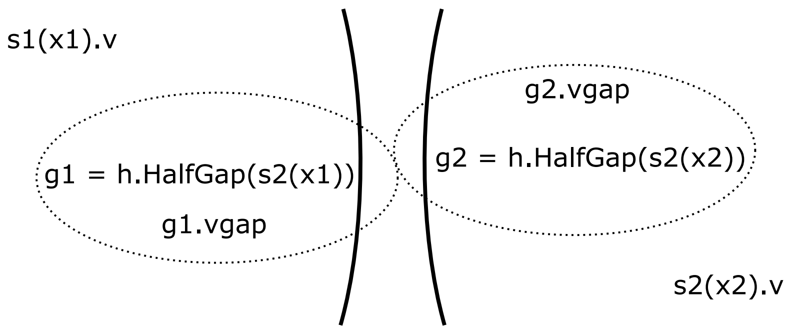
- Each subtree can have at most two split nodes.
- Does not support variable step, linear mechanisms, extracellular, or reaction-diffusion.
- `h.distance` cannot compute path distances that cross a split node.

Tip: For load balancing, it is sometimes convenient to split cells into more pieces than processes.

For an example, see the file `multisplit_distrib.py` at http://modeldb.yale.edu/151681

Gap Junctions

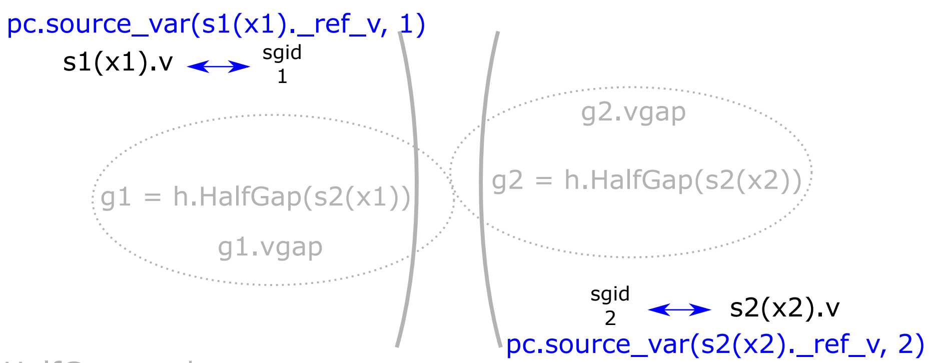
Continuous voltage exchange



HalfGap.mod

```
NEURON  {                        ASSIGNED {
  POINT_PROCESS HalfGap            v (millivolt)
  ELECTRODE_CURRENT i              vgap (millivolt)
  RANGE r, i, vgap                 i (nanoamp)
}                                }
PARAMETER { r = 1e9 (megohm) }   CURRENT { i = (vgap - v) / r }
```

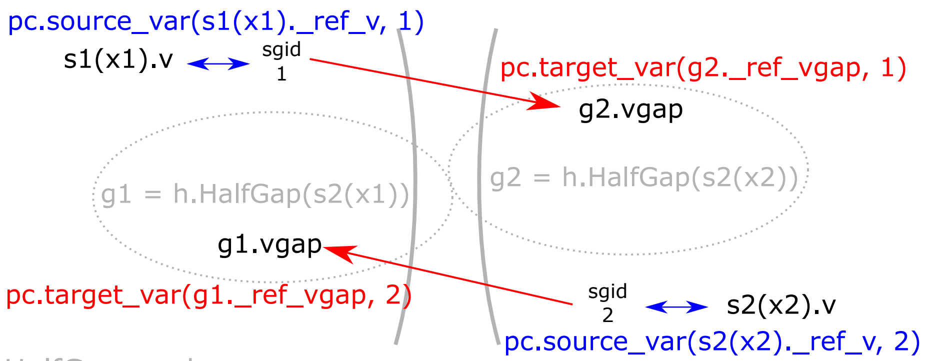
pc.source_var to declare source sgid



HalfGap.mod

```
NEURON  {                        ASSIGNED {
  POINT_PROCESS HalfGap            v (millivolt)
  ELECTRODE_CURRENT i              vgap (millivolt)
  RANGE r, i, vgap                 i (nanoamp)
}                                }
PARAMETER { r = 1e9 (megohm) }   CURRENT { i = (vgap - v) / r }
```

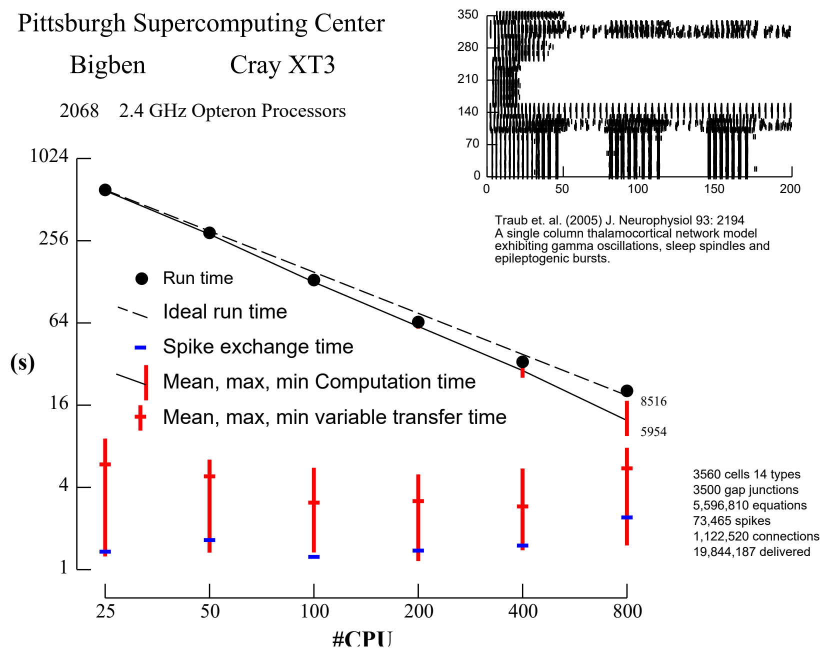
pc.target_var to declare target connection


pc.source_var(s1(x1)._ref_v, 1)
s1(x1).v
sgid
1
pc.target_var(g2._ref_vgap, 1)
g2.vgap
g1 = h.HalfGap(s2(x1))
g2 = h.HalfGap(s2(x2))
g1.vgap
pc.target_var(g1._ref_vgap, 2)
sgid
2
s2(x2).v
pc.source_var(s2(x2)._ref_v, 2)

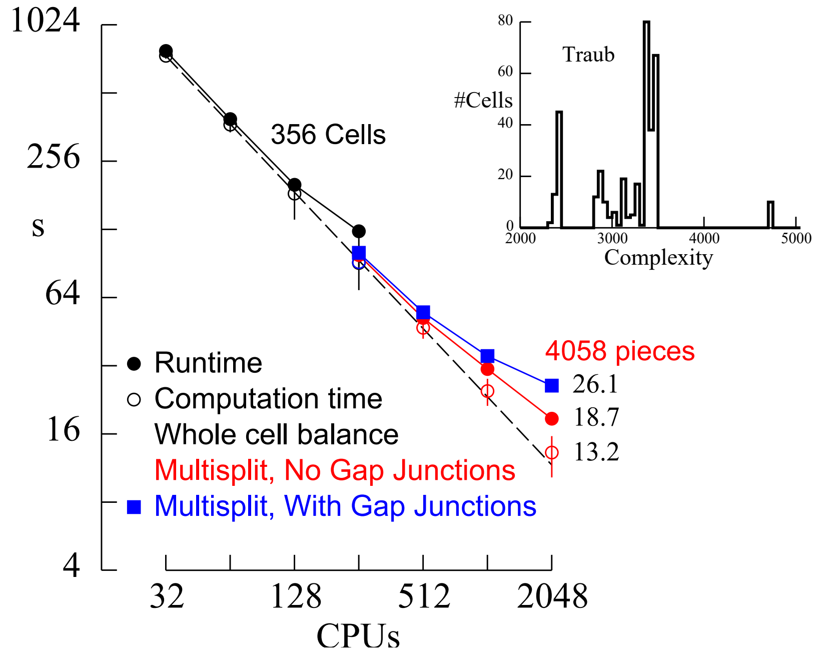
HalfGap.mod

```
NEURON  {                           ASSIGNED {
  POINT_PROCESS HalfGap               v (millivolt)
  ELECTRODE_CURRENT i                 vgap (millivolt)
  RANGE r, i, vgap                    i (nanoamp)
}                                   }
PARAMETER { r = 1e9 (megohm) }      CURRENT { i = (vgap - v) / r }
```

Performance: Traub model



Performance: Traub model with multisplit



Don't reinvent the brain

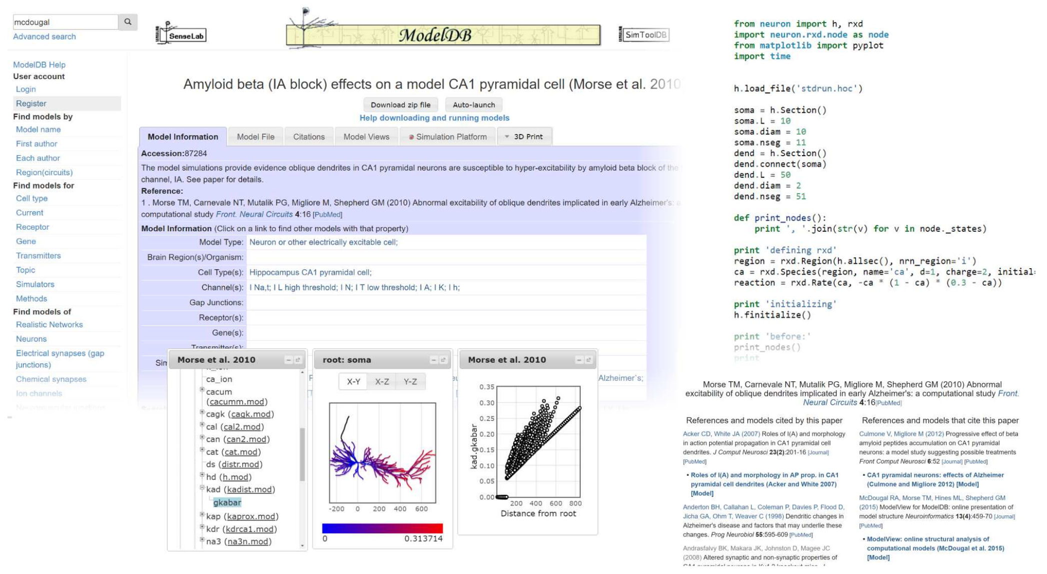
Using ModelDB and other archives for your research

Robert A. McDougal

Yale School of Medicine

11 November 2016

What is ModelDB?



Amyloid beta (IA block) effects on a model CA1 pyramidal cell (Morse et al. 2010

Accession:87284

The model simulations provide evidence oblique dendrites in CA1 pyramidal neurons are susceptible to hyper-excitability by amyloid beta block of the channel, IA. See paper for details.

Reference:

1 . Morse TM, Carnevale NT, Mutalik PG, Migliore M, Shepherd GM (2010) Abnormal excitability of oblique dendrites implicated in early Alzheimer's: a computational study *Front. Neural Circuits* **4**:16 [PubMed]

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

Model Type:	Neuron or other electrically excitable cell;
Brain Region(s)/Organism:	
Cell Type(s):	Hippocampus CA1 pyramidal cell;
Channel(s):	I Na,t; I L high threshold; I N; I T low threshold; I A; I K; I h;
Gap Junctions:	
Receptor(s):	
Gene(s):	

```
from neuron import h, rxd
import neuron.rxd.node as node
from matplotlib import pyplot
import time


h.load_file('stdrun.hoc')

soma = h.Section()
soma.L = 10
soma.diam = 10
soma.nseg = 11
dend = h.Section()
dend.connect(soma)
dend.L = 50
dend.diam = 2
dend.nseg = 51

def print_nodes():
    print ', '.join(str(v) for v in node._states)

print 'defining rxd'
region = rxd.Region(h.allsec(), nrn_region='i')
ca = rxd.Species(region, name='ca', d=1, charge=2, initial
reaction = rxd.Rate(ca, -ca * (1 - ca) * (0.3 - ca))

print 'initializing'
h.finitialize()

print 'before:'
print_nodes()
print
```

Morse TM, Carnevale NT, Mutalik PG, Migliore M, Shepherd GM (2010) Abnormal excitability of oblique dendrites implicated in early Alzheimer's: a computational study *Front. Neural Circuits* **4**:16[PubMed]

References and models cited by this paper

Acker CD, White JA (2007) Roles of I(A) and morphology in action potential propagation in CA1 pyramidal cell dendrites. *J Comput Neurosci* **23(2)**:201-16 [Journal] [PubMed]

- **Roles of I(A) and morphology in AP prop. in CA1 pyramidal cell dendrites (Acker and White 2007) [Model]**

Anderton BH, Callahan L, Coleman P, Davies P, Flood D, Jicha GA, Ohm T, Weaver C (1998) Dendritic changes in Alzheimer's disease and factors that may underlie these changes. *Prog Neurobiol* **55**:595-609 [PubMed]

Andrasfalvy BK, Makara JK, Johnston D, Magee JC (2008) Altered synaptic and non-synaptic properties of

References and models that cite this paper

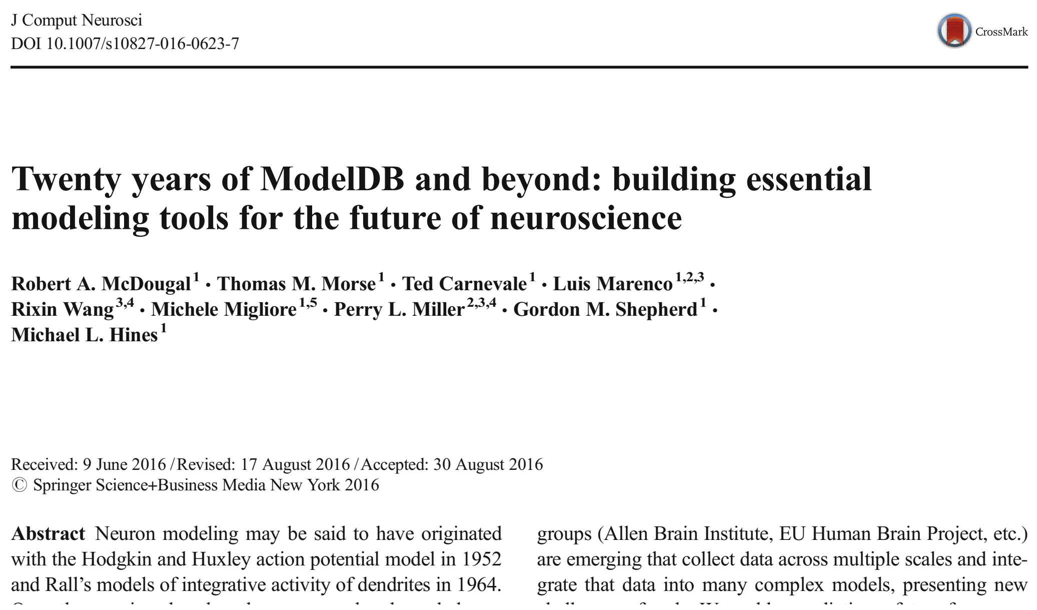
Culmone V, Migliore M (2012) Progressive effect of beta amyloid peptides accumulation on CA1 pyramidal neurons: a model study suggesting possible treatments *Front Comput Neurosci* **6**:52 [Journal] [PubMed]

- **CA1 pyramidal neurons: effects of Alzheimer (Culmone and Migliore 2012) [Model]**

McDougal RA, Morse TM, Hines ML, Shepherd GM (2015) ModelView for ModelDB: online presentation of model structure *Neuroinformatics* **13(4)**:459-70 [Journal] [PubMed]

- **ModelView: online structural analysis of computational models (McDougal et al. 2015) [Model]**

modeldb.yale.edu



J Comput Neurosci
DOI 10.1007/s10827-016-0623-7

CrossMark

Twenty years of ModelDB and beyond: building essential modeling tools for the future of neuroscience

Robert A. McDougal[1] · Thomas M. Morse[1] · Ted Carnevale[1] · Luis Marenco[1,2,3] · Rixin Wang[3,4] · Michele Migliore[1,5] · Perry L. Miller[2,3,4] · Gordon M. Shepherd[1] · Michael L. Hines[1]

Received: 9 June 2016 / Revised: 17 August 2016 / Accepted: 30 August 2016
© Springer Science+Business Media New York 2016

Abstract Neuron modeling may be said to have originated with the Hodgkin and Huxley action potential model in 1952 and Rall's models of integrative activity of dendrites in 1964. groups (Allen Brain Institute, EU Human Brain Project, etc.) are emerging that collect data across multiple scales and integrate that data into many complex models, presenting new

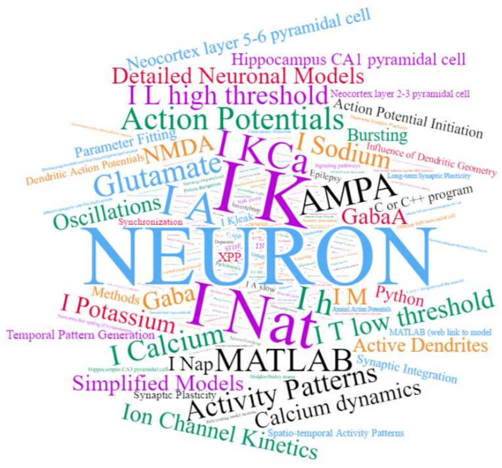
What is in ModelDB?

Models for:

- 178 cell types
- 16+ species
- 48 ion channels, pumps, etc
- 139 topics (Alzheimer's, STDP, etc)
- 25+ mammalian brain regions

1134 published models from 76 simulators

- 544 NEURON models
- 318 "realistic" networks
- 45 connectionist networks



Numbers are as of October 24, 2016

Why use ModelDB?

On reproducibility

"Non-reproducible single occurrences are of no significance to science."

– Karl Popper in *The logic of scientific discovery*, 1959.

What is needed for a model to be reproducible?

Model

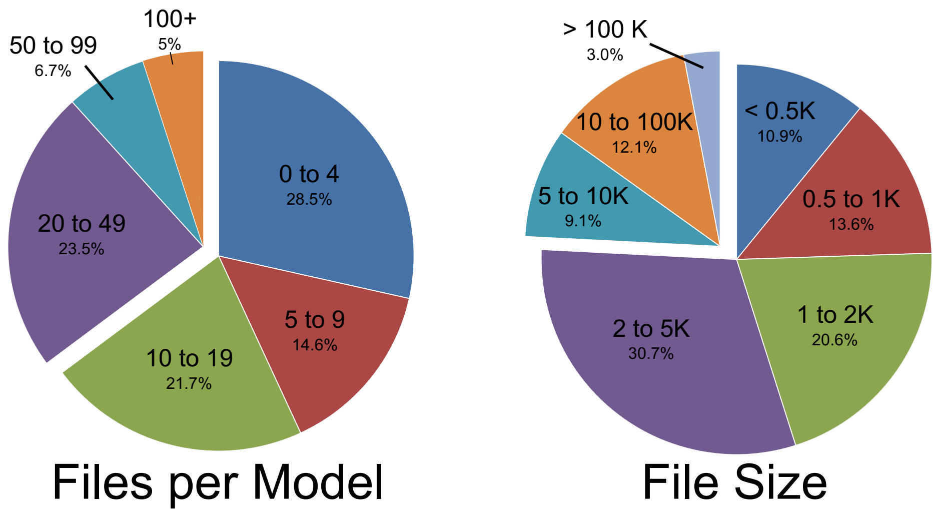
- an approximation of the system of interest
  e.g. a model organism or a complete statement of the properties of the model in mathematical or computable form

Experimental protocol

- what was done with the model to produce the data

Science builds upon previous work; in order to do that, the previous work needs to be reproducible.

Models are complicated



- **38.5%** of ModelDB models have **over 20 files**; **24.2%** of files are **over 5K**.
- It is often hard to fully describe this complexity in a paper.
- Any bugs, typos, errors, or omissions might completely change the dynamics.

Distributions from ModelDB, Fall 2013. A model was counted as having 0 files if it was not hosted on ModelDB.

Model sharing helps, but only reuse what you understand

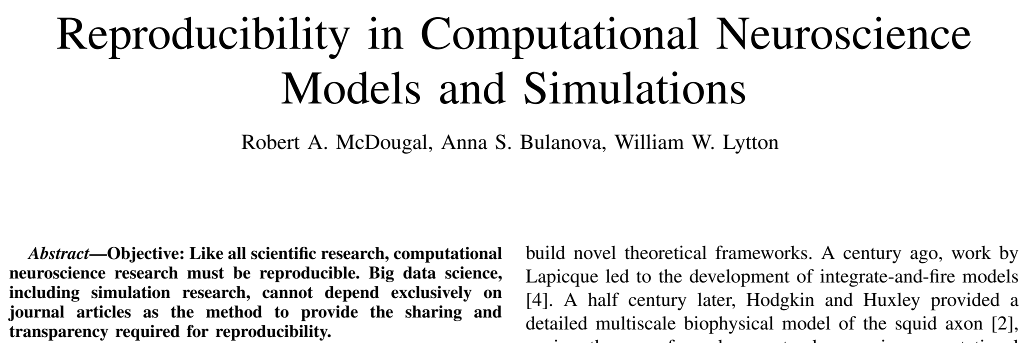
The easiest way to replicate someone else's results – a first step toward building on them – is to get their model code from a repository such as ModelDB.

But beware:

- They may be solving a different problem than you (with respect to species, temperature, age, etc).
- Their code may have bugs.

To reduce the risk of problems:

- **Read** the associated paper.
- **Compare** the model and results to other similar models.
- **Examine** the model with ModelView and/or psection.
- **Test** ion channels individually.
- **Collaborate** with an experimentalist.



Reproducibility in Computational Neuroscience Models and Simulations

Robert A. McDougal, Anna S. Bulanova, William W. Lytton

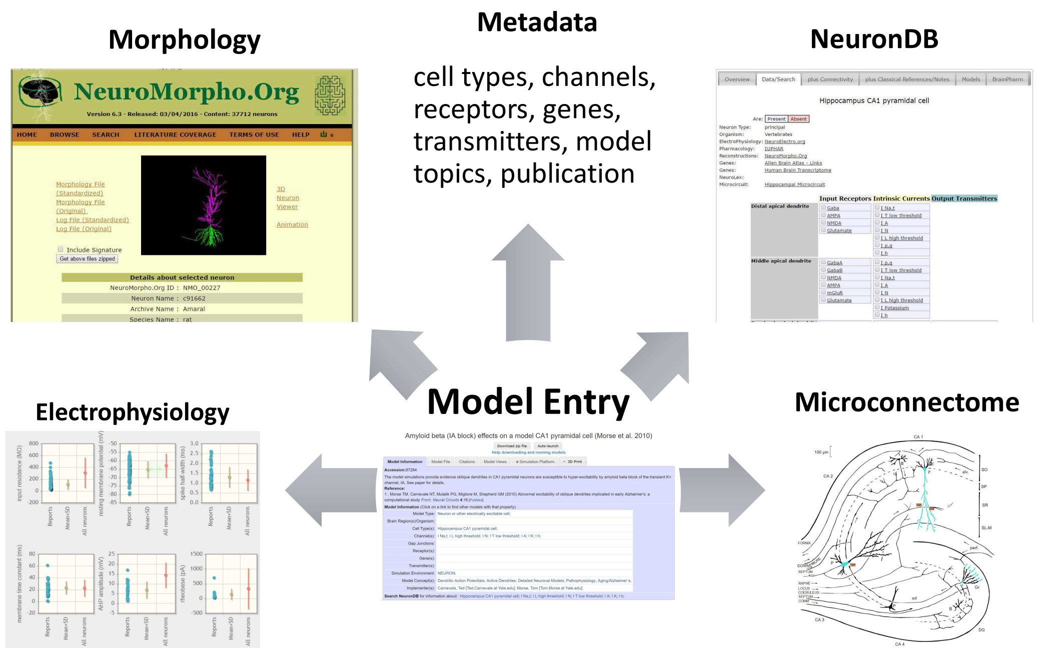
Abstract—**Objective: Like all scientific research, computational neuroscience research must be reproducible. Big data science, including simulation research, cannot depend exclusively on journal articles as the method to provide the sharing and transparency required for reproducibility.**

build novel theoretical frameworks. A century ago, work by Lapicque led to the development of integrate-and-fire models [4]. A half century later, Hodgkin and Huxley provided a detailed multiscale biophysical model of the squid axon [2],

- Simulators (NEURON, MCell, XPPAUT, NEST, etc)
- Multi-simulator interoperability (NeuroML, SWC, PyNN, NeuroConstruct, etc)
- Shared resources (Neuroscience Gateway, Simulation Platform)
- Sharing resources (ModelDB, OpenSourceBrain, NeuroMorpho.Org, etc)
- More: NSDF, NeuroLex, NIF, MIASE, licensing, etc

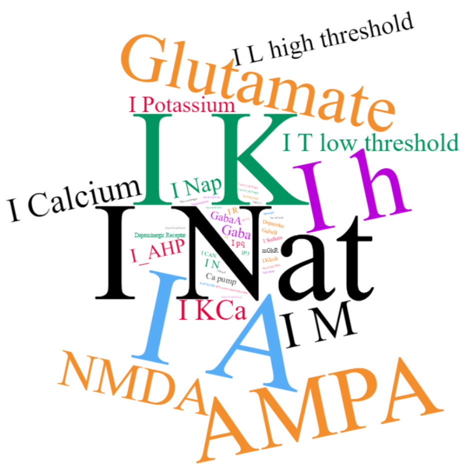
McDougal et al (2016) IEEE TBME 63(10):2021-2035; doi:10.1109/TBME.2016.2539602

Neurobiological context


Morphology
Metadata
cell types, channels, receptors, genes, transmitters, model topics, publication
NeuronDB
NeuroMorpho.Org
Hippocampus CA1 pyramidal cell
Electrophysiology
Model Entry
Amyloid beta (IA block) effects on a model CA1 pyramidal cell (Morse et al. 2010)
Microconnectome

ModelDB is a place to see what has been modeled in a cell type.

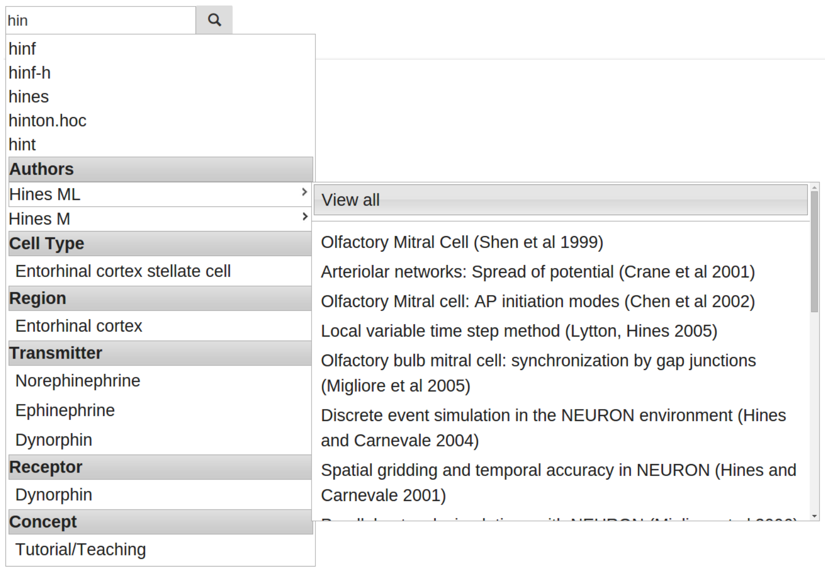
Metadata associated with CA1 Pyramidal Cell Models ($n = 71$)



Not only can you get code, but by comparing models, you can see what mechanisms are considered critical by the community.

How to use ModelDB

Finding models


hin

hinf
hinf-h
hines
hinton.hoc
hint

Authors
Hines ML
Hines M

Cell Type
Entorhinal cortex stellate cell

Region
Entorhinal cortex

Transmitter
Norephinephrine
Ephinephrine
Dynorphin

Receptor
Dynorphin

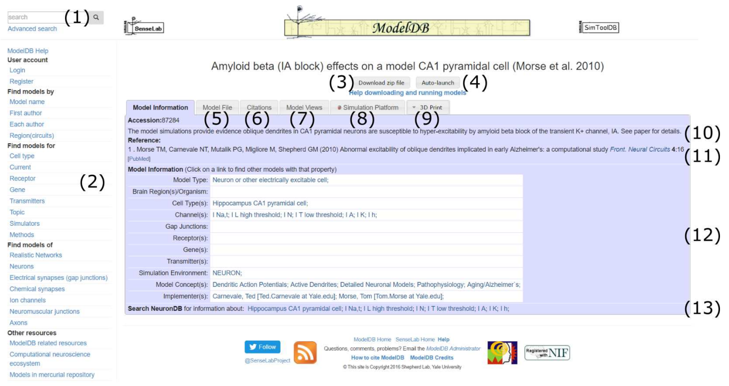
Concept
Tutorial/Teaching

View all

Olfactory Mitral Cell (Shen et al 1999)
Arteriolar networks: Spread of potential (Crane et al 2001)
Olfactory Mitral cell: AP initiation modes (Chen et al 2002)
Local variable time step method (Lytton, Hines 2005)
Olfactory bulb mitral cell: synchronization by gap junctions (Migliore et al 2005)
Discrete event simulation in the NEURON environment (Hines and Carnevale 2004)
Spatial gridding and temporal accuracy in NEURON (Hines and Carnevale 2001)

- **Search box** on the top-left of every page.
- Do **full text** or **attribute** searches.
- Word completions (based on ModelDB entries not English) and attribute results **updated as you type**.
- **Advanced search** and **browsing** are also available.

ShowModel features



Amyloid beta (IA block) effects on a model CA1 pyramidal cell (Morse et al. 2010)

Accession:87284

The model simulations provide evidence oblique dendrites in CA1 pyramidal neurons are susceptible to hyper-excitability by amyloid beta block of the transient K+ channel, IA. See paper for details.

Reference:

1 . Morse TM, Carnevale NT, Mutalik PG, Migliore M, Shepherd GM (2010) Abnormal excitability of oblique dendrites implicated in early Alzheimer's: a computational study *Front. Neural Circuits* **4**:16 [PubMed]

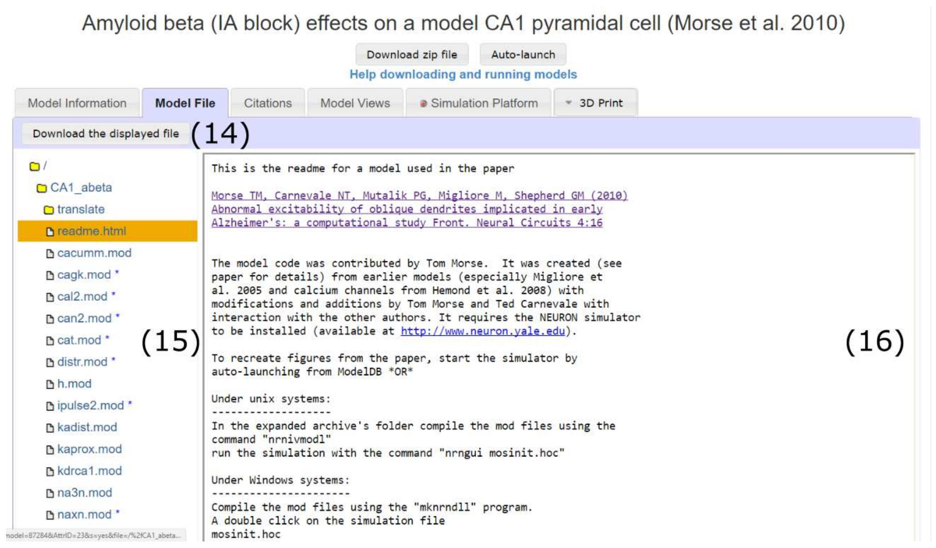
Model Information (Click on a link to find other models with that property)

Model Type:	Neuron or other electrically excitable cell;
Brain Region(s)/Organism:	
Cell Type(s):	Hippocampus CA1 pyramidal cell;
Channel(s):	I Na,t; I L high threshold; I N; I T low threshold; I A; I K; I h;
Gap Junctions:	
Receptor(s):	
Gene(s):	
Transmitter(s):	
Simulation Environment:	NEURON;
Model Concept(s):	Dendritic Action Potentials; Active Dendrites; Detailed Neuronal Models; Pathophysiology; Aging/Alzheimer`s;
Implementer(s):	Carnevale, Ted [Ted.Carnevale at Yale.edu]; Morse, Tom [Tom.Morse at Yale.edu];

Search NeuronDB for information about: Hippocampus CA1 pyramidal cell; I Na,t; I L high threshold; I N; I T low threshold; I A; I K; I h;

(1) Search models. **(2)** Browse models. **(3)** Link to download the entire model code. **(4)** Auto-launch a NEURON simulation (requires browser configuration). **(5)** View model files. **(6)** Find models and papers cited by this model's paper, or that cite this model. **(7)** ModelView: visualize model structure. **(8)** Simulation platform (5 minutes of remote desktop access to experiment with the model). **(9)** 3D printable versions of cells from the model (in 3DModelDB). **(10)** Description of model. **(11)** Paper(s) describing or using model. **(12)** Searchable metadata. **(13)** Links to NeuronDB (channel distributions etc within cell types).

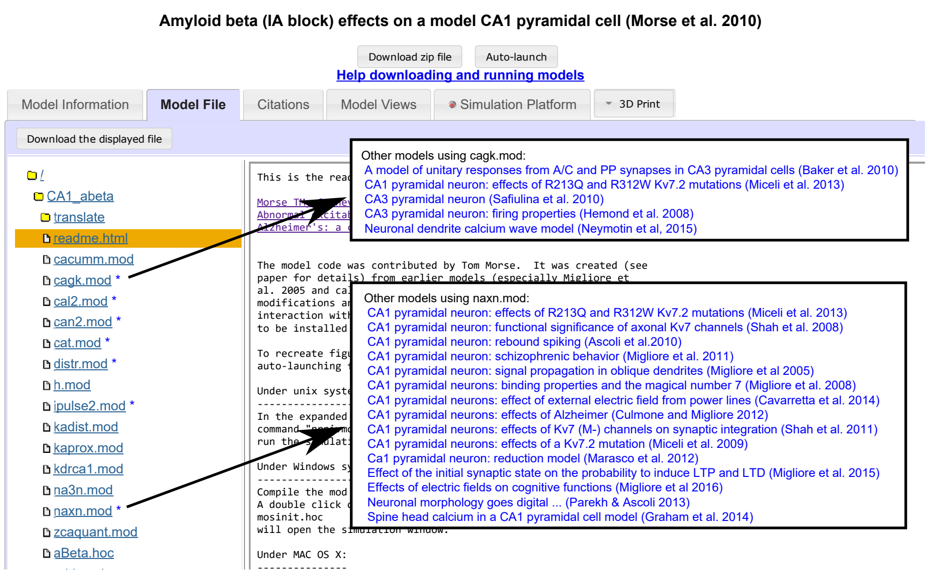
ShowModel features



```
This is the readme for a model used in the paper

Morse TM, Carnevale NT, Mutalik PG, Migliore M, Shepherd GM (2010)
Abnormal excitability of oblique dendrites implicated in early
Alzheimer's: a computational study Front. Neural Circuits 4:16


The model code was contributed by Tom Morse.  It was created (see
paper for details) from earlier models (especially Migliore et
al. 2005 and calcium channels from Hemond et al. 2008) with
modifications and additions by Tom Morse and Ted Carnevale with
interaction with the other authors. It requires the NEURON simulator
to be installed (available at http://www.neuron.yale.edu).

To recreate figures from the paper, start the simulator by
auto-launching from ModelDB *OR*

Under unix systems:
-------------------
In the expanded archive's folder compile the mod files using the
command "nrnivmodl"
run the simulation with the command "nrngui mosinit.hoc"

Under Windows systems:
----------------------
Compile the mod files using the "mknrndll" program.
A double click on the simulation file
mosinit.hoc
```

(14) Download the currently selected file. **(15)** Directory browser, showing model files. **(16)** View pane for the currently selected file.

Identifying existing reuse


Amyloid beta (IA block) effects on a model CA1 pyramidal cell (Morse et al. 2010)

Download zip file | Auto-launch

Help downloading and running models

Model Information | Model File | Citations | Model Views | Simulation Platform | 3D Print

Download the displayed file

- /
  - CA1_abeta
    - translate
    - readme.html
    - cacumm.mod
    - cagk.mod *
    - cal2.mod *
    - can2.mod *
    - cat.mod *
    - distr.mod *
    - h.mod
    - ipulse2.mod *
    - kadist.mod
    - kaprox.mod
    - kdrca1.mod
    - na3n.mod
    - naxn.mod *
    - zcaquant.mod
    - aBeta.hoc

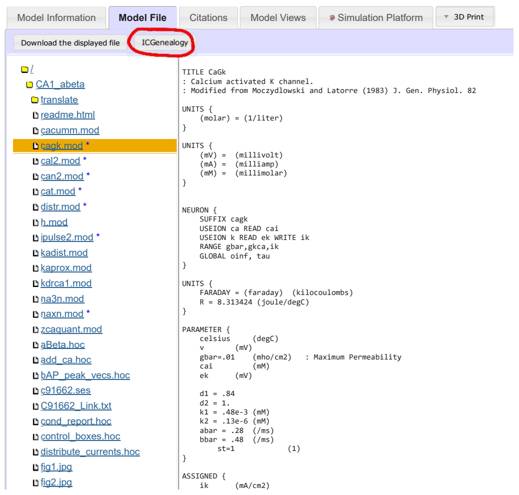
```
This is the read

Morse TM
Abnormal
Alzheimer's: a


The model code was contributed by Tom Morse.  It was created (see
paper for details) from earlier models (especially Migliore et
al. 2005 and ca
modifications ar
interaction with
to be installed

To recreate figu
auto-launching

Under unix syste
----------------
In the expanded
command
run the

Under Windows sy
----------------
Compile the mod
A double click
mosinit.hoc
will open the simulation window.

Under MAC OS X:
---------------
```

Other models using cagk.mod:
- A model of unitary responses from A/C and PP synapses in CA3 pyramidal cells (Baker et al. 2010)
- CA1 pyramidal neuron: effects of R213Q and R312W Kv7.2 mutations (Miceli et al. 2013)
- CA3 pyramidal neuron (Safiulina et al. 2010)
- CA3 pyramidal neuron: firing properties (Hemond et al. 2008)
- Neuronal dendrite calcium wave model (Neymotin et al, 2015)

Other models using naxn.mod:
- CA1 pyramidal neuron: effects of R213Q and R312W Kv7.2 mutations (Miceli et al. 2013)
- CA1 pyramidal neuron: functional significance of axonal Kv7 channels (Shah et al. 2008)
- CA1 pyramidal neuron: rebound spiking (Ascoli et al.2010)
- CA1 pyramidal neuron: schizophrenic behavior (Migliore et al. 2011)
- CA1 pyramidal neuron: signal propagation in oblique dendrites (Migliore et al 2005)
- CA1 pyramidal neurons: binding properties and the magical number 7 (Migliore et al. 2008)
- CA1 pyramidal neurons: effect of external electric field from power lines (Cavarretta et al. 2014)
- CA1 pyramidal neurons: effects of Alzheimer (Culmone and Migliore 2012)
- CA1 pyramidal neurons: effects of Kv7 (M-) channels on synaptic integration (Shah et al. 2011)
- CA1 pyramidal neurons: effects of a Kv7.2 mutation (Miceli et al. 2009)
- Ca1 pyramidal neuron: reduction model (Marasco et al. 2012)
- Effect of the initial synaptic state on the probability to induce LTP and LTD (Migliore et al. 2015)
- Effects of electric fields on cognitive functions (Migliore et al 2016)
- Neuronal morphology goes digital ... (Parekh & Ascoli 2013)
- Spine head calcium in a CA1 pyramidal cell model (Graham et al. 2014)

Asterisks in the file browser indicate that the file is reused in other models; click the asterisk to see a list of the other models.

ICGenealogy: ion channel metadata


Model Information | Model File | Citations | Model Views | Simulation Platform | 3D Print

Download the displayed file | ICGenealogy

- /
  - CA1_abeta
    - translate
    - readme.html
    - cacumm.mod
    - cagk.mod *
    - cal2.mod *
    - can2.mod *
    - cat.mod *
    - distr.mod *
    - h.mod
    - ipulse2.mod *
    - kadist.mod
    - kaprox.mod
    - kdrca1.mod
    - na3n.mod
    - naxn.mod *
    - zcaquant.mod
    - aBeta.hoc
    - add_ca.hoc
    - bAP_peak_vecs.hoc
    - c91662.ses
    - C91662_Link.txt
    - cond_report.hoc
    - control_boxes.hoc
    - distribute_currents.hoc
    - fig1.jpg
    - fig2.jpg

```
TITLE CaGk
: Calcium activated K channel.
: Modified from Moczydlowski and Latorre (1983) J. Gen. Physiol. 82

UNITS {
    (molar) = (1/liter)
}

UNITS {
    (mV) =  (millivolt)
    (mA) =  (milliamp)
    (mM) =  (millimolar)
}


NEURON {
    SUFFIX cagk
    USEION ca READ cai
    USEION k READ ek WRITE ik
    RANGE gbar,gkca,ik
    GLOBAL oinf, tau
}

UNITS {
    FARADAY = (faraday)  (kilocoulombs)
    R = 8.313424 (joule/degC)
}

PARAMETER {
    celsius     (degC)
    v       (mV)
    gbar=.01    (mho/cm2)   : Maximum Permeability
    cai         (mM)
    ek      (mV)

    d1 = .84
    d2 = 1.
    k1 = .48e-3 (mM)
    k2 = .13e-6 (mM)
    abar = .28  (/ms)
    bbar = .48  (/ms)
        st=1            (1)
}

ASSIGNED {
    ik      (mA/cm2)
```

General data

- **ICG id:** 2464
- **ModelDB id:** 87284
- **Reference:** *Morse TM, Carnevale NT, Mutalik PG, Migliore M, Shepherd GM (2010)*: Abnormal Excitability of Oblique Dendrites Implicated in Early Alzheimer's: A Computational Study.

Metadata classes

- **Animal Model:** rat
- **Brain Area:** hippocampus, CA1
- **Classes:** KCa
- **Ion Type:** K
- **Neuron Region:** unspecified
- **Neuron Type:** pyramidal cell
- **Runtime Q:** Q4 (slow)
- **Subtype:** not specified

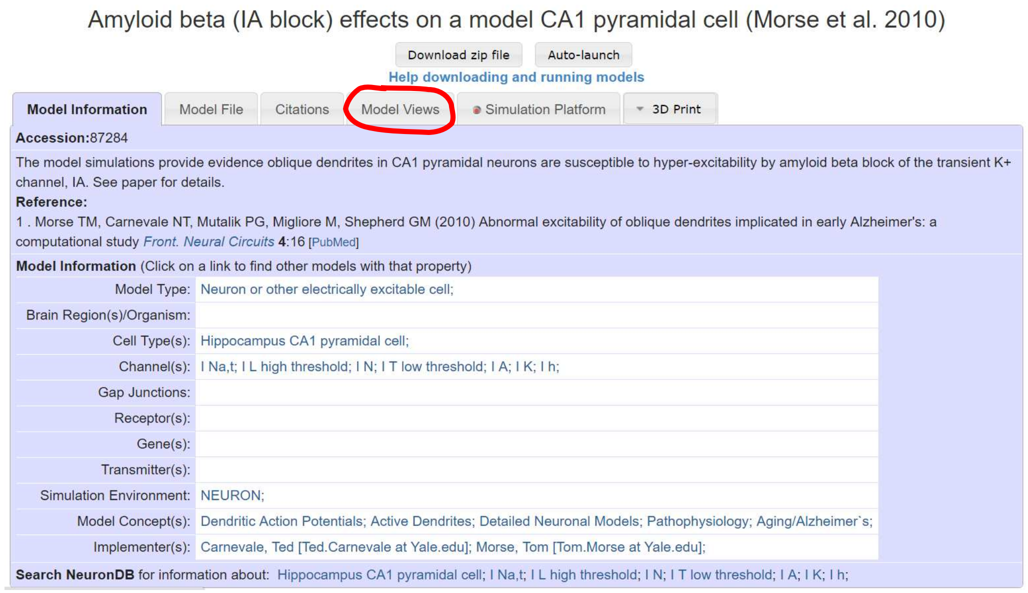
Metadata generic

- **Age:** 7-14 weeks old.
- **Comments:** Calcium activated k channel, modified from moczydlowski and latorre (1983). From hemond et al. (2008), model no. 101629, with no changes (identical mod file). Animal model taken from chen (2005) which is used to constrain model. Channel kinetics from previous study on hippocampal pyramidal neuron (hemond et al. 2008)
- **Runtime:** 76.722

When viewing most mod files describing an ion channel, an ICGenealogy button appears. Clicking this button loads the corresponding page of the ICGenealogy database which shows curated information about the channel model (how it was derived, information about the underlying data, etc) and response curves.

Podlaski, Seeholzer, Vogels

ModelView



Amyloid beta (IA block) effects on a model CA1 pyramidal cell (Morse et al. 2010)

Accession:87284

The model simulations provide evidence oblique dendrites in CA1 pyramidal neurons are susceptible to hyper-excitability by amyloid beta block of the transient K+ channel, IA. See paper for details.

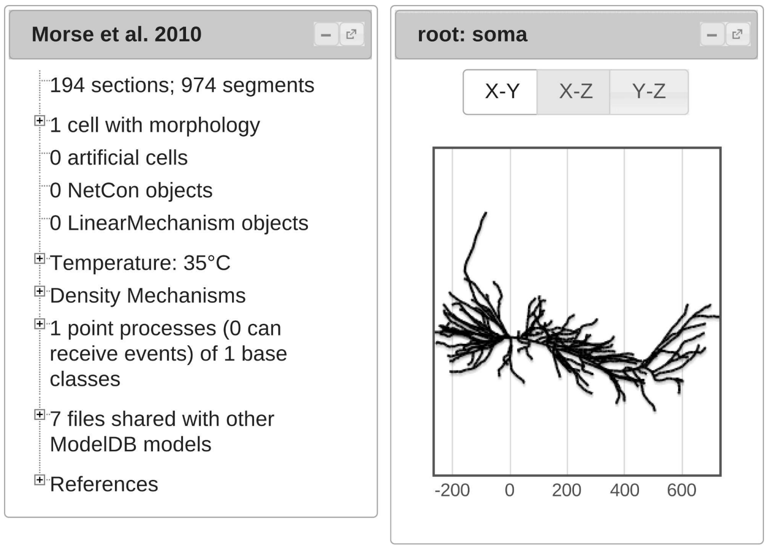
Reference:

1 . Morse TM, Carnevale NT, Mutalik PG, Migliore M, Shepherd GM (2010) Abnormal excitability of oblique dendrites implicated in early Alzheimer's: a computational study *Front. Neural Circuits* **4**:16 [PubMed]

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

Model Type:	Neuron or other electrically excitable cell;
Brain Region(s)/Organism:	
Cell Type(s):	Hippocampus CA1 pyramidal cell;
Channel(s):	I Na,t; I L high threshold; I N; I T low threshold; I A; I K; I h;
Gap Junctions:	
Receptor(s):	
Gene(s):	
Transmitter(s):	
Simulation Environment:	NEURON;
Model Concept(s):	Dendritic Action Potentials; Active Dendrites; Detailed Neuronal Models; Pathophysiology; Aging/Alzheimer`s;
Implementer(s):	Carnevale, Ted [Ted.Carnevale at Yale.edu]; Morse, Tom [Tom.Morse at Yale.edu];

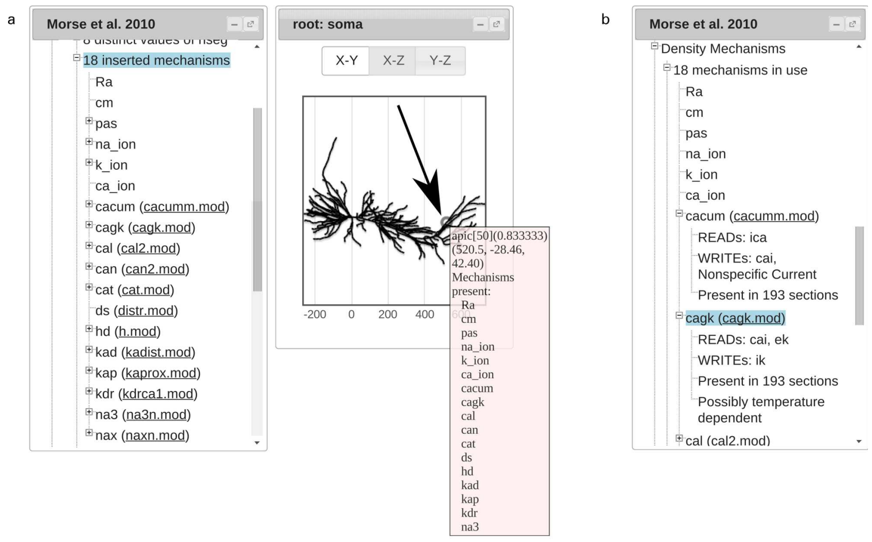
Search NeuronDB for information about: Hippocampus CA1 pyramidal cell; I Na,t; I L high threshold; I N; I T low threshold; I A; I K; I h;



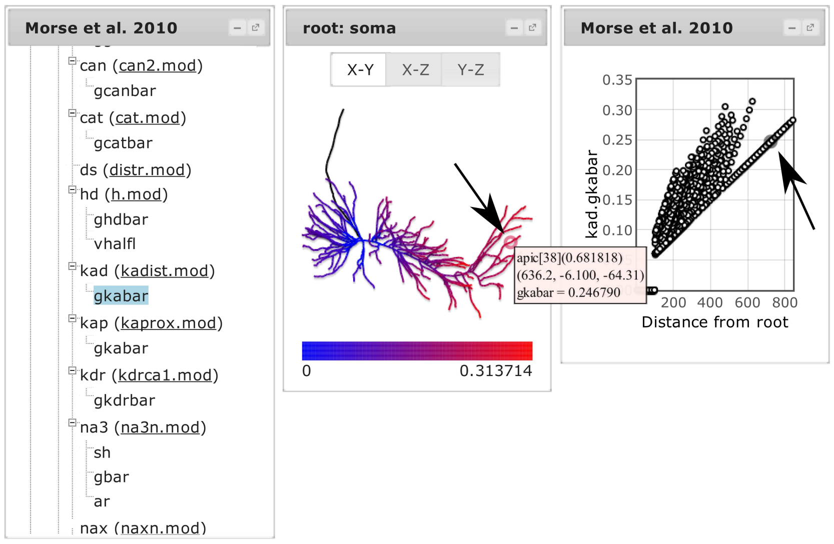
Morse et al. 2010

- 194 sections; 974 segments
- ⊞ 1 cell with morphology
- 0 artificial cells
- 0 NetCon objects
- 0 LinearMechanism objects
- ⊞ Temperature: 35°C
- ⊞ Density Mechanisms
- ⊞ 1 point processes (0 can receive events) of 1 base classes
- ⊞ 7 files shared with other ModelDB models
- ⊞ References

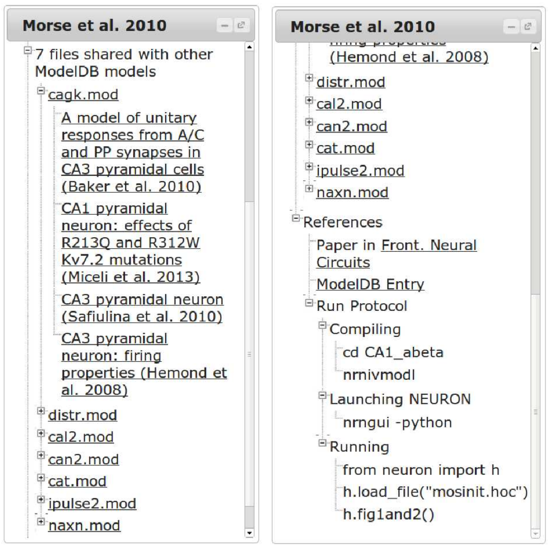
McDougal et al, *Neuroinformatics* 2015



McDougal et al, *Neuroinformatics* 2015



McDougal et al, *Neuroinformatics* 2015



Morse et al. 2010

7 files shared with other ModelDB models
- cagk.mod
  - A model of unitary responses from A/C and PP synapses in CA3 pyramidal cells (Baker et al. 2010)
  - CA1 pyramidal neuron: effects of R213Q and R312W Kv7.2 mutations (Miceli et al. 2013)
  - CA3 pyramidal neuron (Safiulina et al. 2010)
  - CA3 pyramidal neuron: firing properties (Hemond et al. 2008)
- distr.mod
- cal2.mod
- can2.mod
- cat.mod
- ipulse2.mod
- naxn.mod

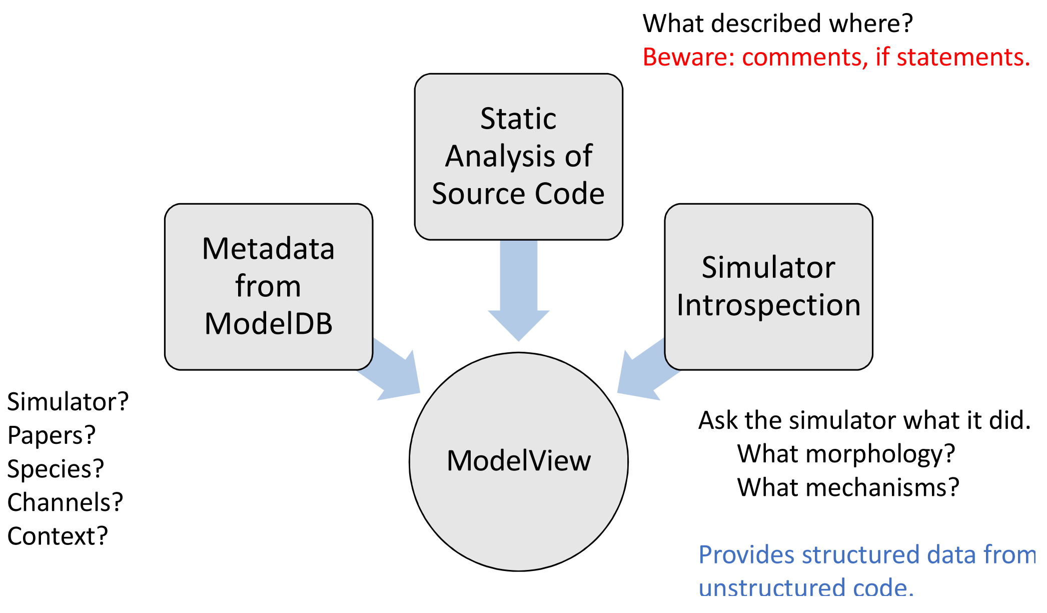
Morse et al. 2010

(Hemond et al. 2008)
- distr.mod
- cal2.mod
- can2.mod
- cat.mod
- ipulse2.mod
- naxn.mod

References
- Paper in Front. Neural Circuits
- ModelDB Entry

Run Protocol
- Compiling
  - cd CA1_abeta
  - nrnivmodl
- Launching NEURON
  - nrngui -python
- Running
  - from neuron import h
  - h.load_file("mosinit.hoc")
  - h.fig1and2()

McDougal et al, *Neuroinformatics* 2015

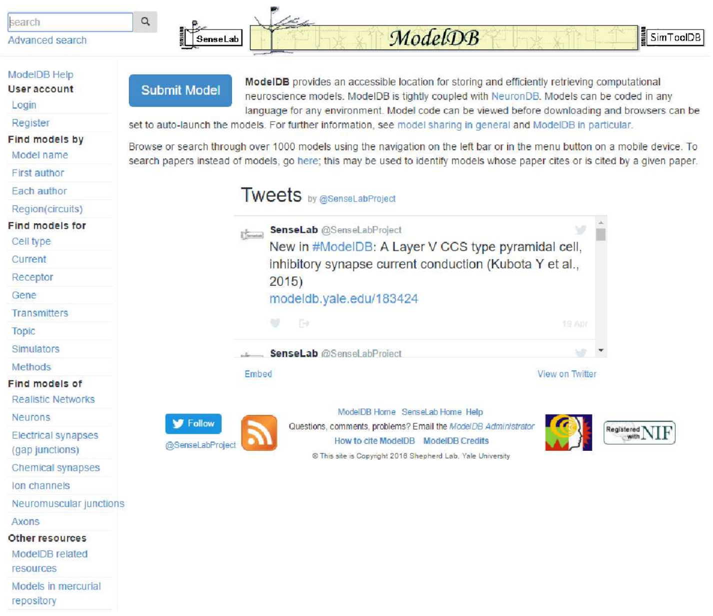


How do people use ModelDB?

- Find a model described in a paper, download it, and experiment to understand the model's predictions.
- Find a model described in a paper. Use ModelView to understand the model's structure.
- Locate models and modeling papers on a given topic.
- Locate model components (e.g. L-type calcium channel) for potential reuse.
- Search for simulator keywords (e.g. FInitializeHandler) to find examples of how to use them.

You can help by sharing your model code on ModelDB after publication.

Sharing your models



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

Browse or search through over 1000 models using the navigation on the left bar or in the menu button on a mobile device. To search papers instead of models, go here; this may be used to identify models whose paper cites or is cited by a given paper.

Tweets by @SenseLabProject

SenseLab @SenseLabProject
New in #ModelDB: A Layer V CCS type pyramidal cell, inhibitory synapse current conduction (Kubota Y et al., 2015)
modeldb.yale.edu/183424
19 Apr

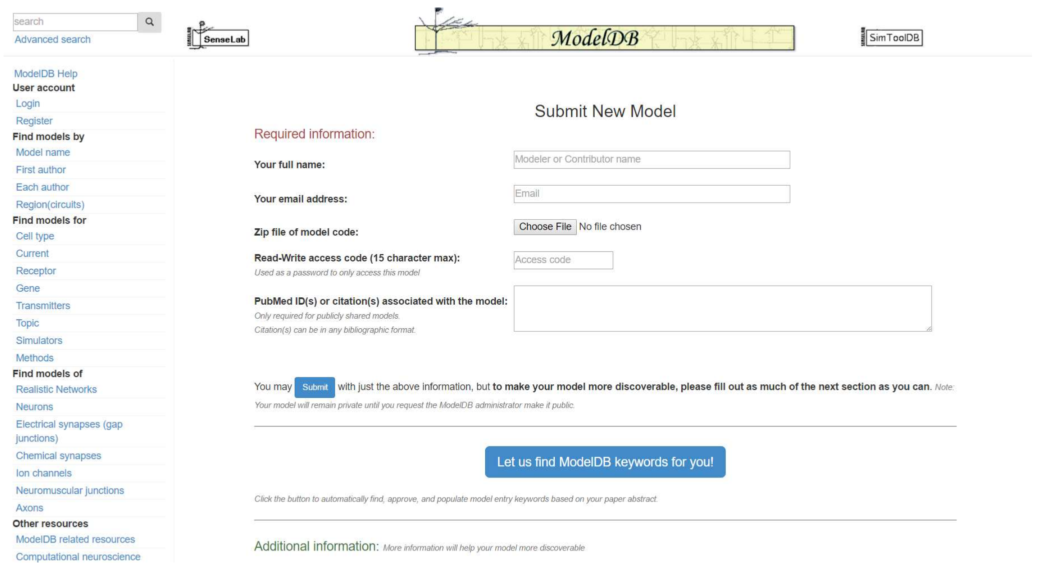
SenseLab @SenseLabProject

Questions, comments, problems? Email the *ModelDB Administrator*

© This site is Copyright 2016 Shepherd Lab, Yale University

McDougal, Dalal, Shepherd in preparation

Sharing your models



Submit New Model

Required information:

Your full name: Modeler or Contributor name

Your email address: Email

Zip file of model code: Choose File No file chosen

Read-Write access code (15 character max): Access code
Used as a password to only access this model

PubMed ID(s) or citation(s) associated with the model:
Only required for publicly shared models
Citation(s) can be in any bibliographic format.

You may Submit with just the above information, but **to make your model more discoverable, please fill out as much of the next section as you can.** *Note: Your model will remain private until you request the ModelDB administrator make it public.*

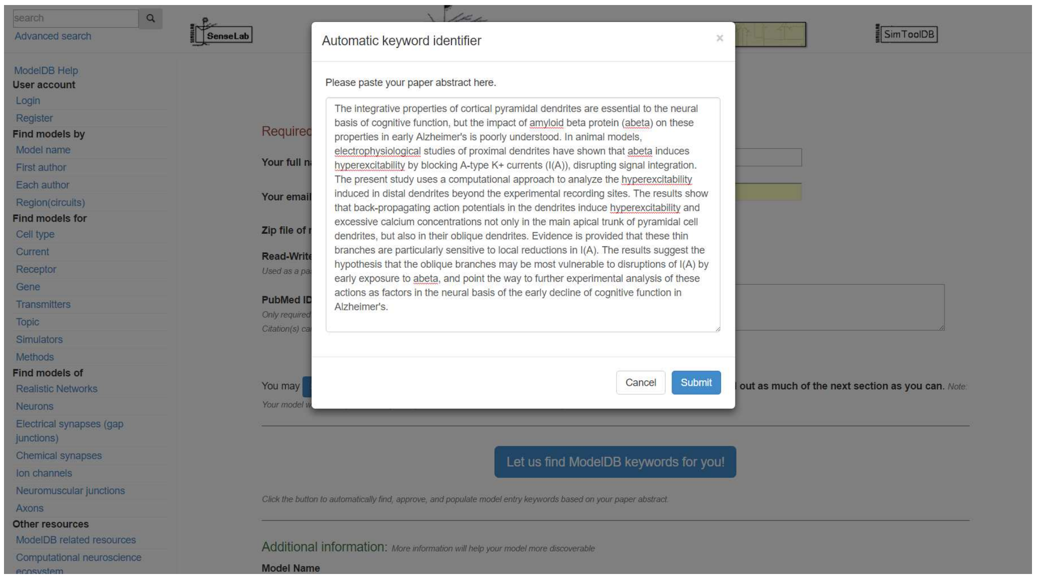
Let us find ModelDB keywords for you!

Click the button to automatically find, approve, and populate model entry keywords based on your paper abstract.

Additional information: *More information will help your model more discoverable*

McDougal, Dalal, Shepherd in preparation

Sharing your models



Automatic keyword identifier

Please paste your paper abstract here.

The integrative properties of cortical pyramidal dendrites are essential to the neural basis of cognitive function, but the impact of amyloid beta protein (abeta) on these properties in early Alzheimer's is poorly understood. In animal models, electrophysiological studies of proximal dendrites have shown that abeta induces hyperexcitability by blocking A-type K+ currents (I(A)), disrupting signal integration. The present study uses a computational approach to analyze the hyperexcitability induced in distal dendrites beyond the experimental recording sites. The results show that back-propagating action potentials in the dendrites induce hyperexcitability and excessive calcium concentrations not only in the main apical trunk of pyramidal cell dendrites, but also in their oblique dendrites. Evidence is provided that these thin branches are particularly sensitive to local reductions in I(A). The results suggest the hypothesis that the oblique branches may be most vulnerable to disruptions of I(A) by early exposure to abeta, and point the way to further experimental analysis of these actions as factors in the neural basis of the early decline of cognitive function in Alzheimer's.

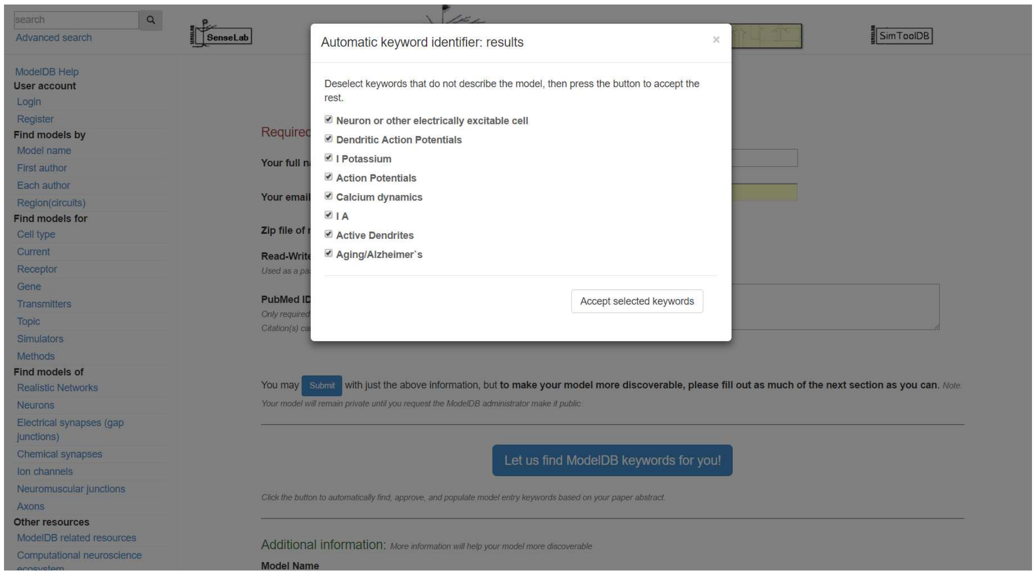
Click the button to automatically find, approve, and populate model entry keywords based on your paper abstract.

Additional information: *More information will help your model more discoverable*

Model Name

McDougal, Dalal, Shepherd in preparation; abstract from Morse et al, 2010.

Sharing your models



Automatic keyword identifier: results

Deselect keywords that do not describe the model, then press the button to accept the rest.

- ✓ Neuron or other electrically excitable cell
- ✓ Dendritic Action Potentials
- ✓ I Potassium
- ✓ Action Potentials
- ✓ Calcium dynamics
- ✓ I A
- ✓ Active Dendrites
- ✓ Aging/Alzheimer`s

Accept selected keywords

Required

Your full n

Your email

Zip file of

Read-Write

Used as a pa

PubMed ID

Only required

Citation(s) ca

You may Submit with just the above information, but **to make your model more discoverable, please fill out as much of the next section as you can.** Note: Your model will remain private until you request the ModelDB administrator make it public.

Let us find ModelDB keywords for you!

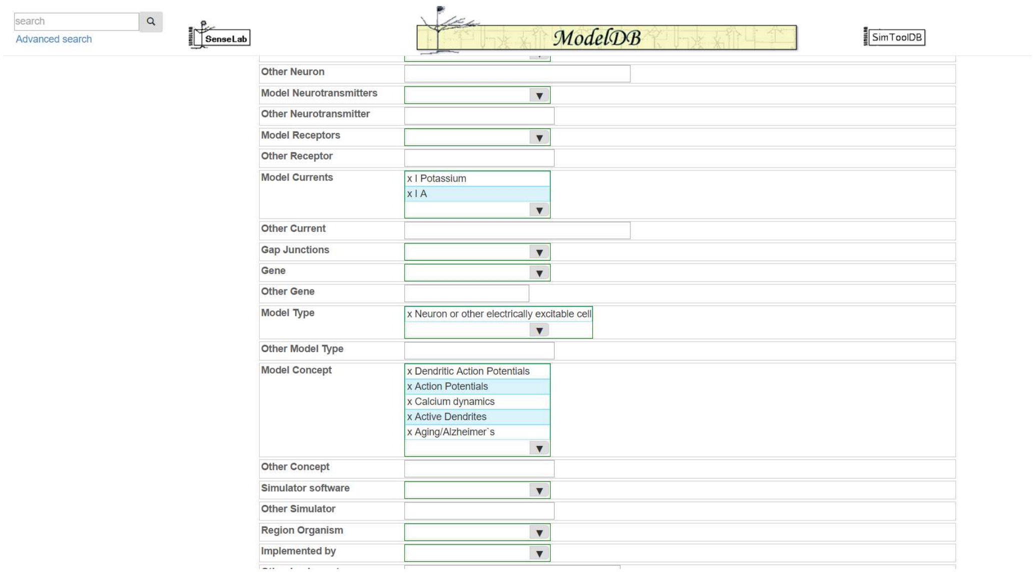
Click the button to automatically find, approve, and populate model entry keywords based on your paper abstract.

Additional information: More information will help your model more discoverable

Model Name

McDougal, Dalal, Shepherd in preparation

Sharing your models

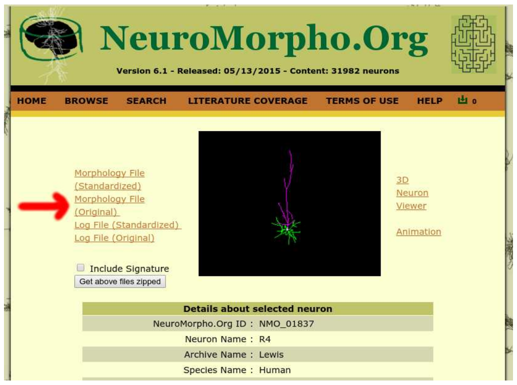


Other Neuron	
Model Neurotransmitters	▼
Other Neurotransmitter	
Model Receptors	▼
Other Receptor	
Model Currents	x I Potassium x I A ▼
Other Current	
Gap Junctions	▼
Gene	▼
Other Gene	
Model Type	x Neuron or other electrically excitable cell ▼
Other Model Type	
Model Concept	x Dendritic Action Potentials x Action Potentials x Calcium dynamics x Active Dendrites x Aging/Alzheimer`s ▼
Other Concept	
Simulator software	▼
Other Simulator	
Region Organism	▼
Implemented by	▼

McDougal, Dalal, Shepherd in preparation

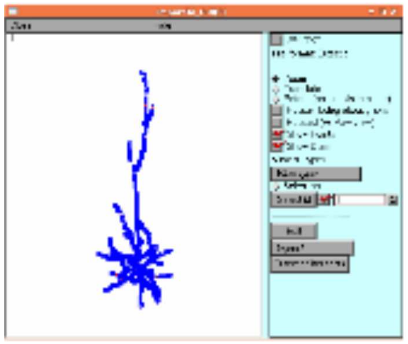
Other resources

NeuroMorpho.Org



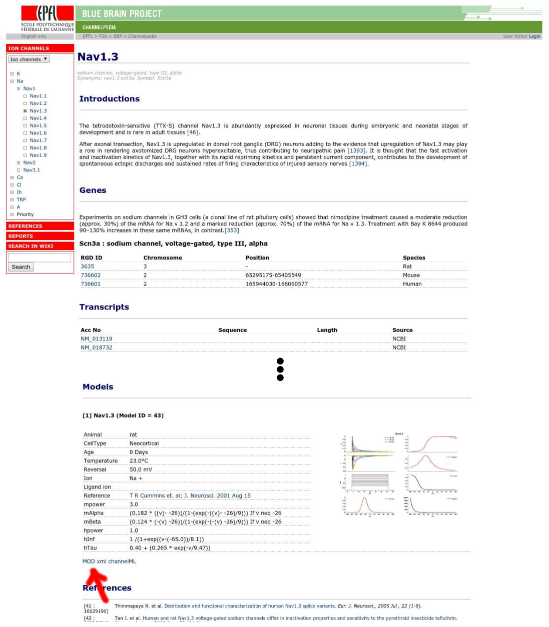
Details about selected neuron

NeuroMorpho.Org ID :	NMO_01837
Neuron Name :	R4
Archive Name :	Lewis
Species Name :	Human


Tools ▶ Miscellaneous ▶ Import 3D

- NeuroMorpho.Org is home to 50,356 reconstructed neurons from 212 cell types and 37 species as of October 24, 2016.
- Warning: not every morphology was reconstructed with the intent of being in a simulation. Before using: rotate to check for *z*-axis errors, check to make sure the diameters are not all equal.
- Use the Import 3D tool to import morphologies into NEURON. For details, see: neuron.yale.edu/neuron/docs/import3d

Channelpedia (Channelpedia.epfl.ch)



Nav1.3

sodium channel, voltage-gated, type III, alpha
Synonyms: nav1.3 scn3a. Symbol: Scn3a

Introductions

The tetrodotoxin-sensitive (TTX-S) channel Nav1.3 is abundantly expressed in neuronal tissues during embryonic and neonatal stages of development and is rare in adult tissues [46].

After axonal transection, Nav1.3 is upregulated in dorsal root ganglia (DRG) neurons adding to the evidence that upregulation of Nav1.3 may play a role in rendering axotomized DRG neurons hyperexcitable, thus contributing to neuropathic pain [1393]. It is thought that the fast activation and inactivation kinetics of Nav1.3, together with its rapid repriming kinetics and persistent current component, contributes to the development of spontaneous ectopic discharges and sustained rates of firing characteristics of injured sensory nerves [1394].

Genes

Experiments on sodium channels in GH3 cells (a clonal line of rat pituitary cells) showed that nimodipine treatment caused a moderate reduction (approx. 30%) of the mRNA for Na v 1.2 and a marked reduction (approx. 70%) of the mRNA for Na v 1.3. Treatment with Bay K 8644 produced 90–130% increases in these same mRNAs, in contrast.[353]

Scn3a : sodium channel, voltage-gated, type III, alpha

RGD ID	Chromosome	Position	Species
3635	3	-	Rat
736602	2	65295175-65405549	Mouse
736601	2	165944030-166060577	Human

Transcripts

Acc No	Sequence	Length	Source
NM_013119			NCBI
NM_018732			NCBI

⋮

Models

[1] Nav1.3 (Model ID = 43)

Animal	rat
CellType	Neocortical
Age	0 Days
Temperature	23.0°C
Reversal	50.0 mV
Ion	Na +
Ligand ion	
Reference	T R Cummins et. al; J. Neurosci. 2001 Aug 15
mpower	3.0
mAlpha	(0.182 * ((v)- -26))/(1-(exp(-((v)- -26)/9))) If v neq -26
mBeta	(0.124 * (-(v) -26))/(1-(exp(-(-(v) -26)/9))) If v neq -26
hpower	1.0
hInf	1 /(1+exp((v-(-65.0))/8.1))
hTau	0.40 + (0.265 * exp(-v/9.47))

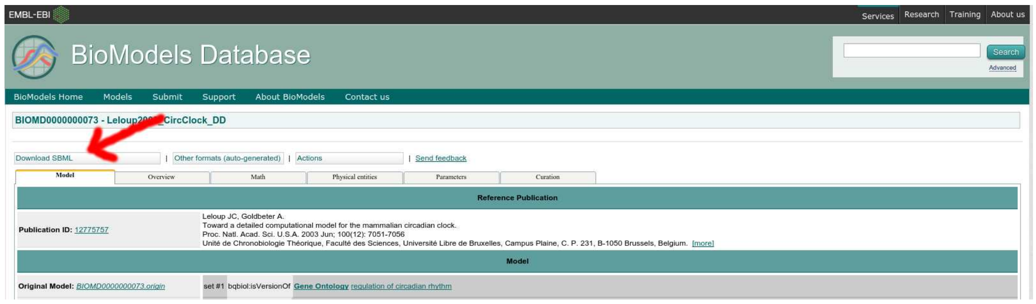
MOD xml channelML

References

[41 : 16029190] Thimmapaya R. et al. Distribution and functional characterization of human Nav1.3 splice variants. *Eur. J. Neurosci., 2005 Jul , 22 (1-9).*
[42 : Tan J. et al. Human and rat Nav1.3 voltage-gated sodium channels differ in inactivation properties and sensitivity to the pyrethroid insecticide tefluthrin.

- Home to information about ion channels.
- Many channels have one or more associated models (e.g. different species or cell types); all are downloadable as MOD files.
- Shows gating variable and channel response to voltage clamp for each model.

Biomodels (www.ebi.ac.uk/biomodels-main)



BIOMD0000000073 - Leloup2003_CircClock_DD

Reference Publication

Publication ID: 12775757	Leloup JC, Goldbeter A. Toward a detailed computational model for the mammalian circadian clock. Proc. Natl. Acad. Sci. U.S.A. 2003 Jun; 100(12): 7051-7056 Unité de Chronobiologie Théorique, Faculté des Sciences, Université Libre de Bruxelles, Campus Plaine, C. P. 231, B-1050 Brussels, Belgium. [more]

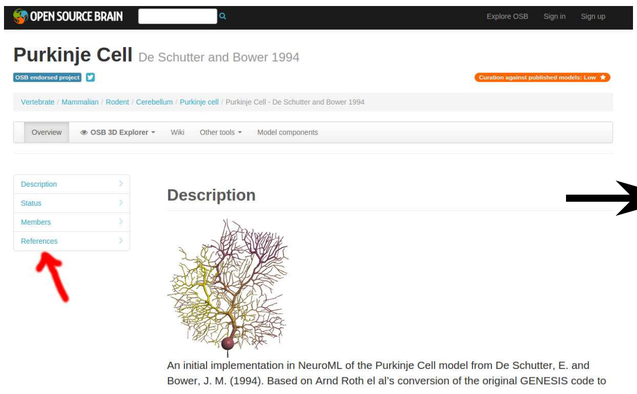
Model

Original Model: *BIOMD0000000073.origin*	set #1 bqbiol:isVersionOf **Gene Ontology** regulation of circadian rhythm

$$\begin{tikzcd}\text{Biomodels model (SBML)} \arrow[r, "\texttt{jnml -sbml-import BIOMD0000000073.xml 1000 5}"'] & \text{LEMS model} \arrow[r, "\texttt{jnml BIOMD0000000073\_LEMS.xml -neuron}"] & \text{MOD file}\end{tikzcd}$$

- Biomodels is a systems biology model repository.
- Models are in SBML but can be converted to MOD files via e.g. jNeuroML (github.com/NeuroML/jNeuroML). Test converted models before using in a larger model. Edits will likely be necessary to get them to interoperate with other mechanisms.
- A native SBML importer for NEURON's rxd module is under development.

Open Source Brain (OpenSourceBrain.org)



OPEN SOURCE BRAIN Explore OSB Sign in Sign up

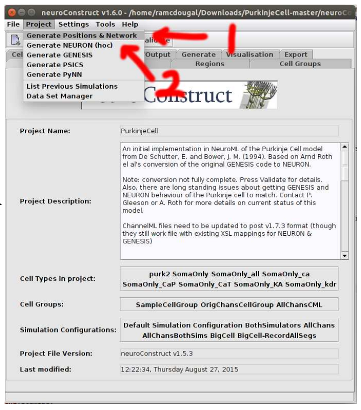
Purkinje Cell De Schutter and Bower 1994

OSB endorsed project Curation against published models: Low

Vertebrate / Mammalian / Rodent / Cerebellum / Purkinje cell / Purkinje Cell - De Schutter and Bower 1994

Description

An initial implementation in NeuroML of the Purkinje Cell model from De Schutter, E. and Bower, J. M. (1994). Based on Arnd Roth el al's conversion of the original GENESIS code to



neuroConstruct v1.6.0 - /home/ramcdougal/Downloads/PurkinjeCell-master/neuroC

Project Name: PurkinjeCell

Project Description: An initial implementation in NeuroML of the Purkinje Cell model from De Schutter, E. and Bower, J. M. (1994). Based on Arnd Roth el al's conversion of the original GENESIS code to NEURON.

Note: conversion not fully complete. Press Validate for details. Also, there are long standing issues about getting GENESIS and NEURON behaviour of the Purkinje cell to match. Contact P. Gleeson or A. Roth for more details on current status of this model.

ChannelML files need to be updated to post v1.7.3 format (though they still work file with existing XSL mappings for NEURON & GENESIS)

Cell Types in project: purk2 SomaOnly SomaOnly_all SomaOnly_ca SomaOnly_CaP SomaOnly_CaT SomaOnly_KA SomaOnly_kdr

Cell Groups: SampleCellGroup OrigChansCellGroup AllChansCML

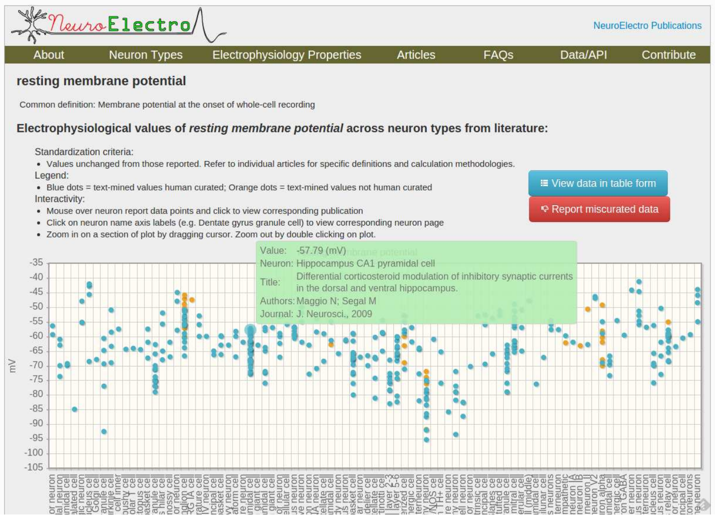
Simulation Configurations: Default Simulation Configuration BothSimulators AllChans AllChansBothSims BigCell BigCell-RecordAllSegs

Project File Version: neuroConstruct v1.5.3

Last modified: 12:22:34, Thursday August 27, 2015

- Open Source Brain promotes collaborative model development via github.
- Models are typically in NeuroML or neuroConstruct format; neuroConstruct (neuroConstruct.org) converts both formats to NEURON.
- The conversion process places different ion channels in different MOD files, which allows extracting model components.

NeuroElectro (NeuroElectro.org)



resting membrane potential

Common definition: Membrane potential at the onset of whole-cell recording

Electrophysiological values of *resting membrane potential* across neuron types from literature:

Standardization criteria:
- Values unchanged from those reported. Refer to individual articles for specific definitions and calculation methodologies.

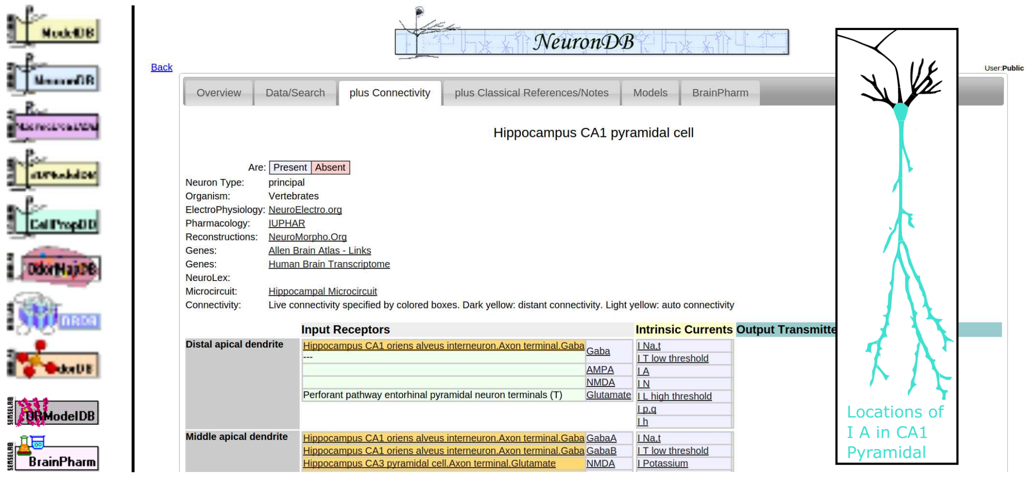
Legend:
- Blue dots = text-mined values human curated; Orange dots = text-mined values not human curated

Interactivity:
- Mouse over neuron report data points and click to view corresponding publication
- Click on neuron name axis labels (e.g. Dentate gyrus granule cell) to view corresponding neuron page
- Zoom in on a section of plot by dragging cursor. Zoom out by double clicking on plot.

Value: -57.79 (mV)
Neuron: Hippocampus CA1 pyramidal cell
Title: Differential corticosteroid modulation of inhibitory synaptic currents in the dorsal and ventral hippocampus.
Authors: Maggio N; Segal M
Journal: J. Neurosci., 2009

- NeuroElectro archives experimentally measured electrophysiology values for different cell types; it shows the spread and allows comparing values across different cell types.
- Read the paper associated with a value to understand: species, experimental conditions, etc.

SenseLab (senselab.med.yale.edu)



Back

Overview | Data/Search | plus Connectivity | plus Classical References/Notes | Models | BrainPharm

User:Public

Hippocampus CA1 pyramidal cell

Are:	Present / Absent
Neuron Type:	principal
Organism:	Vertebrates
ElectroPhysiology:	NeuroElectro.org
Pharmacology:	IUPHAR
Reconstructions:	NeuroMorpho.Org
Genes:	Allen Brain Atlas - Links
Genes:	Human Brain Transcriptome
NeuroLex:	
Microcircuit:	Hippocampal Microcircuit
Connectivity:	Live connectivity specified by colored boxes. Dark yellow: distant connectivity. Light yellow: auto connectivity

	Input Receptors		Intrinsic Currents	Output Transmitte
Distal apical dendrite	Hippocampus CA1 oriens alveus interneuron.Axon terminal.Gaba ---	Gaba	I Na,t	
			I T low threshold	
		AMPA	I A	
		NMDA	I N	
	Perforant pathway entorhinal pyramidal neuron terminals (T)	Glutamate	I L high threshold	
			I p,q	
			I h	
Middle apical dendrite	Hippocampus CA1 oriens alveus interneuron.Axon terminal.Gaba	GabaA	I Na,t	
	Hippocampus CA1 oriens alveus interneuron.Axon terminal.Gaba	GabaB	I T low threshold	
	Hippocampus CA3 pyramidal cell.Axon terminal.Glutamate	NMDA	I Potassium	

- SenseLab is a suite of 10 interconnected databases (listed at left).
- ModelDB and NeuronDB (at right) are the most useful for modeling.
- NeuronDB shows what channels are present and the inputs and outputs *by cell region* (e.g. distal apical dendrite vs proximal apical dendrite).

Stay up to date

Twitter

Many groups announce new developments on Twitter, including:

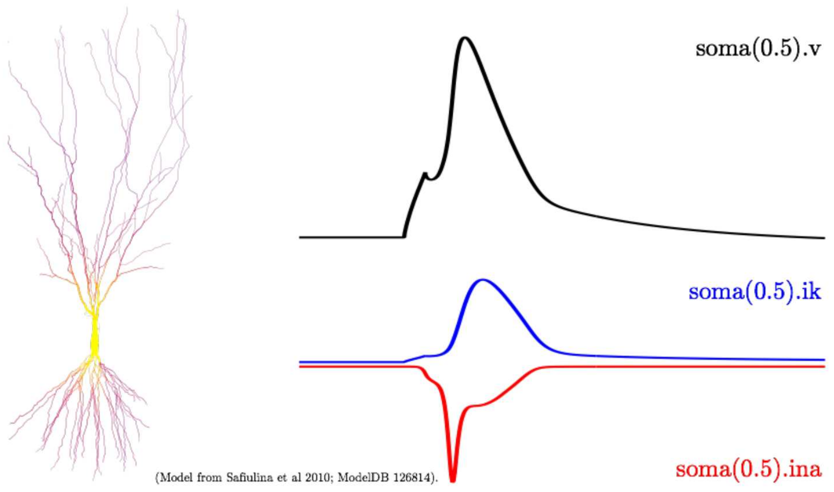
- SenseLab (including ModelDB): @SenseLabProject
- Open Source Brain: @OSBTeam
- NeuroMorpho.Org: @NeuroMorphoOrg
- ICGenealogy Project: @ICGenealogy
- Int. Neuroinformatics Coordinating Facility (INCF): @INCForg

Modeling neuronal reaction-diffusion

Robert A McDougal

7 June 2017

Neurons generate action potentials by moving ions across their membrane.

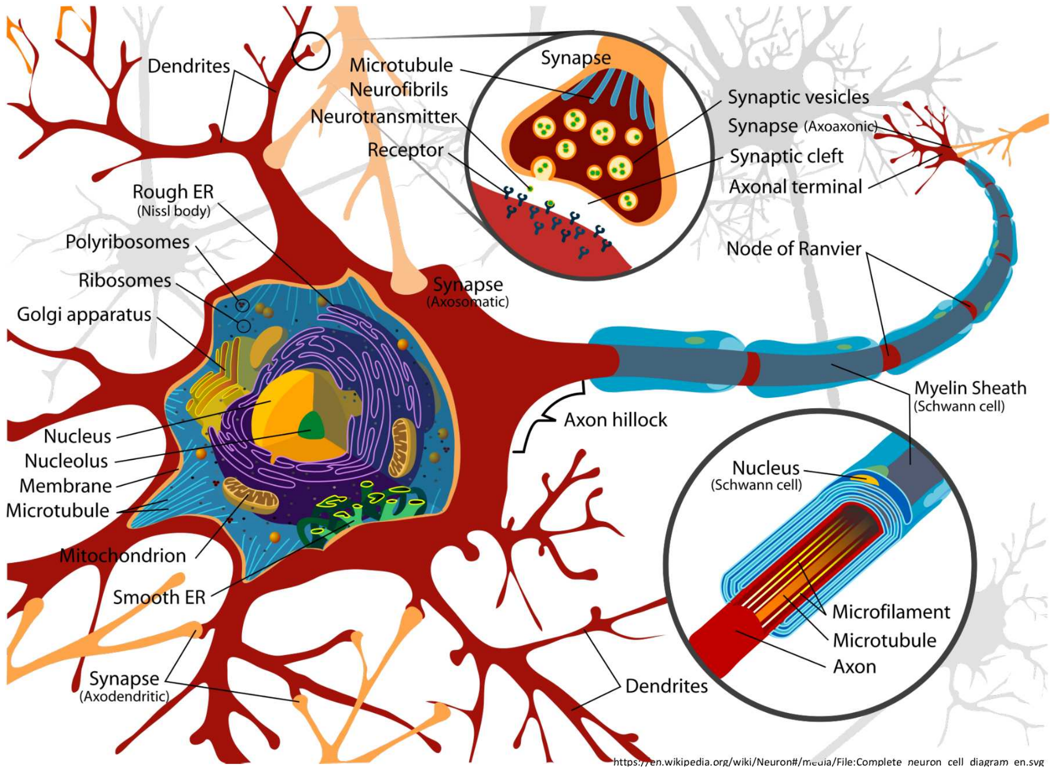


(Model from Safiulina et al 2010; ModelDB 126814).

A neuron is not a transistor

≠

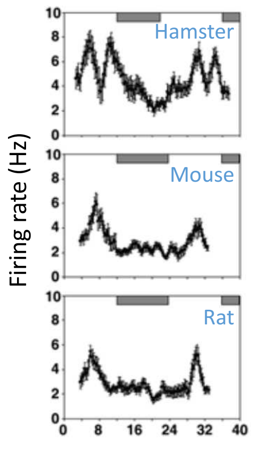
https://commons.wikimedia.org/wiki/File:Bc635-transistor.png
Neuron from Pyapali et al 1998 via http://neuromorpho.org/neuron_info.jsp?neuron_name=n123



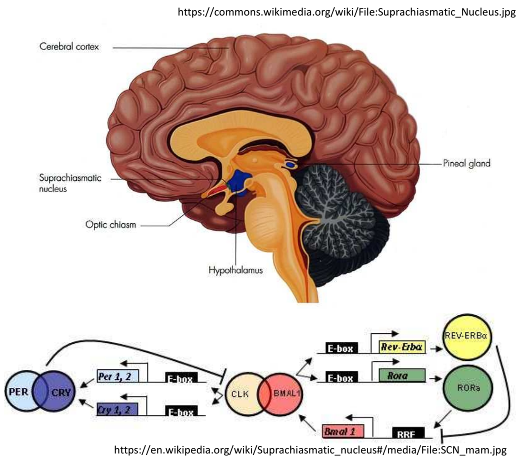
https://en.wikipedia.org/wiki/Neuron#/media/File:Complete_neuron_cell_diagram_en.svg

Neurons have state

(example: protein oscillations in the SCN)

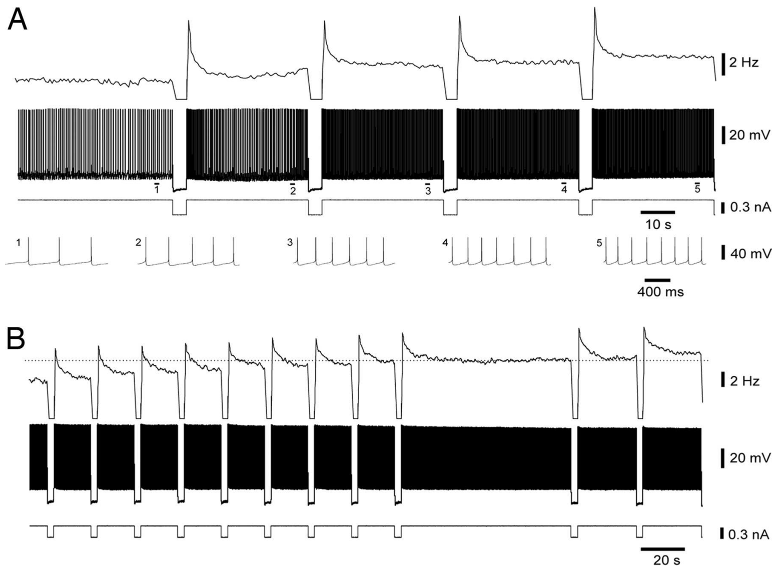


Burgoon et al 2004


https://commons.wikimedia.org/wiki/File:Suprachiasmatic_Nucleus.jpg
Cerebral cortex
Pineal gland
Suprachiasmatic nucleus
Optic chiasm
Hypothalamus
PER
CRY
Per 1, 2
Cry 1, 2
E-box
CLK
BMAL1
Rev-Erbα
REV-ERBα
Rorα
RORα
Bmal 1
RRE
https://en.wikipedia.org/wiki/Suprachiasmatic_nucleus#/media/File:SCN_mam.jpg

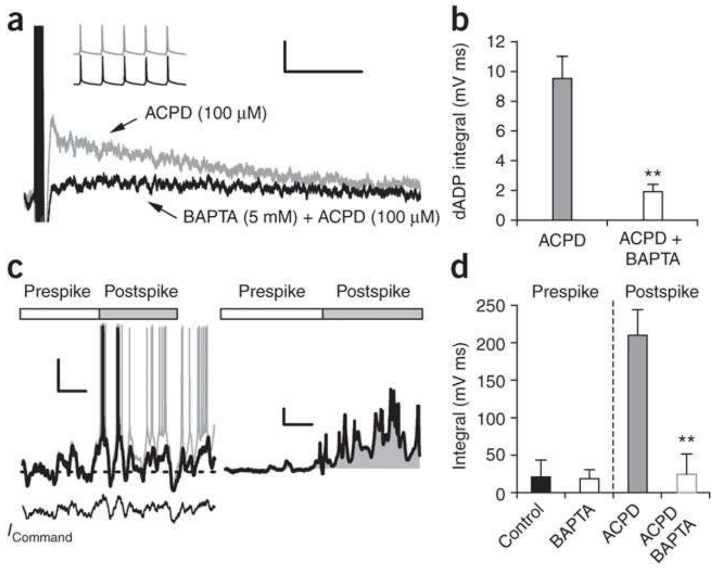
Neurons have state

(example: HAGPA in PFC)



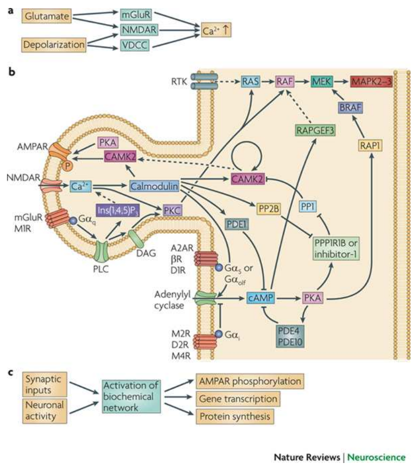
Winograd et al 2008.

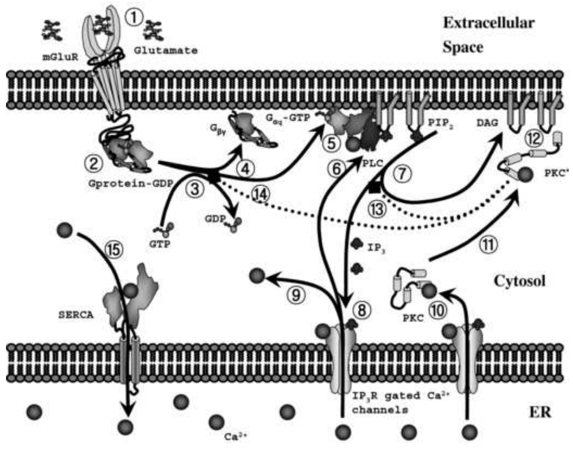
Neurons have state
(example: intracellular calcium)



Sidiropoulou et al 2009

Neurons have state
(example: synaptic pathways)





Jeanette Hellgren Kotaleski & Kim T. Blackwell 2010.

Minchul Kang and Hans G Othmer 2007.

How do we model this?

“Reaction–diffusion systems are mathematical models which explain how the concentration of one or more substances distributed in space changes under the influence of two processes: local chemical reactions in which the substances are transformed into each other, and diffusion which causes the substances to spread out over a surface in space.”

https://en.wikipedia.org/wiki/Reaction%E2%80%93diffusion_system

Mass-Action kinetics

The model

- A reaction's product is formed at a rate proportional to the concentration of the reactants.

Example

- Consider the reaction

$$\mathrm{Na} + \mathrm{Cl} \xrightarrow{k} \mathrm{NaCl}$$

- Then:

$$\begin{aligned}[\mathrm{Na}]' &= -k[\mathrm{Na}][\mathrm{Cl}]\\ [\mathrm{Cl}]' &= -k[\mathrm{Na}][\mathrm{Cl}]\\ [\mathrm{NaCl}]' &= k[\mathrm{Na}][\mathrm{Cl}]\end{aligned}$$

Conservation of mass.

Matter is neither created nor destroyed by reactions.

In our equations, this means:

[Na] + [NaCl] = constant
[Cl] + [NaCl] = constant

Example

Using the law of mass-action, we can write a system of equations describing the formation of *calcium chloride*:

$$\mathrm{Ca} + 2\ \mathrm{Cl} \underset{k_b}{\overset{k_f}{\rightleftarrows}} \mathrm{CaCl_2}$$

$$\begin{aligned}[\mathrm{Ca}]' &= -k_f[\mathrm{Ca}][\mathrm{Cl}]^2 + k_b[\mathrm{CaCl_2}]\\ [\mathrm{Cl}]' &= -2k_f[\mathrm{Ca}][\mathrm{Cl}]^2 + k_b[\mathrm{CaCl_2}]\\ [\mathrm{CaCl_2}]' &= 2k_f[\mathrm{Ca}][\mathrm{Cl}]^2 - k_b[\mathrm{CaCl_2}]\end{aligned}$$

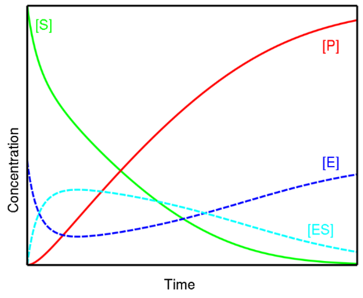
Enzyme kinetics

It is generally **not** the case that a substrate transforms directly into a product:

$$S \rightarrow P$$

Instead, an enzyme is often involved:

$$\mathrm{E + S} \underset{k_b}{\overset{k_f}{\rightleftarrows}} \mathrm{ES} \xrightarrow{k_{\mathrm{cat}}} \mathrm{E + P}$$



https://commons.wikimedia.org/wiki/File:Michaelis_Menten_S_P_E_ES.svg

Michaelis-Menten

If we can assume either:

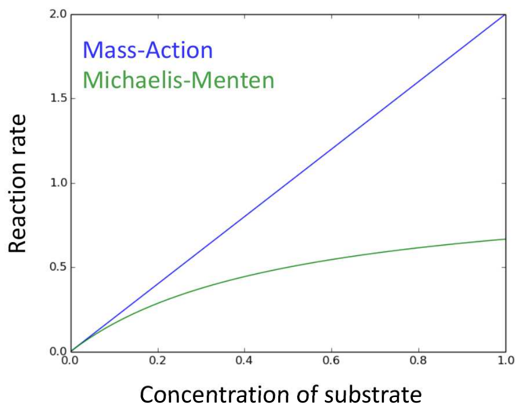
- the substrate (S) and the complex (ES) are in instantaneous equilibrium, or
- the concentration of the complex (ES) does not change on the time-scale of product formation

Then the rate of the enzymatic reaction reduces to:

$$\frac{V_{max}\,[S]}{K_M + [S]}$$

K_M is called the *Michaelis constant*. It is the concentration at which the reaction proceeds at half its maximum rate.

Michaelis-Menten vs Mass-Action



$S \rightarrow P$

Both curves on the left have the same rate of reaction when the substrate concentration is low, but the Michaelis-Menten rate levels off (due to limited enzyme availability) as concentrations increase.

$$y = 2x$$

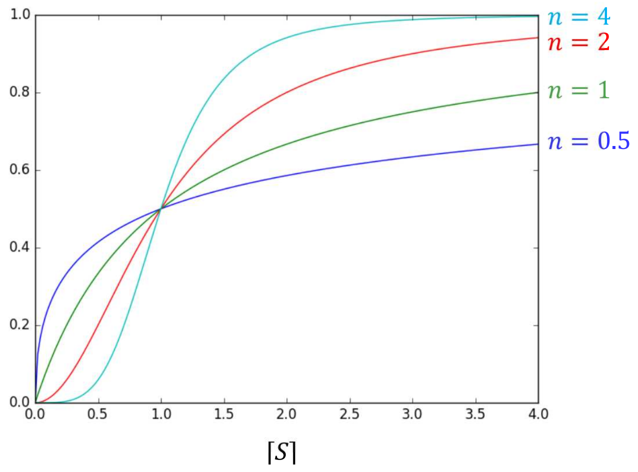
$$y = \frac{x}{x + 0.5}$$

Hill equation: cooperative binding

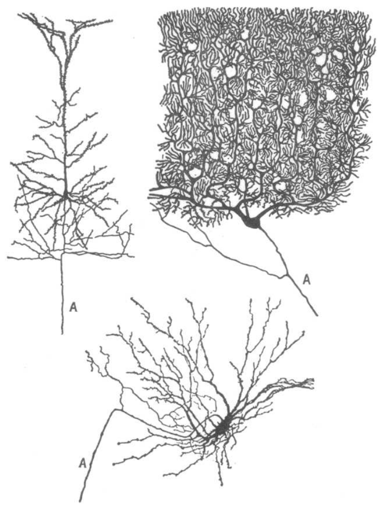
$$\frac{V_{max}\,[S]^n}{[k_A]^n + [S]^n}$$

If n > 1, positive cooperativity.
If n < 1, negative cooperativity.

$$\frac{[S]^n}{1^n + [S]^n}$$



Neurons have spatial extent



Cajal 1909 as reproduced in Rall 1962.

Effects of non-point-ness:

- Ion and protein concentrations vary with space.
- Cellular mechanisms (ER, ion channels, etc) vary with space.

Concentrations at different locations affect each other:

- Transport
- Diffusion

Fick's First Law and the diffusion equation

Fick's First Law:

- Diffusive flux is proportional to the concentration gradient.

$$J = -D\nabla\varphi$$

- Here D is called the *diffusion coefficient*.

Fick's Second Law (the diffusion equation):

$$\frac{\partial\varphi}{\partial t} = \nabla \cdot (D\nabla\varphi) = D\ \nabla^2\varphi$$

where the last equality only holds if D is constant.

Practical limits of pure diffusion

The expected time $\mathrm{E}[t]$ for a molecule with diffusion constant D to diffuse a distance x is:

$$\mathrm{E}[t] = \frac{x^2}{2D}$$

So in particular, if

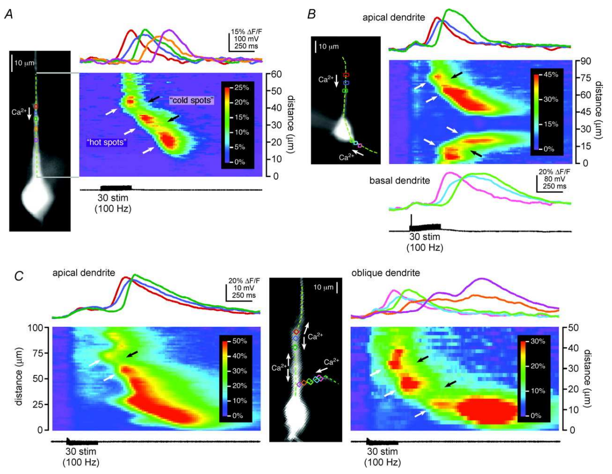
$D = 1$ µm^2/ms and

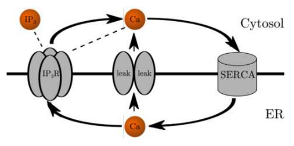
$x = 100$ µm,

Then

$E[t] = \frac{100^2}{2} = 5000$ ms.

Diffusion with regenerative dynamics can quickly spread signals





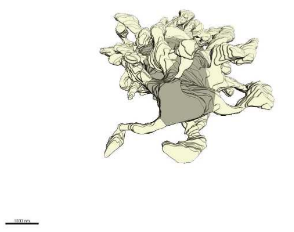
Adapted from Neymotin et al 2015

Fitzpatrick et al 2009.

Where does diffusion occur?

- Cytosol
  - But not full cross section because of organelles
- Organelles (e.g. ER)
- Extracellular space
  - Tortuosity
  - Anisotropy
  - Volume fraction

Calcium in spines



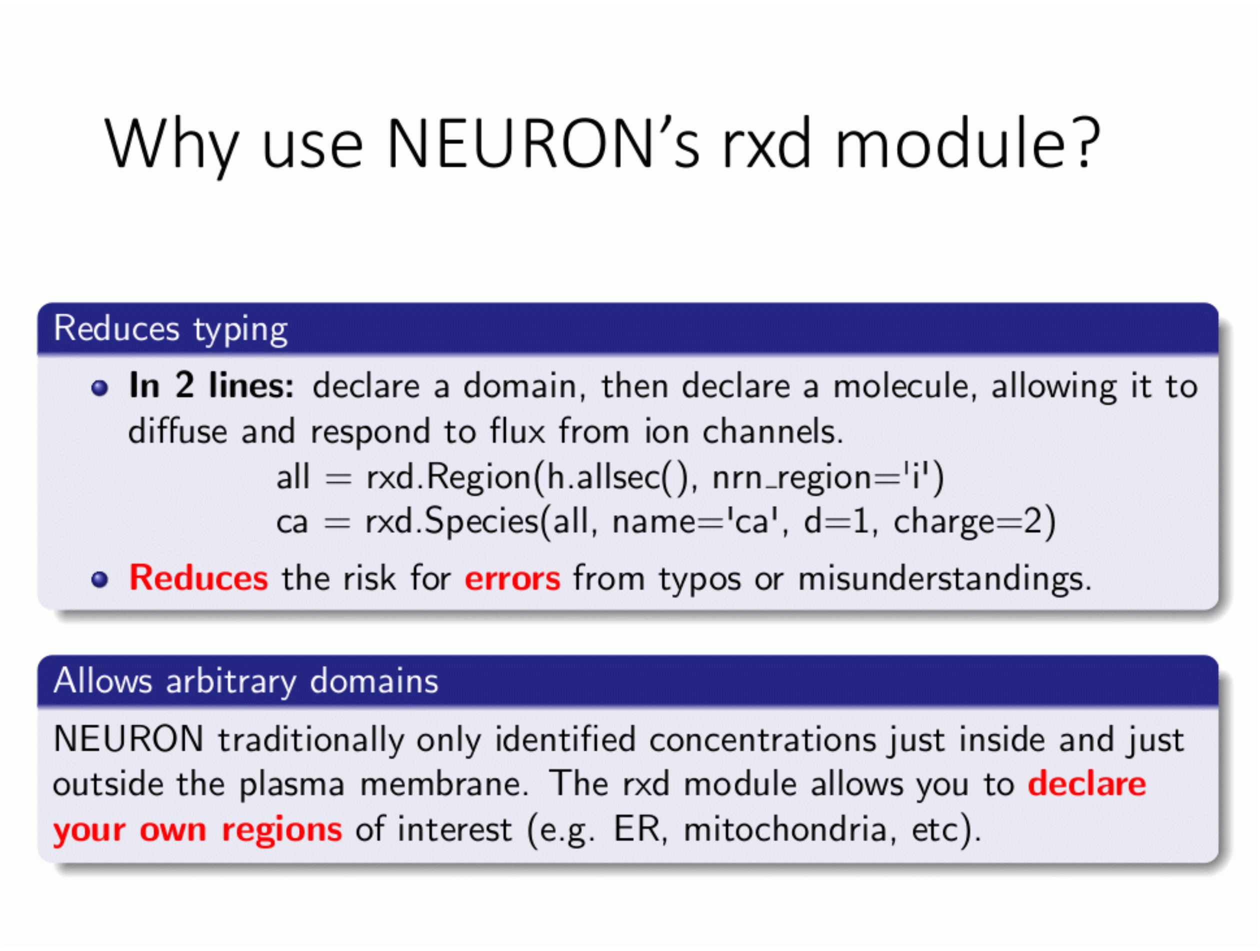
http://dx.doi.org/10.6084/m9.figshare.1266444

A typical dendritic spine head may have a volume of **0.5 µm³**.

A typical cytosolic calcium concentration is **100 nM**.

At these levels, how many molecules of calcium are in a dendritic spine head? What is the percentage change in concentration if one molecule leaves the spine head?

Reaction-diffusion in NEURON



Why use NEURON's rxd module?

Reduces typing

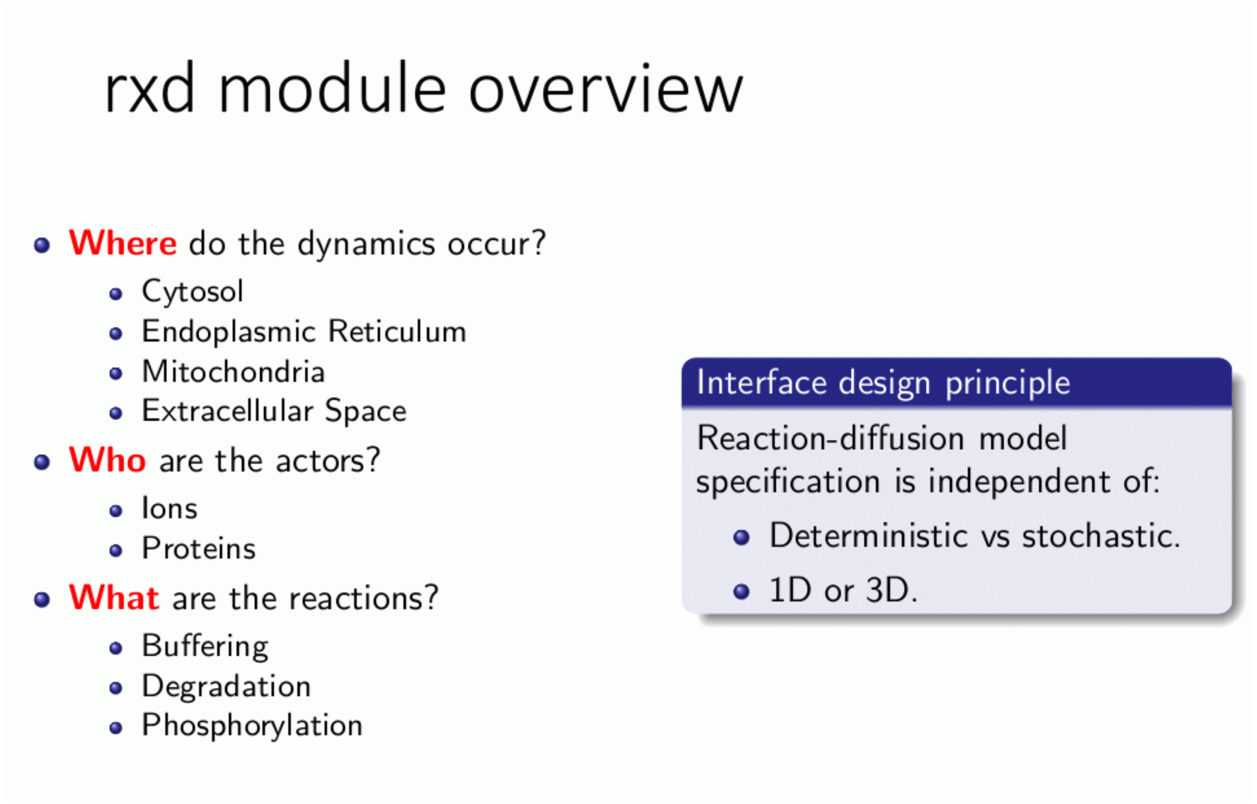
- **In 2 lines:** declare a domain, then declare a molecule, allowing it to diffuse and respond to flux from ion channels.

```
all = rxd.Region(h.allsec(), nrn_region='i')
ca = rxd.Species(all, name='ca', d=1, charge=2)
```

- **Reduces** the risk for **errors** from typos or misunderstandings.

Allows arbitrary domains

NEURON traditionally only identified concentrations just inside and just outside the plasma membrane. The rxd module allows you to **declare your own regions** of interest (e.g. ER, mitochondria, etc).



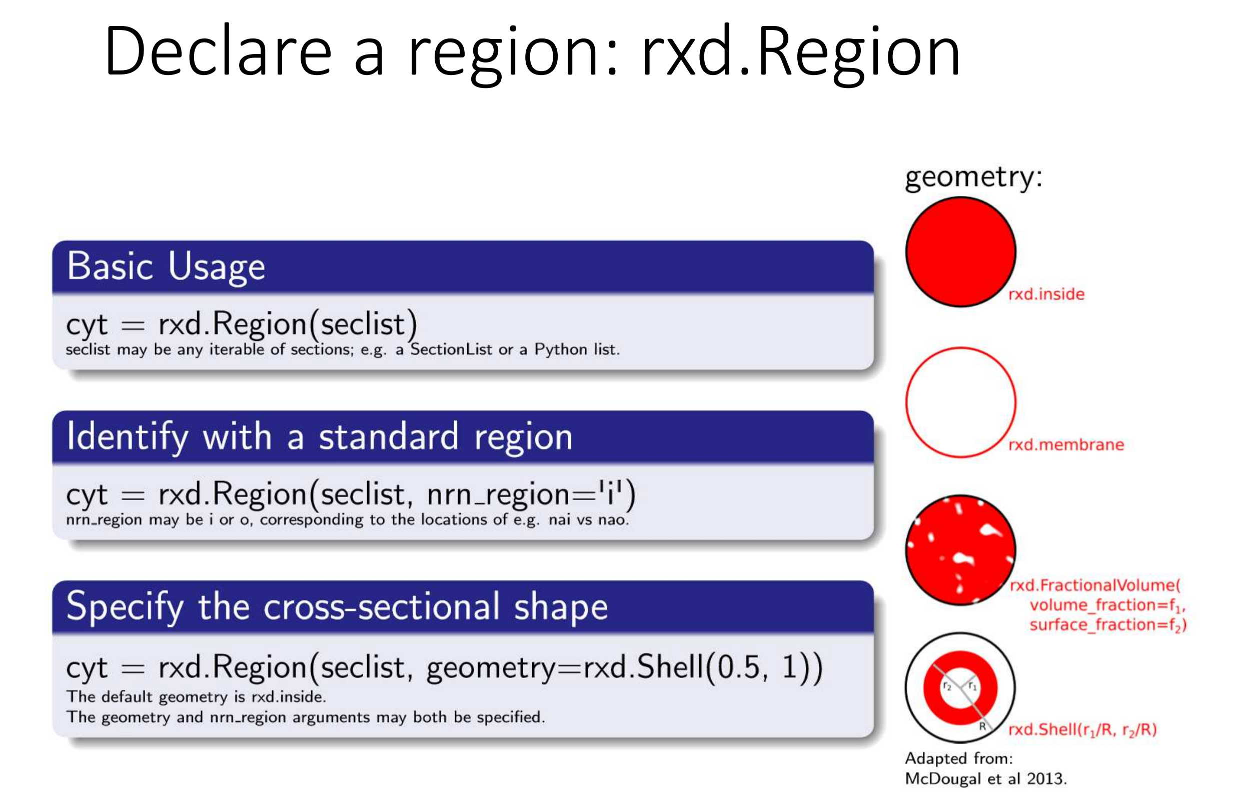
rxd module overview

- **Where** do the dynamics occur?
  - Cytosol
  - Endoplasmic Reticulum
  - Mitochondria
  - Extracellular Space
- **Who** are the actors?
  - Ions
  - Proteins
- **What** are the reactions?
  - Buffering
  - Degradation
  - Phosphorylation

Interface design principle

Reaction-diffusion model specification is independent of:

- Deterministic vs stochastic.
- 1D or 3D.



Declare a region: rxd.Region

Basic Usage

cyt = rxd.Region(seclist)
seclist may be any iterable of sections; e.g. a SectionList or a Python list.

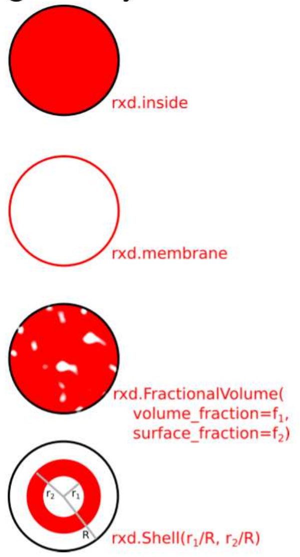
Identify with a standard region

cyt = rxd.Region(seclist, nrn_region='i')
nrn_region may be i or o, corresponding to the locations of e.g. nai vs nao.

Specify the cross-sectional shape

cyt = rxd.Region(seclist, geometry=rxd.Shell(0.5, 1))
The default geometry is rxd.inside.
The geometry and nrn_region arguments may both be specified.

geometry:


rxd.inside
rxd.membrane
rxd.FractionalVolume(volume_fraction=f1, surface_fraction=f2)
r2
r1
R
rxd.Shell(r1/R, r2/R)

Adapted from:
McDougal et al 2013.

rxd.Region tips

Specify nrn_region if concentrations interact with NMODL

If NMODL mechanisms (ion channels, point processes, etc) depend on or affect the concentration of a species living in a given region, that region must declare a nrn_region (typically 'i').

To declare a region that exists on all sections

r = rxd.Region(h.allsec())

Use list comprehensions to select sections

r = rxd.Region([sec for sec in h.allsec() if 'apical' in sec.name()])

Declare ions & proteins: rxd.Species

Basic usage

protein = rxd.Species(region, d=16)

d is the diffusion constant in $\mu m^2/ms$. region is an rxd.Region or an iterable of rxd.Region objects.

Initial conditions

protein = rxd.Species(region, initial=value)

value is in mM. It may be a constant or a function of the node.

Connecting with HOC

ca = rxd.Species(region, name='ca', charge=2)

If the nrn_region of region is "i", the concentrations of this species will be stored in cai, and its concentrations will be affected by ica.

Specifying dynamics: rxd.Reaction

Mass-action kinetics

$\mathrm{ca} + \mathrm{buffer} \xleftrightarrow[kb]{kf} \mathrm{cabuffer}$

buffering = rxd.Reaction(ca + buffer, cabuffer, kf, kb)

kf is the forward reaction rate, kb is the backward reaction rate. kb may be omitted if the reaction is unidirectional.
In a mass-action reaction, the reaction rate is proportional to the product of the concentrations of the reactants.

Repeated reactants

$2\mathrm{H} + \mathrm{O} \xleftrightarrow[kb]{kf} \mathrm{H2O}$

water_reaction = rxd.Reaction(2 * H + O, H2O, kf, kb)

Arbitrary reaction formula, e.g. Hill dynamics

$\mathrm{a} + \mathrm{b} \longrightarrow \mathrm{c}$

hill_reaction = rxd.Reaction(a + b, c, a ^ 2 / (a ^ 2 + k ^ 2), mass_action=False)

Hill dynamics are often used to model cooperative reactions.

rxd.Rate and rxd.MultiCompartmentReaction

rxd.Rate

Use rxd.Rate to specify an explicit contribution to the rate of change of some concentration or state variable.

```
ip3degradation = rxd.Rate(ip3, -k * ip3)
```

rxd.MultiCompartmentReaction

Use rxd.MultiCompartmentReaction when the dynamics span multiple regions; e.g. a pump or channel.

```
ip3r = rxd.MultiCompartmentReaction(ca[er], ca[cyt], kf, kb,
                                    membrane=cyt_er_membrane)
```

The rate of these dynamics is proportional to the membrane area.

Manipulating nodes

Getting a list of nodes

- nodelist = protein.nodes

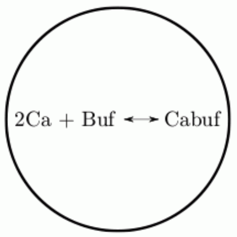
Filtering a list of nodes

- nodelist2 = nodelist(region)
- nodelist2 = nodelist(0.5)
- nodelist2 = nodelist(section)(region)(0.5)

Other operations

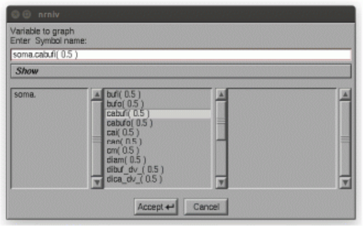
- nodelist.concentration = value
- values = nodelist.concentration
- surface_areas = nodelist.surface_area
- volumes = nodelist.volume
- node = nodelist[0]

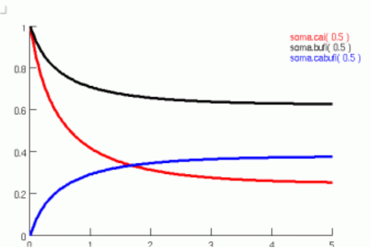
Example: Calcium buffering



```
from neuron import h, rxd, gui

h('create soma')
soma_region = rxd.Region([h.soma], nrn_region='i')

ca = rxd.Species(soma_region, initial=1,
                 name='ca', charge=2)
buf = rxd.Species(soma_region, initial=1,
                  name='buf')
cabuf = rxd.Species(soma_region, initial=0,
                    name='cabuf')

buffering = rxd.Reaction(2 * ca + buf, cabuf, 1, 0.1)
```

Use the GUI to create a graph and run the simulation.





Concentration pointers

To get a pointer to a concentration, use `node._ref_concentration`:

Recording traces

```
v = h.Vector()
v.record(ca.nodes[0]._ref_concentration)
```

Plotting

```
g = h.Graph()
g.addvar('ca[er][dend](0.5)', ca.nodes(er)(dend)(0.5)[0]._ref_concentration)
h.graphList[0].append(g)
```

Tips

To find out what properties and methods are available, use `dir`; e.g.

```
dir(ca.nodes)
```

NEURON's variable step solver has a default absolute tolerance of 0.001.

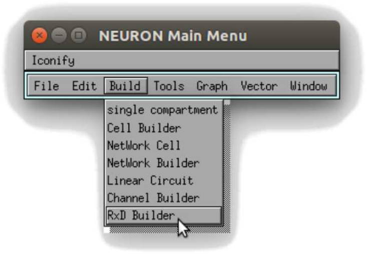
Since NEURON measures concentration in mM and some cell biology concentrations (e.g. calcium) are in µM, this tolerance may be too high. Compensate by using an `atolscale` in the constructor*, e.g.

```
ca = h.Species(cyt, atolscale=1e-6)
```

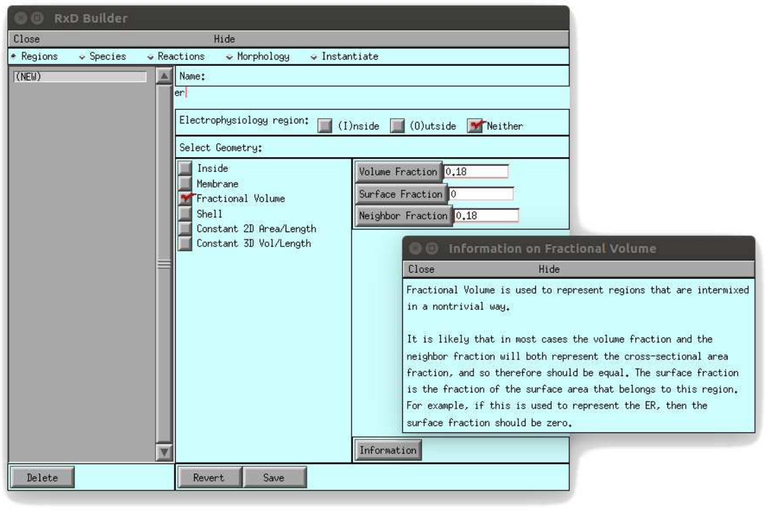
* atolscale is only supported in the development version; on older versions of NEURON, change the scale globally, e.g. h.Cvode().atol(1e-8).

GUI-based specification

Reaction-diffusion dynamics can also be specified using the GUI. This option appears only when rxd is supported in your install (Python and scipy must be available).



GUI-based specification



Name: er

Electrophysiology region: (I)nside (O)utside [✓ Neither]

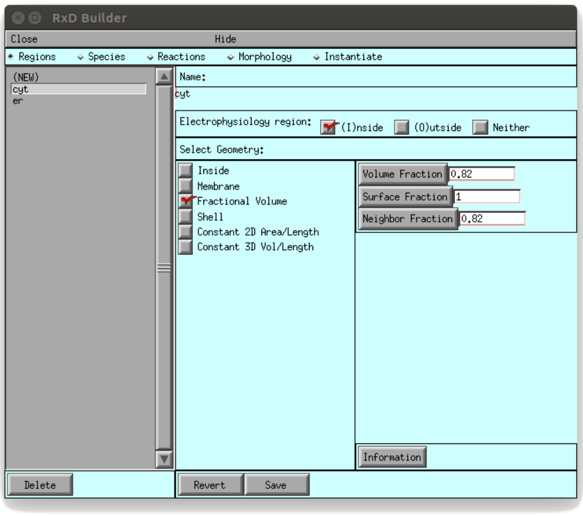
[✓ Fractional Volume]

Volume Fraction 0.18
Surface Fraction 0
Neighbor Fraction 0.18

Fractional Volume is used to represent regions that are intermixed in a nontrivial way.

It is likely that in most cases the volume fraction and the neighbor fraction will both represent the cross-sectional area fraction, and so therefore should be equal. The surface fraction is the fraction of the surface area that belongs to this region. For example, if this is used to represent the ER, then the surface fraction should be zero.

GUI-based specification



(NEW)
cyt
er

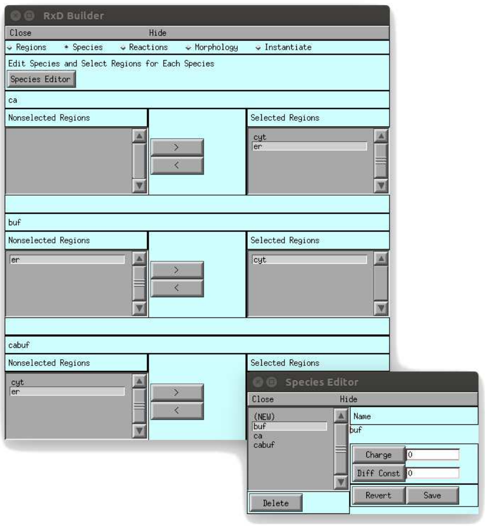
Name:
cyt

Electrophysiology region: [✓ (I)nside] (O)utside Neither

Select Geometry:
Inside
Membrane
[✓ Fractional Volume]
Shell
Constant 2D Area/Length
Constant 3D Vol/Length

Volume Fraction	0.82
Surface Fraction	1
Neighbor Fraction	0.82

GUI-based specification



Edit Species and Select Regions for Each Species

ca
Nonselected Regions
Selected Regions
cyt
er

buf
Nonselected Regions
er
Selected Regions
cyt

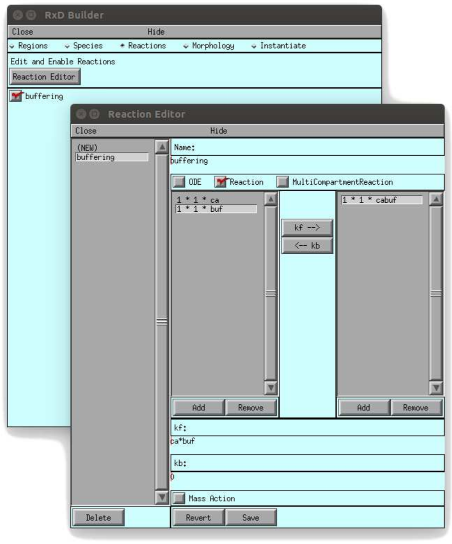
cabuf
Nonselected Regions
cyt
er
Selected Regions

(NEW)
buf
ca
cabuf

Name
buf

Charge	0
Diff Const	0

GUI-based specification



Edit and Enable Reactions

[✓] buffering

(NEW)
buffering

Name:
buffering

ODE [✓ Reaction] MultiCompartmentReaction

1 * 1 * ca
1 * 1 * buf

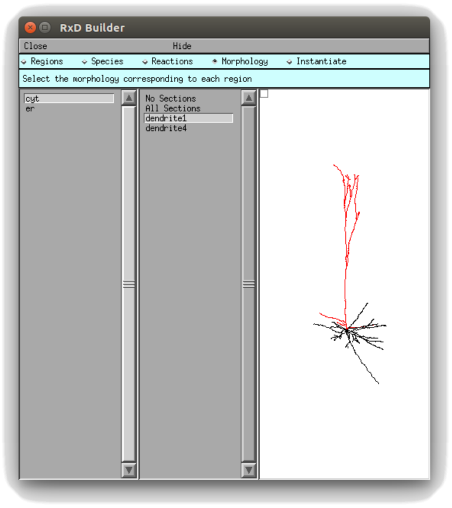
1 * 1 * cabuf

kf:
ca*buf

kb:
0

Mass Action

GUI-based specification

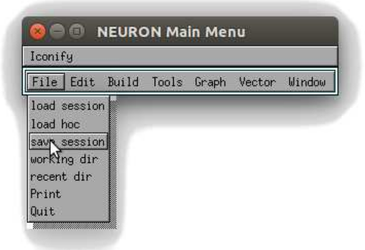


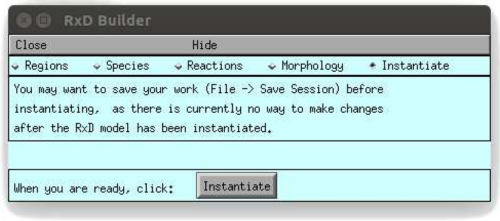
Select the morphology corresponding to each region

cyt
er

No Sections
All Sections
dendrite1
dendrite4

GUI-based specification

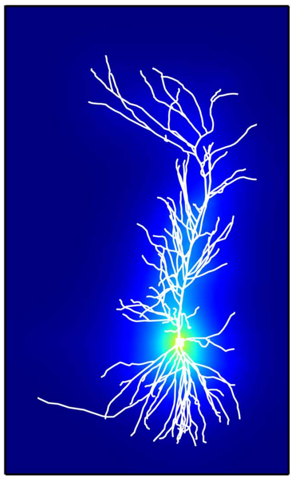




You may want to save your work (File -> Save Session) before instantiating, as there is currently no way to make changes after the RxD model has been instantiated.

When you are ready, click: Instantiate

Extracellular diffusion*

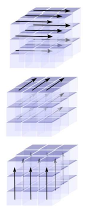


New region type:

```
ecs = rxd.Extracellular(xlo, ylo, zlo, xhi, yhi, zhi,
                        dx=dx, tortuosity=1, volume_fraction=1)
```

Setting/getting extracellular concentrations:

```
ca[ecs].states3d[5:15, 5:15, :] = 1
pyplot.imshow(ca[ecs].states3d[:, :, 0],
              interpolation='nearest', vmin=0, vmax=1,
              extent=ca[ecs].extent('xy'), origin='lower')
```



$$1-\frac{r_x}{2}\nabla_x^2\phi^{(t+\frac{1}{3})}=\left(\frac{r_x}{2}\nabla_x^2+r_y\nabla_y^2+r_z\nabla_z^2\right)\phi^{(t)}$$

$$1-\frac{r_y}{2}\nabla_y^2\phi^{(t+\frac{2}{3})}=-\frac{r_y}{2}\nabla_y^2\phi^{(t+\frac{1}{3})}$$

$$1-\frac{r_z}{2}\nabla_z^2\phi^{(t+1)}=-\frac{r_z}{2}\nabla_z^2\phi^{(t+\frac{2}{3})}$$

We use a finite-volume method, the Douglas-Gunn Alternating Direction Implicit algorithm is **unconditionally stable**.

Each time-step is divided into an x-, y- and z-direction and requires solving diagonally dominant tridiagonal systems of equations. This is solved with the **Thomas algorithm**, so the runtime scales linearly with the number of voxels.

We currently support zero-flux Neumann boundary conditions which conserves the total concentration.

* Extracellular diffusion support is currently only available in the development version.

3D Simulations

Specifying 3D Simulations

Just add one line of code[2]:

```
rxd.set_solve_type(dimension=3)
all = rxd.Region(h.allsec())
ca = rxd.Species(all, d=1)
ca.initial = lambda node: 1 if node.x3d < 50 else 0
```

Plotting

Get the concentration values expressed on a regular 3D grid via nodelist.value_to_grid()

```
values = ca.nodes.value_to_grid()
```

Pass the result to a 3d volume plotter, such as Mayavi's VolumeSlicer:

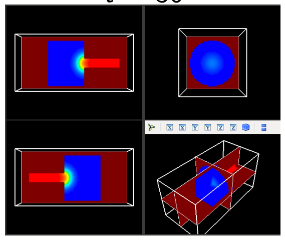
```
graph = VolumeSlicer(data=ca.nodes.value_to_grid())
graph.configure_traits()
```

[2] `rxd.set_solve_type` can optionally take a list of sections as its first argument; in that case only the specified sections will be simulated in three dimensions.

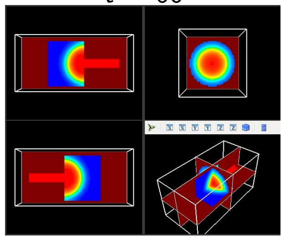
Example: wave curvature

```
from neuron import h, gui, rxd
import volume_slicer

sec1, sec2 = h.Section(), h.Section()
h.pt3dadd(2, 0, 0, 2, sec=sec1)
h.pt3dadd(9.9, 0, 0, 2, sec=sec1)
h.pt3dadd(10, 0, 0, 2, sec=sec1)
h.pt3dadd(10, 0, 0, 10, sec=sec2)
h.pt3dadd(18, 0, 0, 10, sec=sec2)

def do_init(node):
    return 1 if node.x3d < 8 else 0

all3d = rxd.Region(h.allsec(), dimension=3)
ca = rxd.Species(all3d, initial=do_init, d=0.05)
r = rxd.Rate(ca, -ca * (1 - ca) * (0.1 - ca))

def plot_it():
    graph = volume_slicer.VolumeSlicer(
        data=ca.nodes.value_to_grid(),
        vmin=0, vmax=1)
    graph.configure_traits()

h.finitialize()
for t in [30, 60]:
    h.continuerun(t)
    plot_it()
```

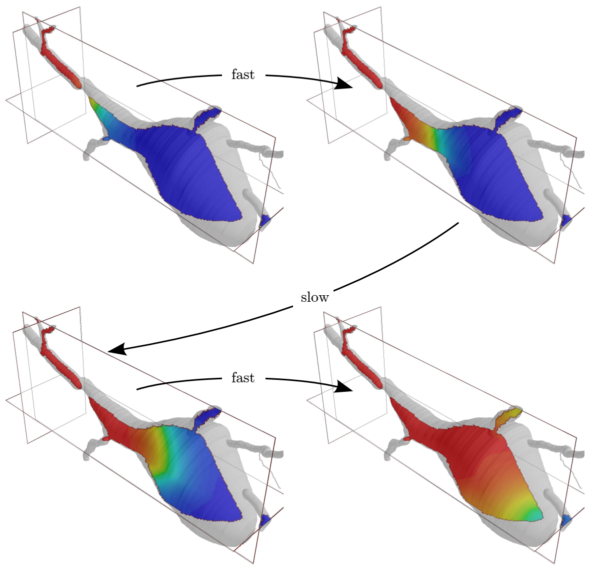
$t = 30$



$t = 60$



Wave curvature at soma entry



Under development

- Enhancements to extracellular diffusion.
- Stochastic reaction-diffusion.
- SBML support.
- Better reaction-diffusion performance.
- Parallel reaction-diffusion.

Contact us if you would like to alpha test any of these features.

For more information

Journal Articles on Reaction-Diffusion in NEURON

- McDougal, R. A., Hines, M. L., Lytton, W. W. (2013). Reaction-diffusion in the NEURON simulator. *Frontiers in Neuroinformatics*, 7.
- McDougal, R. A., Hines, M. L., Lytton, W. W. (2013). Water-tight membranes from neuronal morphology files. *Journal of Neuroscience Methods*, 220(2), 167-178.

Online Resources

- NEURON Forum
- Programmer's Reference
- NEURON Reaction-Diffusion Tutorials

Receipt

Received:	$170
From:	
For:	Using the NEURON Simulation Environment Held Nov. 10, 2017 in Washinton, DC https://www.neuron.yale.edu/neuron/static/courses/dc2017/dc2017.html
By:	N.T. Carnevale Director, Using the NEURON Simulation Environment 203-494-7381 ted.carnevale@yale.edu
For deposit in:	Yale University account "NNC--Fees"

Survey

We'd appreciate your frank opinions and suggestions to help us refine this course and design future offerings on related subjects.

Please score these	. . . . .	**according to this scale**	
Overall impression	____	no opinion	0
Relevance to my research	____	poor, not helpful	1
Didactic presentations	____	fair	2
Written handouts	____	good	3
Slides	____	excellent, very helpful	4
Computer projection	____		
Classroom	____		
Food	____		

Best feature ______________________________

Weakest feature ______________________________

____

Additional topics that should be covered, topics that should receive more or less coverage, or other suggestions for improvement.

____

____

____

____

Circle one

Y N I would recommend this course to others who are interested in neural modeling.

Y N I have developed my own modeling software using a high-level language (FORTRAN, C/C++, Python etc.).

Y N I have created my own models using modeling software.

Which software? ______________________________

My primary area of research interest is ______________________________

To help us better meet the needs of NEURON users, please circle all platforms that you plan to use for modeling.

Hardware Mac PC Other ______________________________

OS MacOS X Win 7 | 8 | 9 | 10 UNIX | Linux | OS X | BSD

If Linux, which distribution? ______________________________