

# 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.

# Practical uses of computational modeling in neuroscience research

Test plausibility: can phenomenon X be explained by  
{what we know} + {what we can reasonably assume}?

Predict experimental results:

- Given model A, what phenomenon X should happen?
- How will experimental procedure B affect phenomenon X?
- Will experimental procedure B cause  
some new phenomenon Y?

Improve understanding of

- phenomenon X under conditions of health or disease
- something you learned from a course, book,  
or scientific paper (private enlightenment)

# The NEURON Simulation Environment

For models of neurons and neural circuits that are

- closely linked to experimental observations

and involve

- complex anatomical and biophysical properties
- electrical and/or chemical signaling

## **Features**

- Performance
- Ease of use
- Active development
- User support
- Large user base

# The NEURON Simulation Environment

## Active development

- Open source project directed by Michael Hines at Yale  
<https://github.com/neuronsimulator/nrn>
- Supported by NIH and European Human Brain Project

## User support

- Downloads, documentation, tutorials at  
<https://nrn.readthedocs.io/>
- Individual user support via Forum  
<https://www.neuron.yale.edu/phpBB/>,  
email, phone / Zoom / Skype etc.
- Courses: meetings (SFN, OCNS, other),  
summer course, workshops, webinars

# The NEURON User Community

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

As of February 2023

- > 2800 publications
- > 2000 subscribers to Forum and mailing list
- ModelDB <https://modeldb.science/> has source code for > 800 published models that used NEURON

*plus code for ~1000 models that used MATLAB, C/C++, Python, XPPAUT, Brian etc.*

# Topics

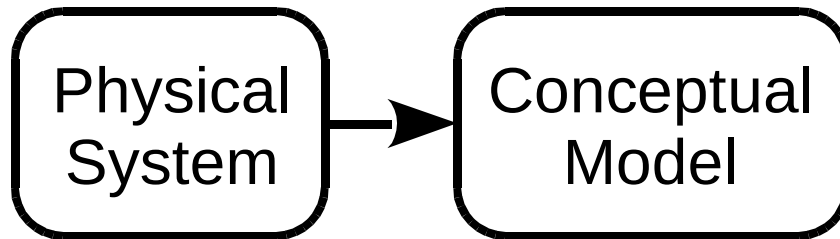
1. How to create and use models of neurons and networks of neurons
  - How to specify anatomical and biophysical properties
  - How to control, display, and analyze models and simulation results
2. How NEURON works
3. How to add user-defined biophysical mechanisms

# From Physical System to Computational Model



Physical  
System

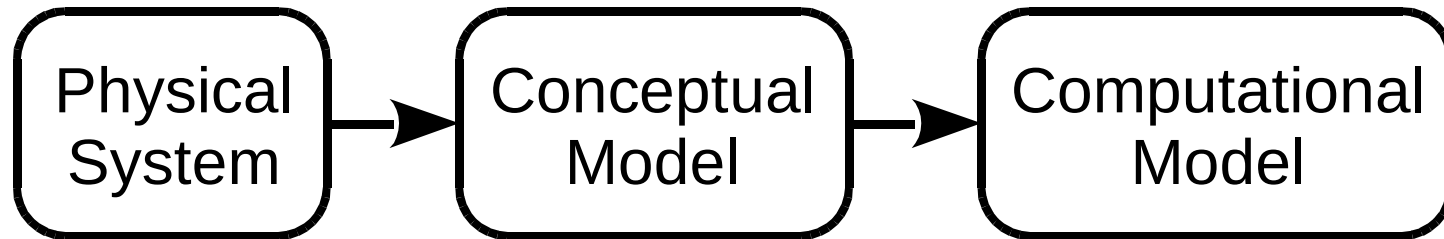
# From Physical System to Computational Model



Conceptual model:  
simplified representation of the physical system



# From Physical System to Computational Model



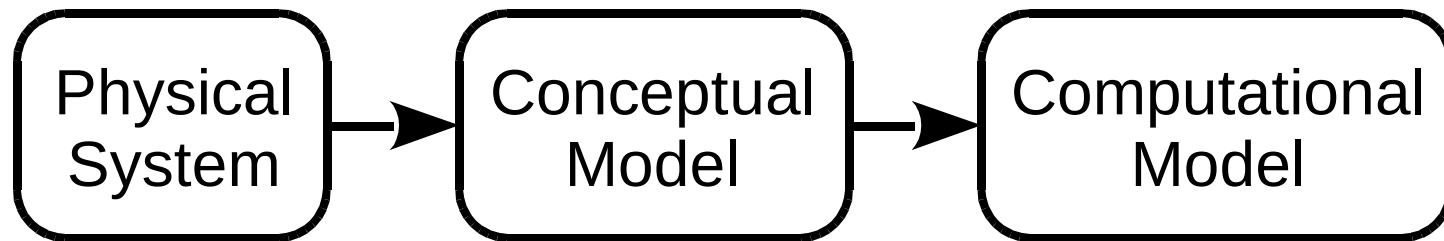
Conceptual model:

simplified representation of the physical system

Computational model:

accurate representation of the conceptual model

# From Physical System to Computational Model



Conceptual model:

simplified representation of the physical system

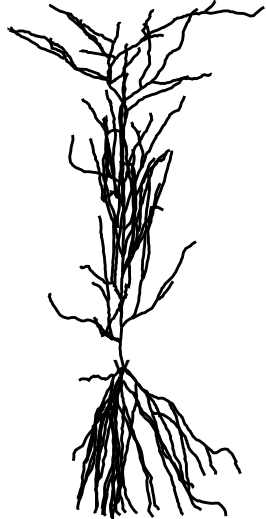
Computational model:

accurate representation of the conceptual model

*A close match between the conceptual model  
and  
the computational model  
is essential ("conceptual control").*

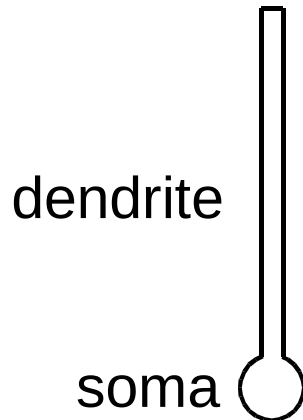
# From Physical System to Computational Model

Physical  
system



Ca1  
pyramidal  
cell

Conceptual  
model



ball  
and  
stick

Computational  
model

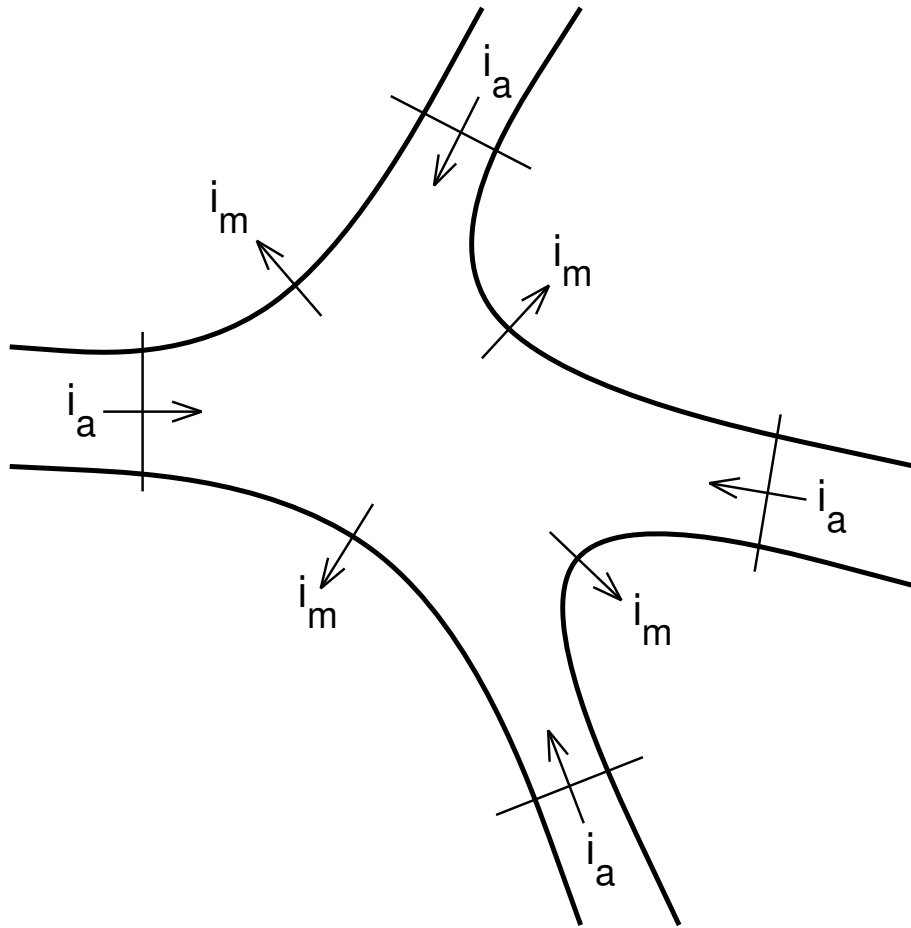
```
# python
soma = h.Section(name='soma')
dendrite = h.Section(name='dendrite')
dendrite.connect(soma(1))
```

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

# Fundamental Concepts

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

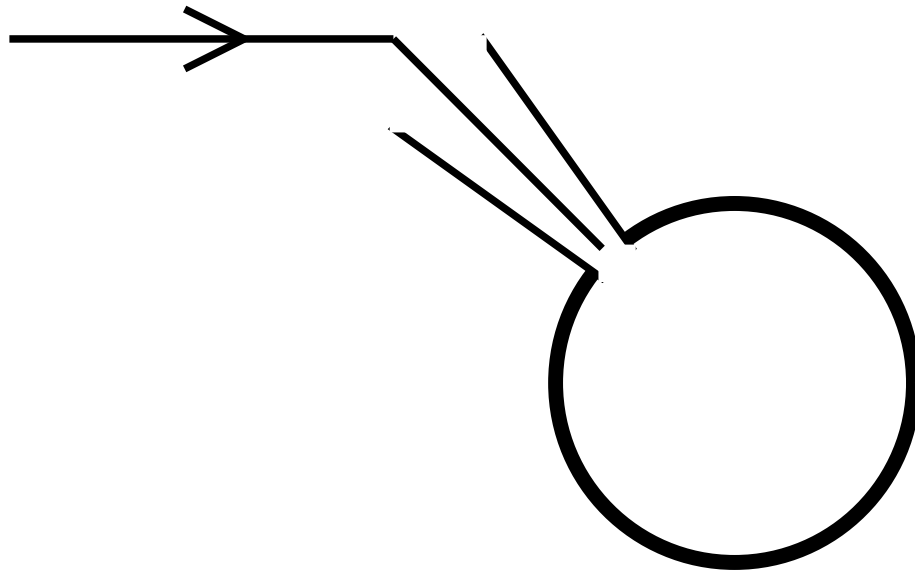
# Conservation of Charge



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

# Electrically Compact Model

Injected  
current



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

# Is NEURON 8.x installed and working?

In a terminal (user entries **bold**):

```
[ted@blitz Desktop]$ neurondemo
```

```
NEURON -- VERSION 8.2.0+ HEAD (156b9dee+) 2022-07-01
```

```
Duke, Yale, and the BlueBrain Project -- Copyright 1984-2021
```

```
See http://neuron.yale.edu/neuron/credits
```

```
loading membrane mechanisms from  
/home/ted/.local/lib/python3.8/site-  
packages/neuron/.data/share/nrn/demo/release/x86_64/.libs/libnrnm  
ech.so
```

```
Additional mechanisms from files
```

```
"cabpump.mod" "cachan1.mod" "camchan.mod" "capump.mod"  
"invlfire.mod" "khhchan.mod" "mcna.mod" "nacaex.mod" "nachan.mod"  
"release.mod"
```

NEURON Demonstrations panel: click on radio button next to "Release"

RunControl panel: click on "Init & Run". Did it work?

Try the other models if you wish. To quit, execute

```
oc>>quit()
```

# Can Python use NEURON as a module?

... and do you have Python?

In a terminal (user entries **bold**):

```
[ted@blitz Desktop]$ python3  
Python 3.8.10 (default, Mar 13 2023, 10:26:41)  
[GCC 9.4.0] on linux  
Type "help", "copyright", "credits" or "license" for more  
information.  
>>> from neuron import h, gui  
>>>
```

Importing `h` and `gui` should generate no error message, and you should see the NEURON Main Menu toolbar.

```
>>> quit()
```



# Exercise: Single Compartment Model

Start with a lipid bilayer (no channels)

Add linear ion channels (passive leak)

Use the GUI to:

- build a model and an interface for using it
- run simulations and analyze results
- change stimulus params (intensity and duration)
- adjust graphical displays of simulation results
- adjust simulation params  $dt$  and Points Plotted / ms

# An aside: why the GUI?

The GUI always works--no typos, can't violate syntax.

Ideal for interactive tasks: development, debugging, exploratory simulations (how well do you understand your own models?).

Intuitive, easily understood presentation of model specification and simulation results.

Simplifies many tasks that would be tedious to do by writing your own code, e.g. CellBuilder, Channel Builder, Linear Circuit Builder, Impedance tools, VariableStepControl, ModelView, even the lowly Shape Plot.

*Adding a customized user interface to your own programs makes it easier to collaborate with non-programmers.*

The most powerful approach to using NEURON: combine the GUI and your own code to exploit the strengths of both.

# Single Compartment *continued*

Open a terminal window and execute

```
cd Desktop  
mkdir simplecell  
cd simplecell
```

Now you have a place to work.

Start Python and tell it to use the neuron module.

```
python3
```

At Python's >>> prompt execute

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

# Single Compartment *continued*

Get a CellBuilder.

**Neuron Main Menu / Build / CellBuilder**

Set soma's surface area to 100  $\mu\text{m}^2$ .

**CellBuilder / Geometry** tab

Click on **soma**

Click on the **area** checkbox

Toggle **Specify Strategy** off

Make sure the numeric field next to  
the **area( $\mu\text{m}^2$ )** button says **100**.

# Single Compartment *continued*

Set soma membrane capacitance **cm** to 1 uF/cm<sup>2</sup>.

**CellBuilder / Biophysics** tab

Click on **soma**

Click on the **cm** checkbox to toggle it on.

The other checkboxes (**Ra**, **pas**, **extracellular**, **hh**) should be off.

Toggle **Specify Strategy** off.

Make sure the numeric field next to the **cm(uF/cm<sup>2</sup>)** button says 1.

# Single Compartment *continued*

Before doing anything else, save your work.

**Neuron Main Menu / File / save session**

Popup window: enter **simplecell.ses**

Click on **Save** button to write a file called

**simplecell.ses**

in your **Desktop/simplecell** directory.

You can use `simplecell.ses` to recreate your configured CellBuilder.

# Single Compartment *continued*

Next click on the **Continuous Create** button.

Your computer's memory now contains a model cell with the properties specified by the CellBuilder.

Let's do some experiments on this model.

You'll need:

a RunControl panel to launch simulations

**NMM/Tools/RunControl**

a graph of soma membrane potential vs. time

**NMM/Graph/Voltage axis**

# Single Compartment *continued*

a current clamp ("IClamp") that injects a 1 nA x 1 ms current pulse that starts at  $t = 1$  ms.

**NMM/Tools/Point Processes/Managers/  
Point Manager**

Make this be a current clamp.

**PointProcessManager/SelectPointProcess/  
IClamp**

Show the IClamp's parameters.

**PPM/Show/Parameters**

Set **delay(ms)**, **dur(ms)**, and **amp(nA)** to **1**.



# Single Compartment *continued*

Plot stimulus current vs. time.

**NMM/Graph/Current axis**

Make it plot the IClamp's i variable.

Click on the **Graph's menu button** (left upper corner of its canvas) and select "**Plot what?**"

Click in the variable name browser's edit field and enter

**IClamp[0].i**

then click

**Accept**

# Single Compartment *continued*

Before you do anything else

1. Save your work to a session file. You might call this **cell1rig.ses**.
2. On a sheet of paper sketch what you think the plots of somatic membrane potential and IClamp[0].i will look like.

Then launch a simulation by clicking on  
**RunControl / Init & Run**

Compare the results with your sketches.

# Single Compartment *continued*

In the RunControl panel note the simulation parameters **dt**, **Tstop**, and **Points plotted/ms**.

**dt** specifies the interval between solution points.

**Tstop** specifies simulation duration in "model time."

**Points plotted/ms** sets the interval between points plotted in graphs.

Plotting interval must be a whole multiple of dt

plotting interval in ms =  $i * dt$  where  $i = 1, 2, \dots$

so points plotted/ms =  $1 / (i * dt)$

During initialization, NEURON forces dt to a value consistent with points plotted/ms.

# Single Compartment *continued*

Things to do:

Quit Python (enter **quit()** at the >>> prompt, or click on **NMM / File / Quit**).

Start Python and load the session file you saved earlier.

```
[ted@blitz simplecell]$ python3  
>>> from neuron import h, gui  
>>> h.load_file('cell1rig.ses')  
>>>
```

The model cell and custom user interface you saved are ready to be used.

# Single Compartment *continued*

Things to do:

Use "**View = plot**" to rescale Graph axes.

Maximum soma.v too big? Divide IClamp.amp by 10 and try again.

What IClamp.amp depolarizes the model by 10 mV?

The 0.1 nA x 1 ms current pulse drives soma.v from \_\_\_\_ to \_\_\_\_ mV, i.e. a \_\_\_\_ mV depolarization from rest.

This is a linear model, so dividing IClamp.amp by \_\_\_\_ should reduce the depolarization to 10 mV. Does it?

# Single Compartment *continued*

Insert passive channels and see how this affects the model.

In the CellBuilder click on **Biophysics**, then toggle **Specify Strategy** on. Click on the **pas** button so it shows a check mark.

What happened to the model cell's resting potential?

How can we initialize this model to its resting potential?

# Single Compartment *continued*

Inserting pas changed the model cell's resting potential from -65 to -70 mV.

To initialize this model to its new resting potential, change the numeric value next to the RunControl panel's Init button from -65 to -70 (controls value of simulation parameter **v\_init**).

# Single Compartment *continued*

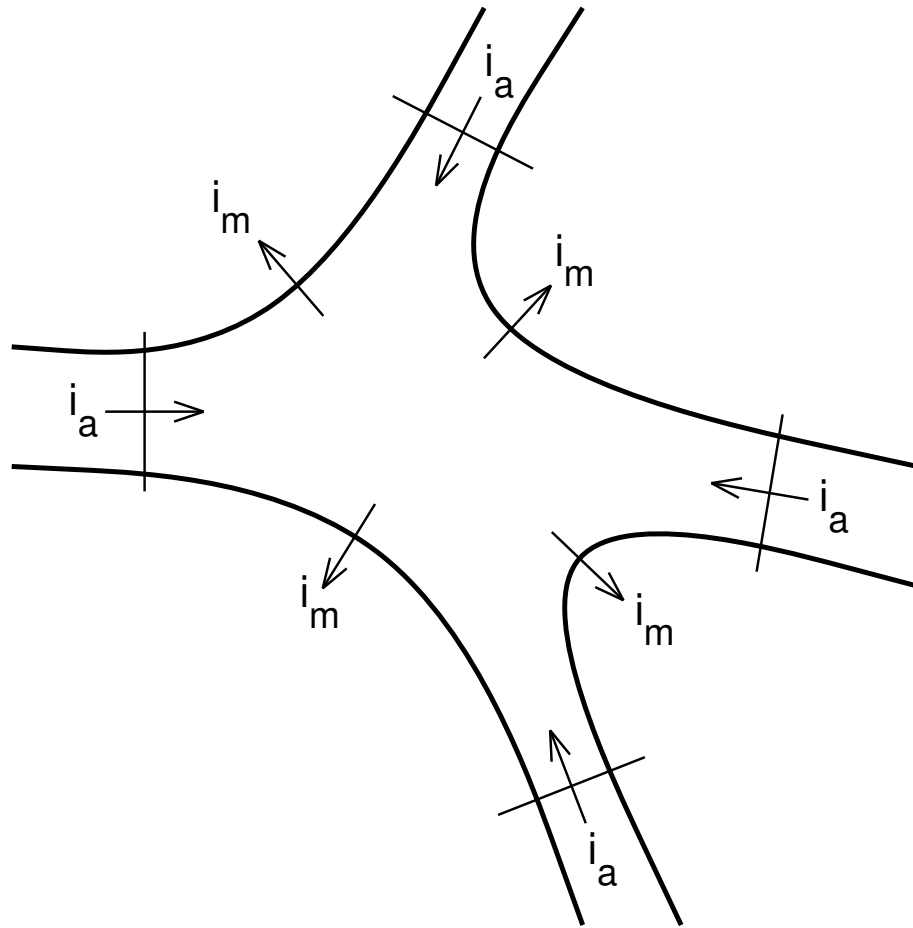
Change **IClamp.amp** to 1 A (1e9 nA) and run a simulation. Rescale the graphs if necessary. What would happen to a real cell?

Change **IClamp.amp** to 1 nA and dur to 0.01 ms. What happens and why?

Change **Points plotted/ms** to 100 and try again. What happens now, and why?



# Conservation of Charge



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

# Models with Significant Electrical Extent

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

$v_j$       membrane potential in compartment j

$i_{ion_j}$       net transmembrane ionic current in compartment j

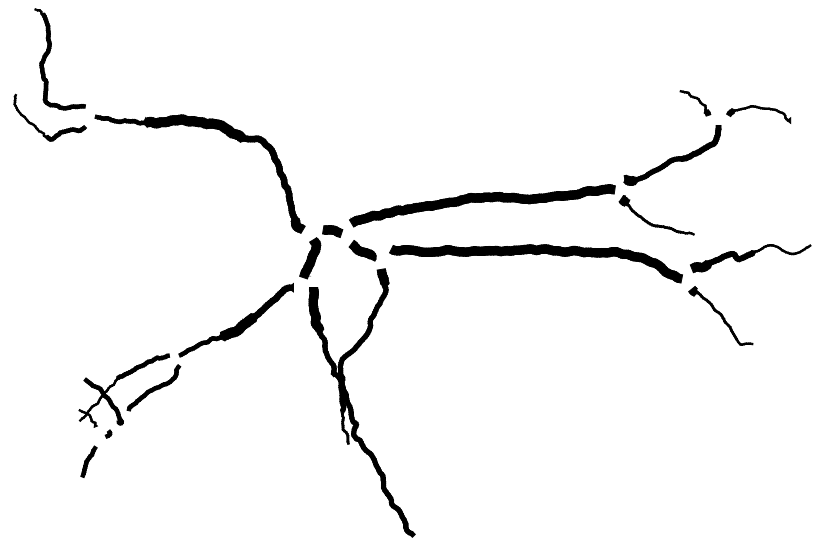
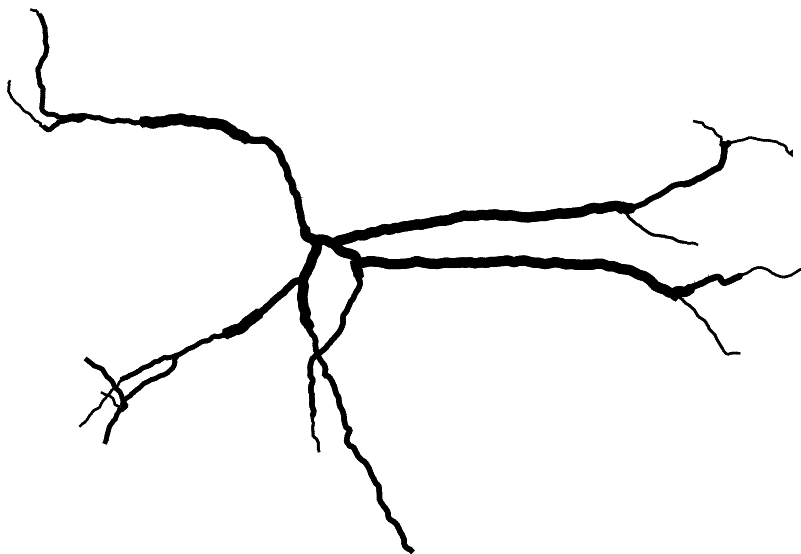
$c_j$       membrane capacitance of compartment j

$r_{jk}$       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

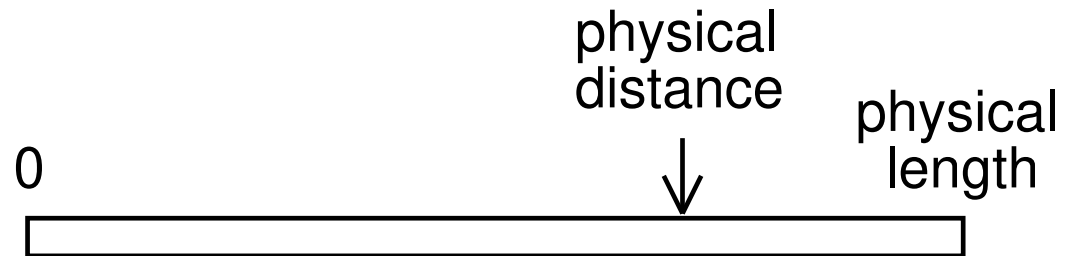
# Range Variables

Name	Meaning	Units
diam	diameter	[ $\mu\text{m}$ ]
cm	specific membrane capacitance	[ $\mu\text{f}/\text{cm}^2$ ]
g_pas (hoc) pas . g (Python)	specific conductance of the pas mechanism	[siemens/ $\text{cm}^2$ ]
v	membrane potential	[mV]

range

normalized position along the length of a section

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



## Syntax:

*secname(range).rangevar*

Translation: "in *secname*

at the location corresponding to *range*

access the value of *rangevar*"

## Examples:

```
# v at middle of dend
```

```
dend(0.5).v # shortcut: dend.v
```

```
# at each point in dend
```

```
# where v is calculated
```

```
# print range, anat distance, and v
```

```
for seg in dend.allseg():
```


```
    print(seg.x, seg.x*dend.L, dend(seg.x).v)
```

nseg

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

nseg=1 

nseg=2 

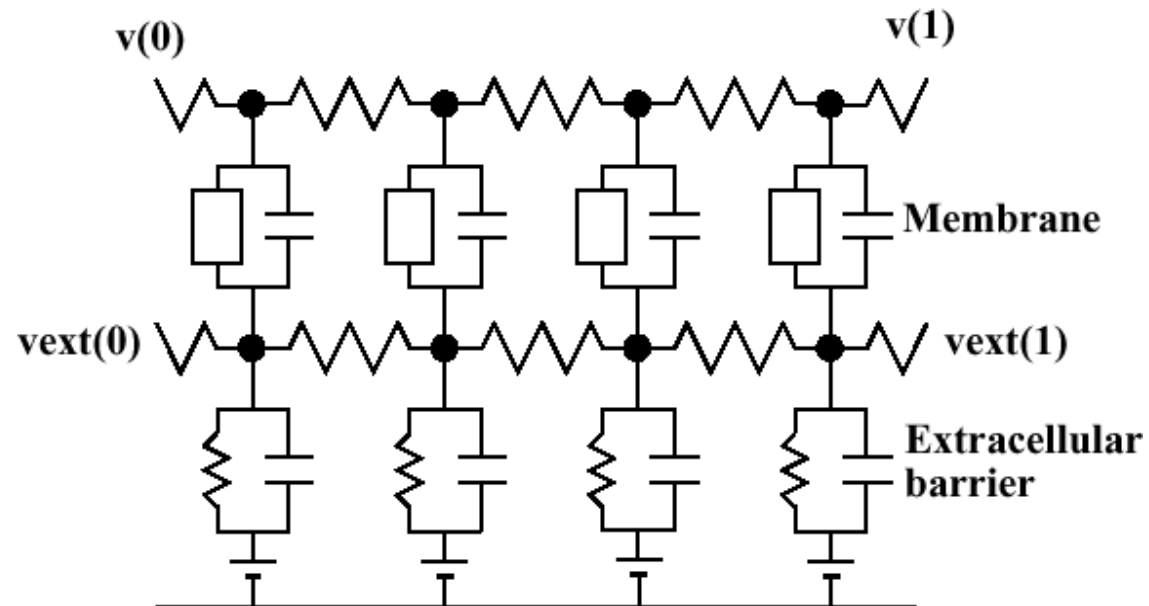
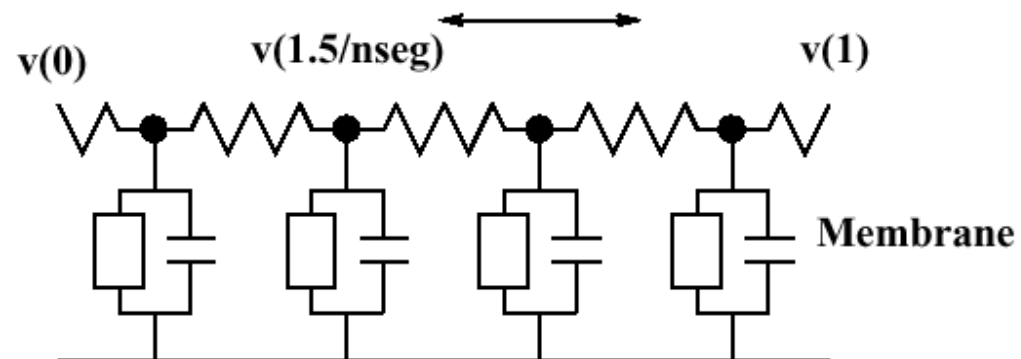
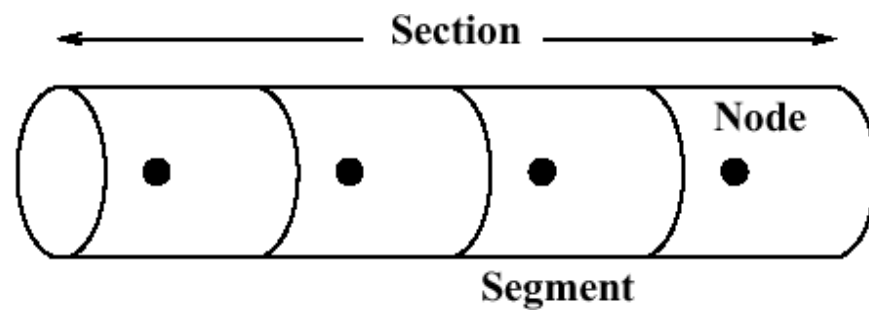
nseg=3 

Example: `axon.nseg = 3`

To test spatial resolution

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

and repeat the simulation





## Physical System



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

## Conceptual Model

### Hodgkin-Huxley cable equations

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

$$\frac{dm}{dt} = -\alpha_m m + \beta_m (1 - m) \quad \alpha_m = \frac{0.1(V + 40)}{1 - e^{-0.1(V + 40)}} \quad \beta_m = 4e^{-(V + 65)/18}$$

$$\frac{dh}{dt} = -\alpha_h h + \beta_h (1 - h) \quad \alpha_h = 0.07e^{-0.05(V + 65)} \quad \beta_h = \frac{1}{1 + e^{-0.1(V + 35)}}$$

$$\frac{dn}{dt} = -\alpha_n n + \beta_n (1 - n) \quad \alpha_n = \frac{0.01(V + 55)}{1 - e^{-0.1(V + 55)}} \quad \beta_n = 0.125e^{-(V + 65)/80}$$

# Conceptual Model

## Hodgkin-Huxley cable equations

$$\frac{D}{4R_a} \frac{\partial^2 V}{\partial x^2} = C_m \frac{\partial V}{\partial t}$$

$$+ \bar{g} m^3 h \cdot (V - E_{na}) + \bar{g}_k n^4 \cdot (V - E_k) + g_l \cdot (V - E_l)$$

$$\frac{dm}{dt} = -\alpha_m m + \beta_m (1-m) \quad \alpha_m = \frac{0.1(V+40)}{1 - e^{-0.1(V+40)}} \quad \beta_m = 4e^{-(V+65)/18}$$

$$\frac{dh}{dt} = -\alpha_h h + \beta_h (1-h) \quad \alpha_h = 0.07e^{-0.05(V+65)} \quad \beta_h = \frac{1}{1 + e^{-0.1(V+35)}}$$

$$\frac{dn}{dt} = -\alpha_n n + \beta_n (1-n) \quad \alpha_n = \frac{0.01(V+55)}{1 - e^{-0.1(V+55)}} \quad \beta_n = 0.125e^{-(V+65)/80}$$

## Computational implementation

### Python for NEURON

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

```
axon.L = 2.0e4
```

```
axon.diam = 100.0
```

```
axon.nseg = 43
```

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

# Example: Squid Axon

For this exercise create a new folder called axon.

Use a CellBuilder to create a model with a single section called axon that has these properties:

**L** 2e4 um

**diam** 100 um

**nseg** 43

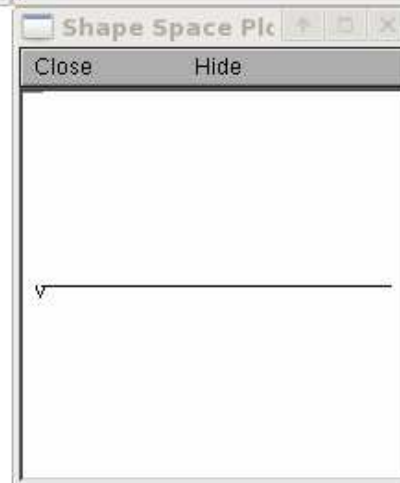
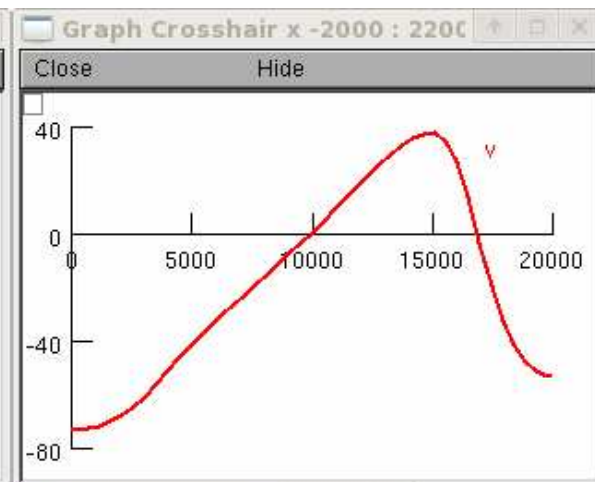
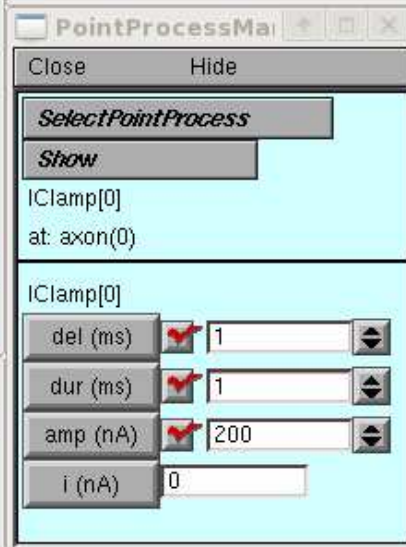
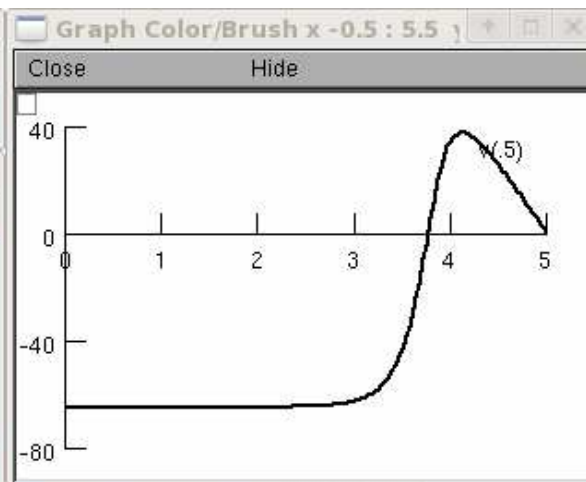
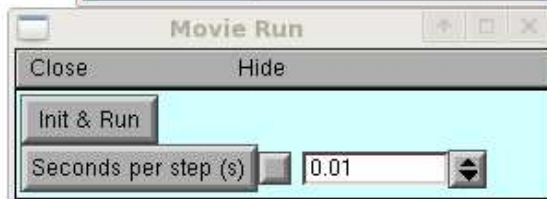
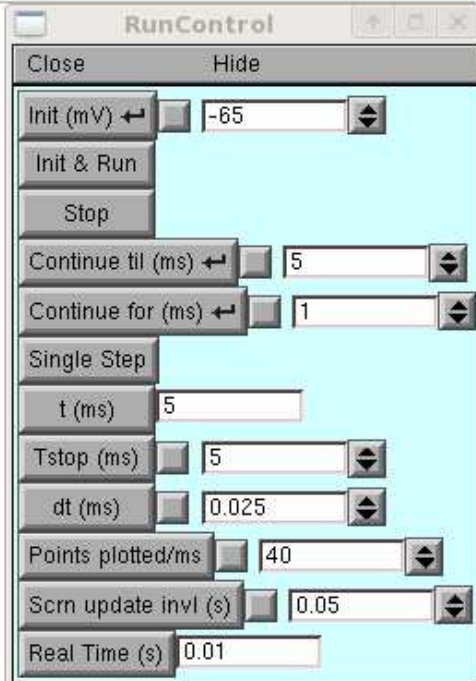
membrane has the **hh** mechanism

Save configured CellBuilder to a file called **hhaxon.ses**

# Squid Axon *continued*

Use the GUI to build an interface that includes:

- a RunControl panel to launch simulations
- a PointProcessManager configured as an IClamp attached to the 0 end of the axon
- a voltage axis graph that shows membrane potential at axon(0.5)
- a space plot that plots  $v$  vs. distance along the axon
- a Movie Run tool to launch simulations that show a smooth evolution of the space plot over time



## Squid Axon *continued*

Use a current pulse of 0.1 ms duration to trigger a spike that starts at the 0 end of the axon.

Adjust the pulse amplitude to find the spike threshold to two place precision.

Set the pulse amplitude to  $2 \times \text{threshold}$ . Measure spike amplitude and half-width at axon(0.5).

Cut sodium channel density in half. What happens to spike amplitude and half-width? Repeat until **gnabar** is so low that you don't get a spike. What is the smallest gnabar needed to produce a spike?

# Squid Axon *homework 1*

Use a text editor to create a file called axonrig.py

This file should contain the following commands:

```
from neuron import h, gui  
load_file('hhaxon.ses')
```

Save the file and exit the text editor.

At the system prompt execute the command

```
python3 -i axonrig.py
```

Your model axon and its user interface should be ready for you to use.

# Squid Axon *homework 2*

At the system prompt execute the command  
`python3 -i axonrig.py`

Attach a second IClamp to the 1 end of the axon.

Set both IClamps to deliver a 2\*spike threshold stimulus at 1 ms and run a simulation.

What happens, and why?



# Squid Axon *homework 2*

The spikes collide and annihilate each other. Why?

Hints:

Examine space plots of gna\_hh and gk\_hh at 0.1 ms intervals. Where is gk\_hh activated?

Where is gna\_hh inactivated? (check space plots of m\_hh and h\_hh)

# Squid Axon *homework 3*

At the system prompt execute the command  
`python3 -i axonrig.py`

Apply a 2 x threshold stimulus  
(for this model cell, 1720 nA x 0.1 ms)

Use the CellBuilder to change nseg to 15  
and run a simulation.

What happens and why?

Restore the original value of nseg (43),  
and determine conduction velocity  
over the middle half of the axon.

# Squid Axon *homework 4*

At the system prompt execute the command  
`python3 -i axonrig.py`

Change `IClamp.dur` to 10 ms and adjust amp to force `axon(0).v` to -80 mV. What happens after the current stops? Why?

# Squid Axon *homework 4*

After the end of the hyperpolarizing current,  $v(0)$  rebounds above spike threshold.

To see why, make two graphs:  
plots of  $m_{hh}$ ,  $h_{hh}$ , and  $n_{hh}$  at  $axon(0)$  vs.  $t$ ,  
and  
plots of  $gna_{hh}$  and  $gk_{hh}$  at  $axon(0)$  vs.  $t$ .

Read about "anode break excitation" sometime.

Anode break excitation in desheathed frog nerve.  
Frankenhaeuser B, Widen L. J Physiol. 131:243-7, 1956.  
doi: 10.1113/jphysiol.1956.sp005459. PMID 13296060.