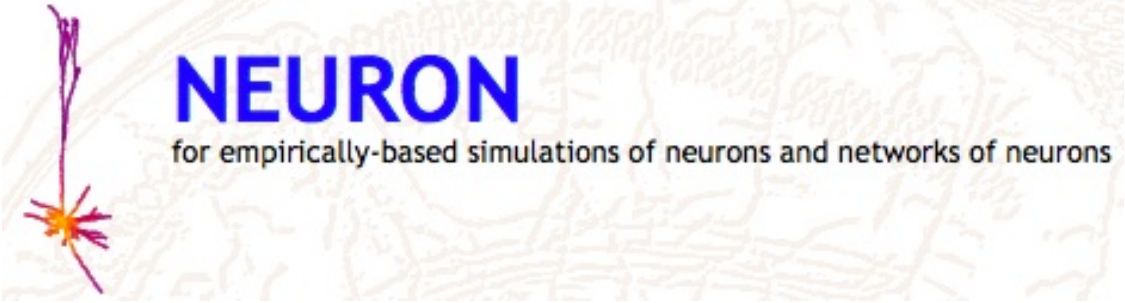
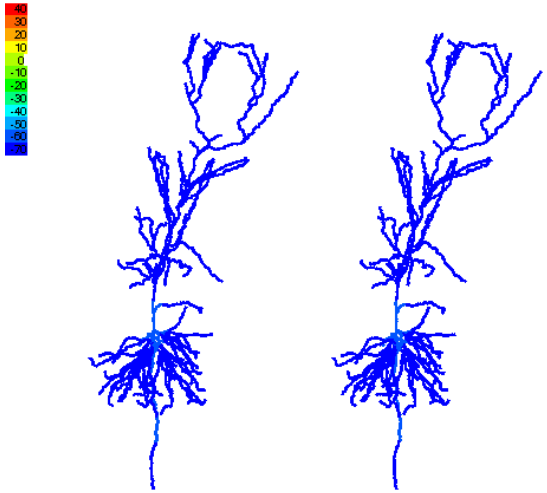
LASCON 2018

Tutorial 3
NEURON I

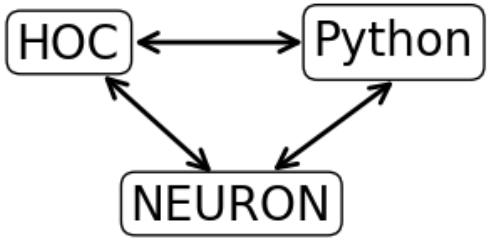




Instructors: Arnd Roth and Salvador Dura-Bernal

Interface language: Python

- One of the cleanest languages (cf. `>>> import this`)
- Advantages:
  - Easy to learn
  - Writing portable, readable code
  - Interfacing with 100s of other packages (NumPy, SciPy, Matplotlib, PyNN, etc.)
  - Access to all Neuron objects/functions via module (from neuron import h)



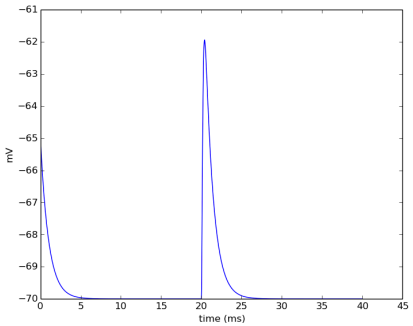
First steps

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

NEURON + Python Basics

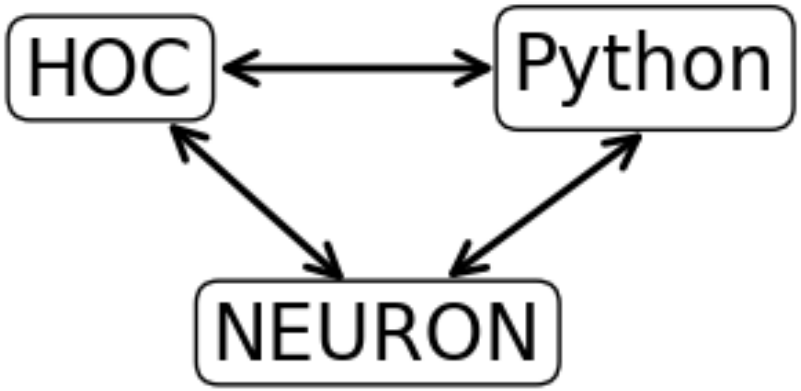
The objectives of this part of the tutorial are to get familiar with basic operations of NEURON using Python. In this worksheet we will:

- Create a passive cell membrane in NEURON.
- Create a synaptic stimulus onto the neuron.
- Modify parameters of the membrane and stimulus.
- Visualize results with matplotlib.



Step 1: Import the NEURON module

- Any code that is not part of Python's *Built-in Functions* must be imported.
- The Python interface to NEURON works through the "h" module.
- The h module permits a direct interface to NEURON as well as to NEURON's other interpreter language, hoc.



Step 1: Import the NEURON module

We begin by loading NEURON's h module and its graphical user interface:

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

The results of evaluating this code in Python should look something like the following output:

```
NEURON -- VERSION 7.5 master (0388d94) 2017-08-09
Duke, Yale, and the BlueBrain Project -- Copyright 1984-2016
See http://neuron.yale.edu/neuron/credits
```

Step 2: Create a cell

We create a simple model cell as a NEURON *Section*. Evaluate the line:

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

There is no output, so how can we tell we successfully created a section?

Note 1: Checking cell exists

NEURON's psection() (short for "print section") function can provide a lot of detail on sections.

Let's validate that we have a soma and view some of its properties:

```
h.psection()
```

This shows the soma is a cylinder with:

length 100 um,
diameter 500 um,
axial resistivity 35.4 ohm*cm,
specific membrance capacitance 1 mu F/cm^2

Note 1: Checking cell exists

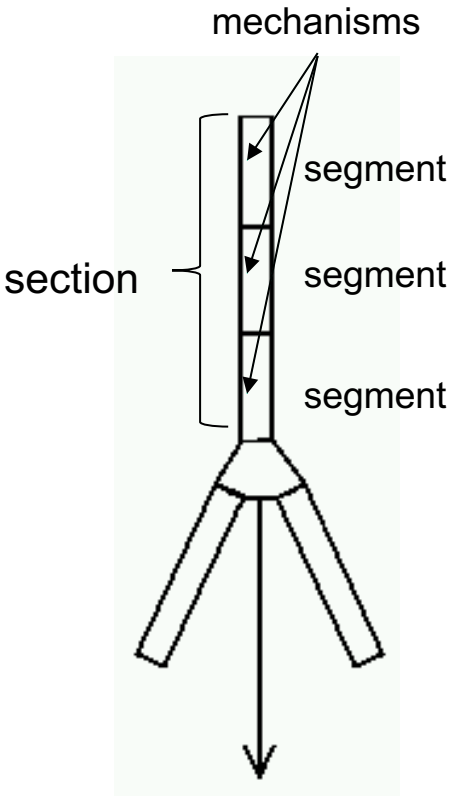
We can also probe objects with Python's built-in dir() function. Let's see what it says about soma:

```
dir(soma)
```

- This tells us all of the Python methods and variables associated with the object.
- Those starting with '__' are reserved by Python
- To see all of the functions available to the NEURON module h, try calling **`dir(h)`**.

Note 2: Sections and Segments

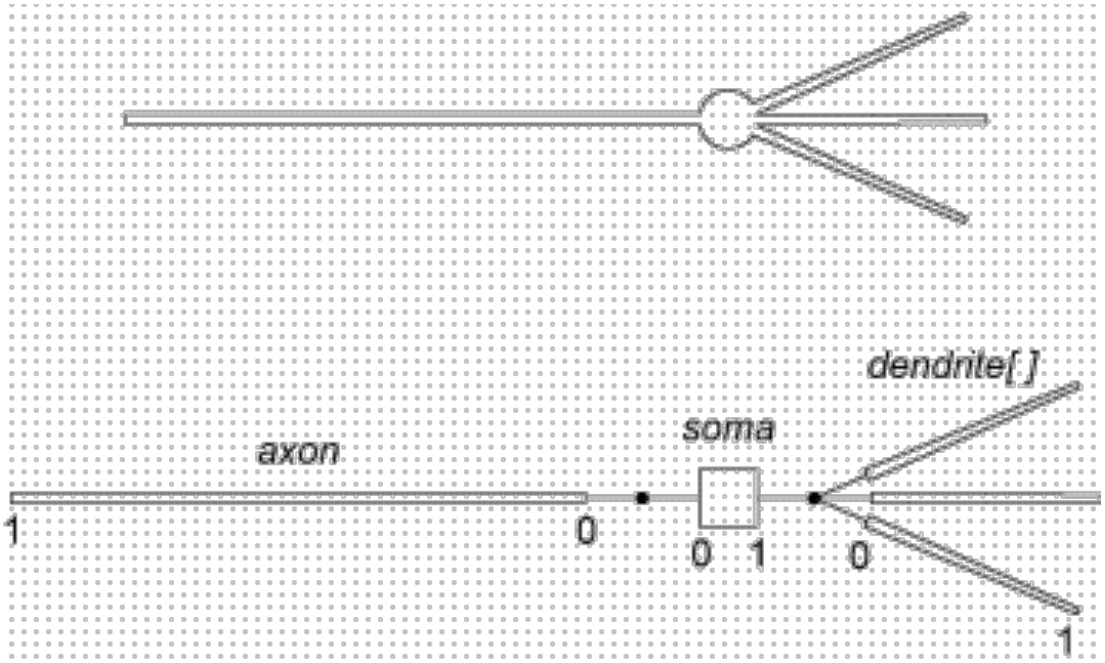
- A NEURON Section is considered a piece of cylindrical cable.
- To increase spatial resolution, you can **divide** the cable into a number of **segments** of **equal length** where voltage is calculated separately
- The **number of segments** within a section is given by the variable, `nseg`
- Do **not confuse** sections with segments!



Note 2: Sections and Segments

- To **access** a part of the section, specify a value between 0 and 1, where 0 is typically the end **closest** to the soma and 1 is the **distal** end.

- Because nseg divides the cable into equal-length parts, use an **odd number** so that to address the middle of the cable, (0.5), gives the middle segment.



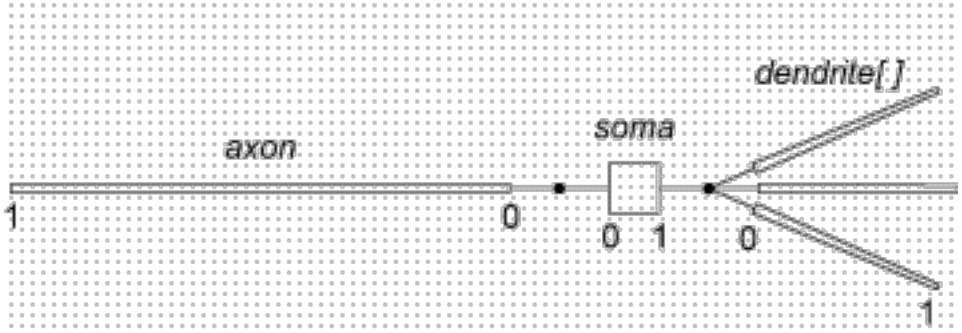
Note 2: Sections and Segments

To summarize, we access sections by their name and segments by some location on the section.

- Section: **`section`**
- Segment: **`section(loc)`**

Using the Python type() function can tell us what a variable is:

```
print "type(soma) =", type(soma)
print "type(soma(0.5)) =", type(soma(0.5))
```



Step 3: Add dendrite and connect

Create a dendritic section ‘dend’ and connect it to the ‘1’ end of the soma.

```
dend = h.Section(name='dend')
dend.connect(soma(1))
```

Let’s check the connection.

```
h.psection(dend)
```

Let’s further confirm with NEURON’s topology() function.

```
h.topology()
```

Both of these approaches show that **`dend[0]`** is connected to **`soma[1]`**.

Step 4: Set geometry

Let's set the spatial properties of the cell using a "stylized" geometry. Later we will explore setting 3D points explicitly.

```
# Surface area of cylinder is 2*pi*r*h (sealed ends are implicit).
# Here we make a square cylinder in that the diameter
# is equal to the height, so diam = h. ==> Area = 4*pi*r^2
# We want a soma of 500 microns squared:
# r^2 = 500/(4*pi) ==> r = 6.2078, diam = 12.6157

soma.L = soma.diam = 12.6157 # Makes a soma of 500 microns squared.
dend.L = 180 # microns

dend.diam = 1 # microns
dend.nseg = 11 # odd number of segments

print "Surface area of soma =", h.area(0.5, sec=soma)
```

Step 5: Set biophyisical variables

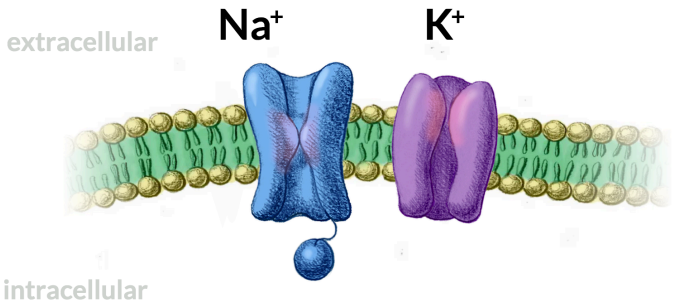
All sections include a variable to store its axial resistance (**`Ra`**) and membrance capacitance (**`cm`**).

We can use the **`h.allsec()`** method which iterates over all sections

```
for sec in h.allsec():
     sec.Ra = 100 # Axial resistance in Ohm * cm
     sec.cm = 1 # Membrane capacitance in micro Farads / cm^2
```

Note 3: Distributed mechanisms

- Distributed mechanisms modify membrane properties eg. V or g_{Na}
- They are inserted in a *Section*, and automatically distributed to all of its *Segments*
- Hodgkin-Huxley sodium, potassium and leakage channels
  - `sec.insert('hh')`
- Passive channels
  - `sec.insert('pas')`



- Other NMODL (.mod) mechanisms defined (eg. other ionic channels)

Step 6: Insert mechanisms

NEURON comes with a few built in biophysical mechanisms that can be added to a model.

pas	Passive (“leak”) channel.
hh	Hodgkin-Huxley sodium, potassium, and leakage channels.

The ‘insert’ method is used to insert mechanisms into the membrane. Let’s insert an active Hodgkin-Huxley conductance in the soma and a passive leak conductance in the dendrite:

```
# Insert active Hodgkin-Huxley current in the soma
soma.insert('hh')

# Insert passive leak current in dendrite
dend.insert('pas')
```

Note 4: Accessing segment variables

Segment variables follow the idiom:

`section(loc).var`

And for mechanisms on the segment:

`section(loc).mech.var`

or

`section(loc).var_mech`

Try:

```
print soma(0.5).pas.g
print soma(0.5).g_pas

mech = soma(0.5).pas
print dir(mech)
print mech.g
```

Note 4: Accessing segment variables

To access or set the variables of ALL segments in a section you can use:

`section.var_mech`

Try:

```
dend.g_pas = 0.001

print dend(0.1).pas.g
print dend(0.9).pas.g
```

Step 7: Set mechanism variables

Lets set the variables of the soma **hh** and dend **pas** mechanisms:

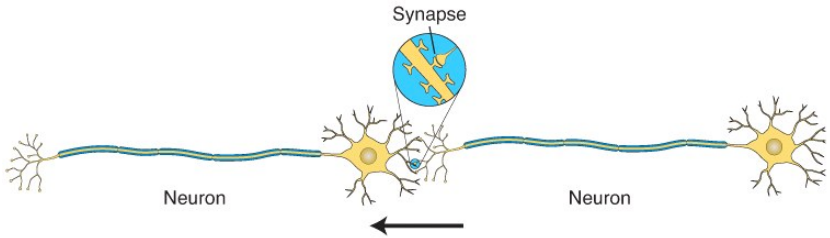
```
soma.gnabar_hh = 0.12 # Sodium conductance in S/cm2
soma.gkbar_hh = 0.036 # Potassium conductance in S/cm2
soma.gl_hh = 0.0003 # Leak conductance in S/cm2
soma.el_hh = -54.3 # Reversal potential in mV # Insert passive current in the dendrite

dend.g_pas = 0.001 # Passive conductance in S/cm2
dend.e_pas = -65 # Leak reversal potential mV
```

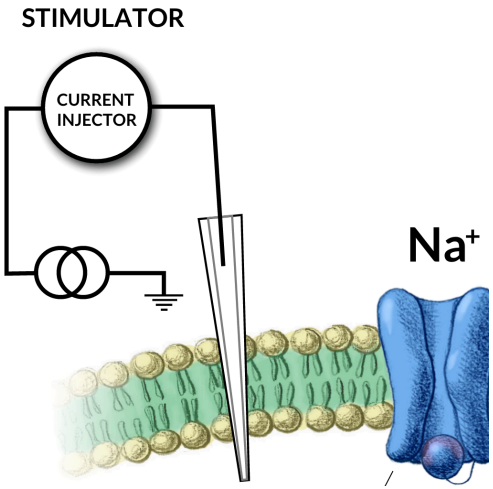
Note 5: Point Processes

- Point processes are sources of current in specific segment

- Synapses
  - syn = h.AlphaSynapse(soma(0.5))
  - syn = h.ExpSyn(dend(0.8))



- Current Clamp
  - stim = h.IClamp(soma(0.5))



- Artificial Cells (special type of point process)
  - ns = h.NetStim()
  - cell = h.IntFire1()
  - cell = h.IntFire2()

Step 8: Insert current clamp

Let's insert an current clamp (**`IClamp`**) object onto the end of the dendrite to induce some membrane dynamics.

```
stim = h.IClamp(dend(1.0))
```

When making a new PointProcess, you pass the segment to which it will bind.

Again, with dir() function, we can validate that **`stim`** is an object and contains some useful parameters. Let's set some of those parameters.

```
dir(stim)

stim.amp = 0.1  # input current in nA
stim.delay = 20 # turn on after this time in ms
stim.dur = 3    # duration in ms
```

Step 9: Set up recording variables

We need to set up variables we wish to record from the simulation.

We will store them using **`h.Vector()`**, a NEURON class used to store and operate on 1D arrays of numbers.

We will record the soma membrane potential, which is **`soma(0.5).v`** and dendrite membrane potential at 1.0: **`dend(1.0).v`**

But note that references to variables are available as **`_ref_variable`**, so to record we need:

```
v_vec_soma = h.Vector() # Membrane potential vector
v_vec_dend = h.Vector() # Membrane potential vector
t_vec = h.Vector() # Time stamp vector

v_vec_soma.record(soma(0.5)._ref_v)
v_vec_dend.record(dend(1.0)._ref_v)
t_vec.record(h._ref_t)
```

Step 10: Run the simulation

To run the simulation, we execute the following lines.

```
h.tstop = 40.0
h.run()
```

Note: If we had not included **`gui`** in the list of things to import, we would have also had to execute the following code which defines the run() func:

```
h.load_file('stdrun.hoc')
```

Step 11: Plot the results

We utilize the **`pyplot`** module from the matplotlib Python package to visualize the output.

```
from matplotlib import pyplot as plt

plt.figure(figsize=(8,4)) # Default figsize is (8,6)
plt.plot(t_vec, v_vec)
plt.plot(t_vec, v_vec_soma, 'b', label='soma')
plt.plot(t_vec, v_vec_dend, 'r', label='dend')
plt.xlabel('time (ms)')
plt.ylabel('mV')
plt.show()
```