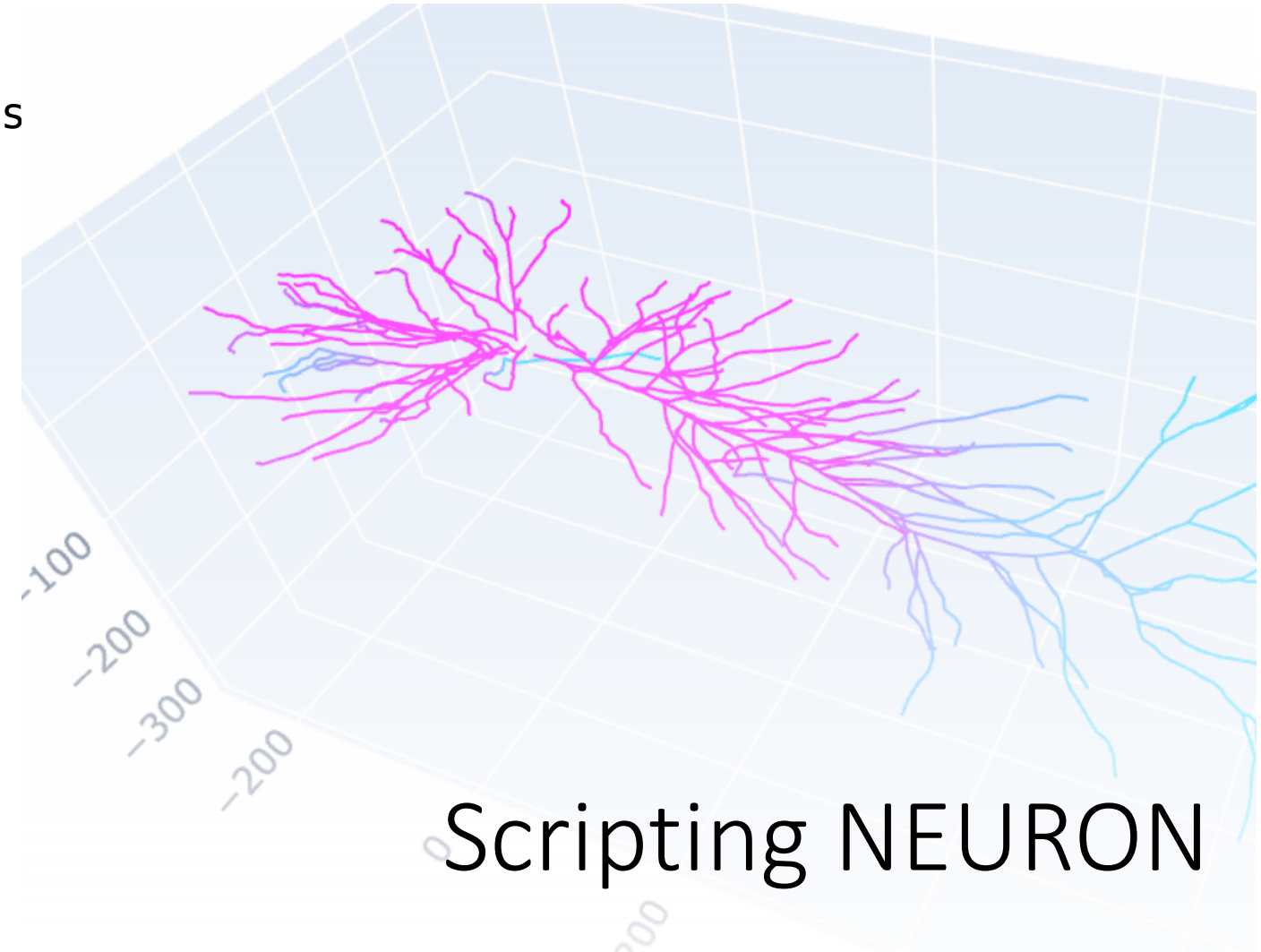
```python
from neuron import h
from neuron.units import mV, ms
from matplotlib import cm
import plotly
h.load_file('stdrun.hoc')

h.load_file('c91662.ses')
h.hh.insert(h.allsec())

ic = h.IClamp(h.soma(0.5))
ic.delay = 1 * ms
ic.dur = 1 * ms
ic.amp = 10

h.finitialize(-65 * mV)
h.continuerun(2 * ms)

ps = h.PlotShape(False)
ps.variable('v')
ps.plot(plotly, cmap=cm.cool).show()
```




# Scripting NEURON

Robert A. McDougal

23 November 2020

# The slides are available at…

neuron.yale.edu/ftp/neuron/neuron2020/NEURON2020day3a.pdf

# What is a script?

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

- In NEURON, scripts can:
  - set-up a module
  - define and perform an experimental protocol
  - record data
  - save and load data
  - and more …

# 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 **re-collecting data** after change in experimental protocol.
- Provides a complete, **reproducible** version of the experimental protocol.

www.neuron.yale.edu/neuron/

NEURON

NEWS DOWNLOAD DOCUMENTATION COURSES PUBLICATIONS RESOURCES ABOUT US

FORUM MODELDB PROGRAMMER'S REFERENCE

# Welcome to the community of NEURON users and developers!

**The NEURON simulation environment is used in laboratories and classrooms 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 by helping to organize future meetings of the NEURON Users Group, and 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.**

## DOWNLOAD 7.8.1

Download macOS installer

Quickstart Guide

All standard versions

Alpha versions

## THE NEURON FORUM

### The Neuron Forum

- NEURON Installation
- Making and using models
- Programming NEURON with Python
- NEURON in education

## LATEST NEWS

| | |
|---|---|
| 14 August 2020 | NEURON 7.8.1 released |
| 18 June 2020 | Monthly developer's meeting: June 19, 2020 |

neuron.yale.edu

neuronsimulator.github.io/nrn/py_doc/visualization/graph.html





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:

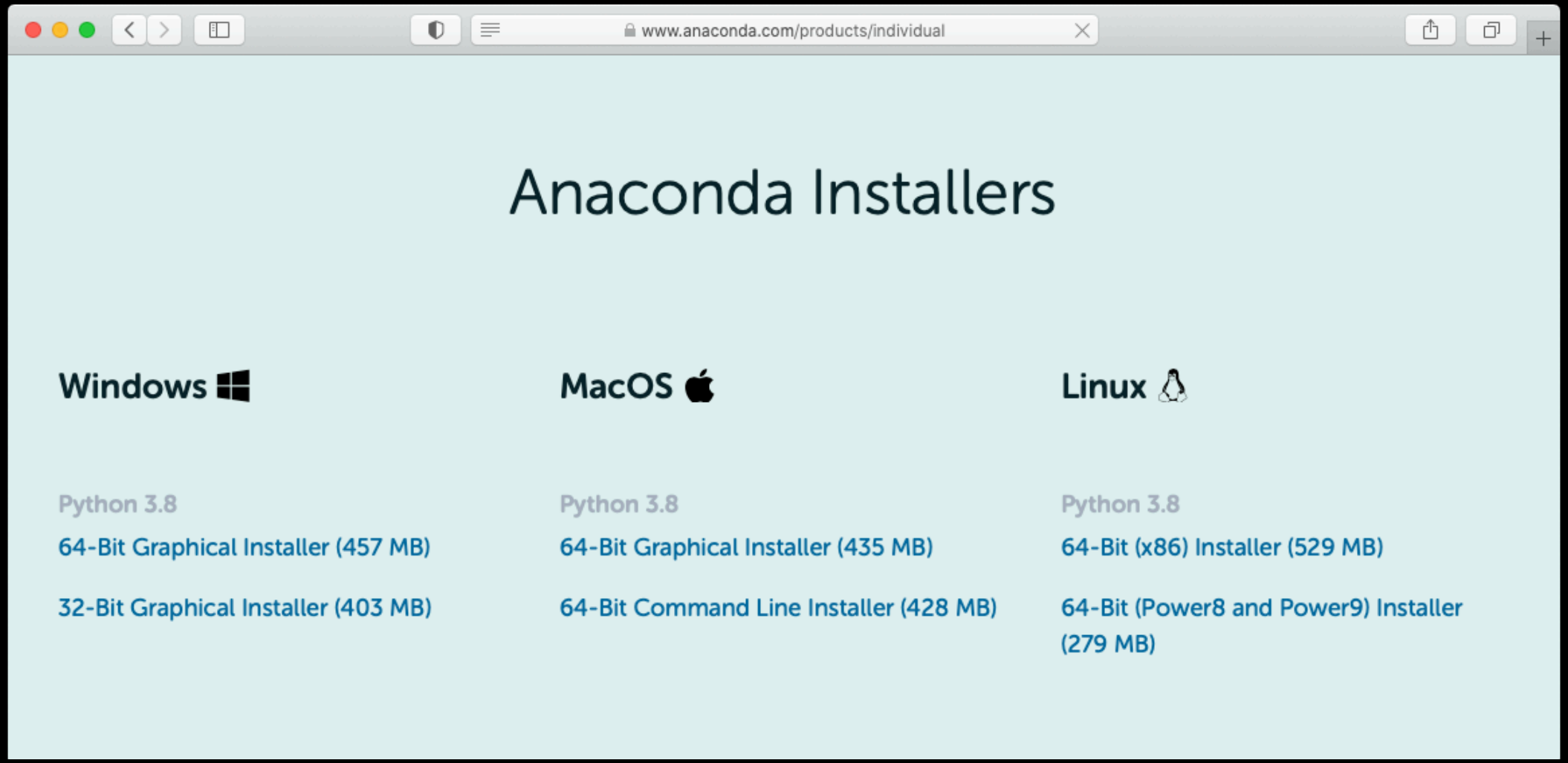
An instance of the Graph class manages a window on which x-y plots can be drawn by calling various member functions. The first form immediately maps the window to the screen. With a 0 argument the window is not mapped but can be sized and placed with the `view()` function.

Example:

The most basic interpreter prototype for producing a plot follows:

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

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.




There are many Python distributions. Any should work, but many people prefer Anaconda as it comes with a large set of useful libraries.

spike-demo.py

spike-demo.py

sers > ramcdougal > Dropbox > notebook > 20201122 > spike-demo.py > ...

```python
from neuron import h
from neuron.units import ms, mV, µm
import matplotlib.pyplot as plt
h.load_file("stdrun.hoc")

soma = h.Section(name='soma')
soma.L = soma.diam = 10 * µm
soma.insert(h.hh)

ic = h.IClamp(soma(0.5))
ic.delay = 2 * ms
ic.dur = 0.1 * ms
ic.amp = 1

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

h.finitialize(-65 * mV)
h.continuerun(10 * ms)

plt.plot(t, v)
plt.show()
```

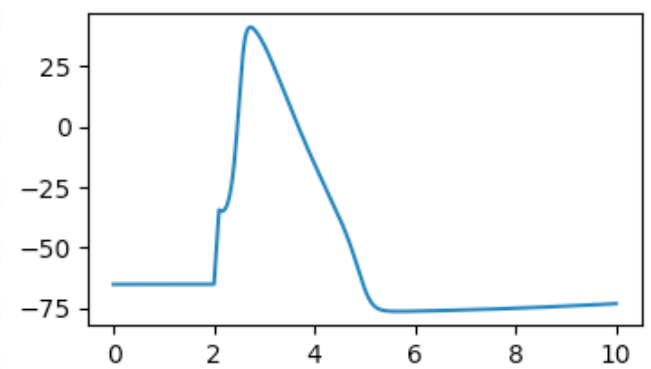
master* Python 3.8.3 64-bit ('base': conda) 1 0

20201122 — python spike-demo.py — 69×30

~/Dropbox/notebook/20201122 — python spike-demo.py

```
Last login: Sun Nov 22 20:44:33 on ttys004
(base) ramcdougal@Roberts-MacBook-Pro 20201122 % python
Python 3.8.3 (default, Jul  2 2020, 11:26:31)
[Clang 10.0.0 ] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information
.
>>> from neuron import h
>>> h.celsius
6.3
>>> exit()
(base) ramcdougal@Roberts-MacBook-Pro 20201122 % python spike-demo.py
```

Figure 1




x=6.11945 y=-

localhost:8888/notebooks/spike-demo.ipynb

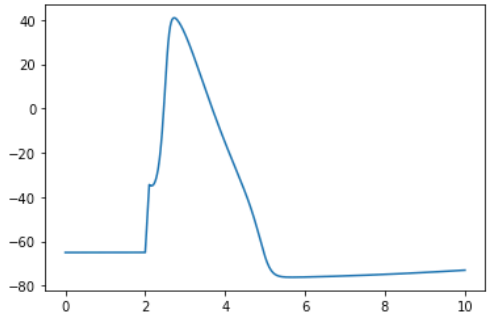
jupyter spike-demo (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

In [1]:
```
from neuron import h
from neuron.units import ms, mV, µm
import matplotlib.pyplot as plt
h.load_file("stdrun.hoc")

soma = h.Section(name='soma')
soma.L = soma.diam = 10 * µm
soma.insert(h.hh)

ic = h.IClamp(soma(0.5))
ic.delay = 2 * ms
ic.dur = 0.1 * ms
ic.amp = 1

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

h.finitialize(-65 * mV)
h.continuerun(10 * ms)

plt.plot(t, v)
plt.show()
```




In [ ]:

colab.research.google.com/drive/1t91Ebz3oy

spike-demo.ipynb

File Edit View Insert Runtime Tools Help All chang...

Comment Share

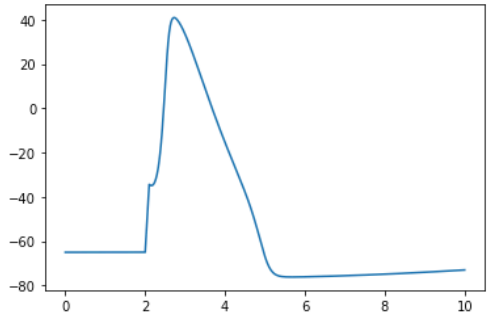
+ Code + Text

RAM Disk Editing

[11]
```
!pip install neuron
```

```
Requirement already satisfied: neuron in /usr/local/lib/python3.6/dist-packages (
Requirement already satisfied: numpy>=1.9.3 in /usr/local/lib/python3.6/dist-pack
```

```
from neuron import h
from neuron.units import ms, mV, µm
import matplotlib.pyplot as plt
h.load_file("stdrun.hoc")

soma = h.Section(name='soma')
soma.L = soma.diam = 10 * µm
soma.insert(h.hh)

ic = h.IClamp(soma(0.5))
ic.delay = 2 * ms
ic.dur = 0.1 * ms
ic.amp = 1

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

h.finitialize(-65 * mV)
h.continuerun(10 * ms)

plt.plot(t, v)
plt.show()
```



# Introduction to Python

# Displaying results: the print function

```
[1] print("NEURON is a great tool for simulation.")
```

```
NEURON is a great tool for simulation.
```

```
[2] print(5 * (3 + 2))
```

```
25
```

```
[6] print(soma.diam)
```

```
10.0
```

# Variables

- Give things a name to access them later:

```
diameter = 4
print("The diameter is", diameter)
print("The square of the diameter is", diameter ** 2)
```

```
The diameter is 4
The square of the diameter is 16
```

# Lists and for loops

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

```
cell_parts = ["soma", "apical", "basal", "axon"]
for part in cell_parts:
    print(part)
```

- Items in a list can be accessed directly using the [] notation. Note: lists start at position 0.

```
print(cell_parts[2])
```

```
basal
```

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

```
print("brain" in cell_parts)
```

```
False
```

# Dictionaries

- If there is no natural order, specify your own key-value pairs:

```
diameters = {"soma": 10, "axon": 2, "apical": 5}
print(diameters["apical"])
```

```
5
```

- Loop over keys and values using `.items()`:

```
for name, diam in diameters.items():
    print("The diameter of", name, "is", diam, "microns")
```

```
The diameter of soma is 10 microns
The diameter of axon is 2 microns
The diameter of apical is 5 microns
```

# Functions

- If a calculation is used more than once, give it a name via `def` and refer to it by the name.
- If there is a complicated self-contained calculation, give it a name.
- Return the result of the calculation with the `return` keyword.

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

```
vol1 = volume_of_cylinder(5, 20)
apical_vol = volume_of_cylinder(apical.diam, apical.L)
```

# Libraries (aka “modules”)

- Python modules provide functions, classes, and values that your 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))
```

```
0.5000000000000001
```

- One can also load specific items from a module or give a short-hand name for the module:

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

```
import pandas as pd
```

# Other useful Python modules

- `math`
  - Basic math functions
- `numpy`
  - Advanced math functions
- `pandas`
  - Basic data science and database access
- `sklearn`
  - Machine learning
- `plotly`, `plotnine`, `matplotlib`, `mayavi`
  - Plotting

# Getting help

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

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

```
['APCount', 'AlphaSynapse', 'AtolTool', 'AtolToolItem', 'BBSaveState',
'CVode', 'DEG', 'Deck', 'E', 'ExecCommand', 'Exp2Syn', 'ExpSyn', 'FARAD
AY', 'FInitializeHandler', 'Family', 'File', 'GAMMA', 'GUIMath', 'Glyph
', 'Graph', 'HBox', 'IClamp', 'Impedance', 'Inserter', 'IntFire1', 'Int
Fire2', 'IntFire4', 'KSChan', 'KSGate', 'KSState', 'KSTrans', 'L', 'Lin
earMechanism', 'List', 'Matrix', 'MechanismStandard', 'MechanismType',
```

⋮

# Getting help

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

```
help(h.IClamp)
```

```
NEURON+Python Online Help System
================================

Syntax:
    ``stimobj = h.IClamp(section(x))``
```

⋮

# Getting help

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

The data for the course can be found in this Google Drive folder. To add it to your own Google Drive, click on the course folder at the top next to the "Shared with me" header, select "Add shortcut to drive" from the dropdown and then create the shortcut by selecting "My Drive" (or your subfolder of choice).

*Solutions for each set of exercises will be posted in the evenings after each class.*

## Monday: An Introduction to Python for Data Science

| | | | | |
|---|---|---|---|---|
| Basic calculations, variables, data types | Lecture (20m 2s) | Colab notebook | Exercises | Solutions |
| Functions, Methods, f-strings | Lecture (24m 12s) | Colab notebook | Exercises | Solutions |
| Looping (for loops) and making choices (if statements) | Lecture (30m 23s) | Colab notebook | Exercises | Solutions |
| Loading and using libraries (modules) | Lecture (9m 34s) | Slides | Exercises | Solutions |
| Loading and manipulating data with pandas | Lecture (36m 37s) | Slides | Exercises | Solutions |
| Visualizing data with ggplot | Lecture (15m 3s) | Slides | Exercises | Solutions |

## Tuesday: Data Management and Databases

This summer course is organized by the Yale Center for Medical Informatics and the Center for Biomedical Data Science

ycmi.github.io/summer-course-2020

# Basic NEURON Scripting

# Loading NEURON

- Core NEURON functionality

```
from neuron import h
```

- Units definitions

```
from neuron.units import mV, ms, um
```

You will almost always need these.

- Chemical dynamics

```
from neuron import rxd
```

- Graphics

```
from neuron import gui
```

# NEURON run control library

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

stdrun.hoc loads NEURON's "standard run" system, which provides the h.continuerun function for running a simulation until a specific time.

# 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 the result to a variable:

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

- A single Section can have multiple references to it.

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

```
True
```

- Printing a Section displays its name. Use str(section) to get the name as a string:

```
s = str(apical)
print(apical)
```

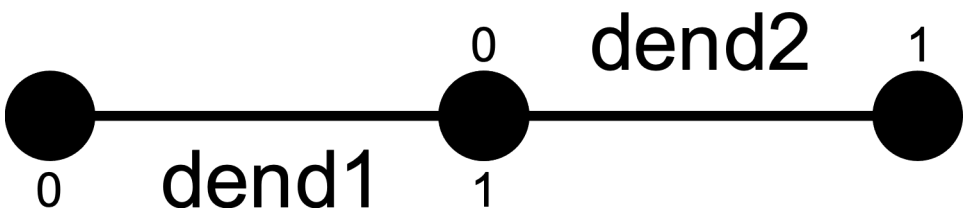
```
apical
```

# Connecting sections

To construct 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()`.

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

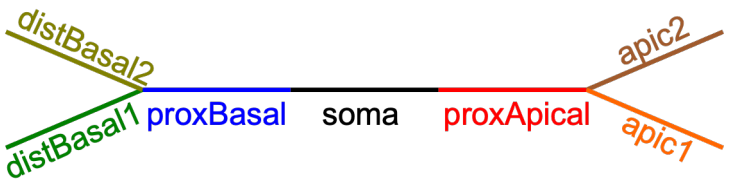
# Example

```python
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 with .L and diameter with .diam:

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

diameter may also be specified per segment

If no units are specified, NEURON assumes µm.

To specify the $(x, y, z; d)$ points a section `sec` passes through, use e.g. `sec.pt3dadd(x, y, z, d)`. The section `sec` has `sec.n3d()` 3D points; their $i^{\text{th}}$ x-coordinate is `sec.x3d(i)`. The methods `.y3d`, `.z3d`, and `.diam3d` work similarly.

# Caution: Squid

NEURON's defaults are based on the squid giant axon.

`sec.diam`: 500 µm

`sec.Ra`: 35.4 Ω cm

`h.celsius`: 6.3 C

Comingio Merculiano (1845–1915) - Jatta di Guiseppe (1896) *Cefalopodi viventi nel Golfo di Napoli (sistematica)*, Berlin: R. Friedländer & Sohn.

# Tip: define classes of cells not individual cells

- Consider the code

```
class Pyramidal:
    def __init__(self):
        self.soma = h.Section(name="soma", cell=self)
        self.soma.L = self.soma.diam = 10
```

- 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.diam)
```

```
10.0
```

# Tip: define classes of cells not individual cells

- By defining a cell in a class, once we are happy with it, we can a copy of the cell in a single line of code:

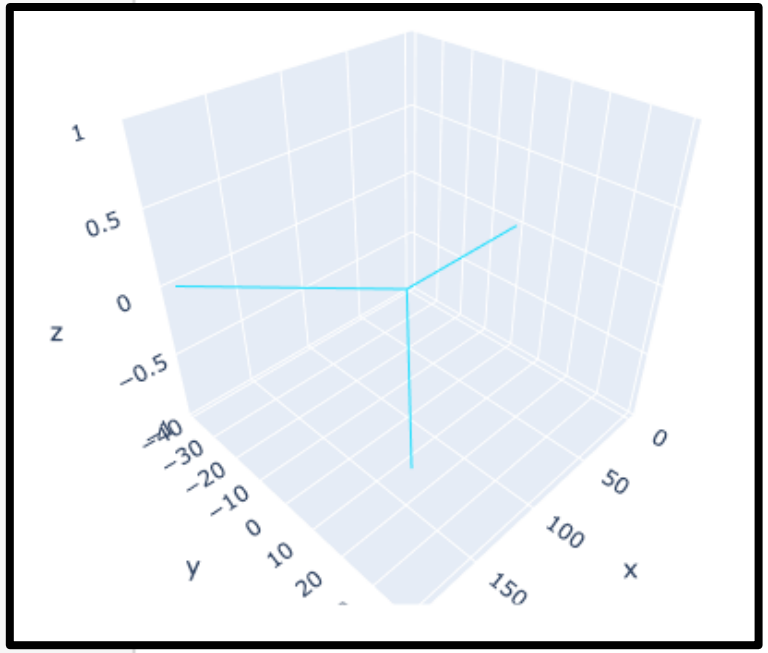
```
pyr2 = Pyramidal()
```

- Or even many copies:

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

# Viewing the morphology with h.PlotShape

```python
from neuron import h
from neuron.units import um
import plotly

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 * um
        dend1.diam = 2 * um
        dend2.diam = 2 * um

        # important: store the sections
        self.main = main; self.dend1 = dend1; self.dend2 = dend2
        self.all = main.wholetree()

my_cell = Cell()
ps = h.PlotShape(False)
ps.plot(plotly).show()
```




Passing `True` instead of `False` will plot in an InterViews window instead.

The InterViews windows can be saved as postscript using e.g.

```
ps.printfile("filename.eps")
```

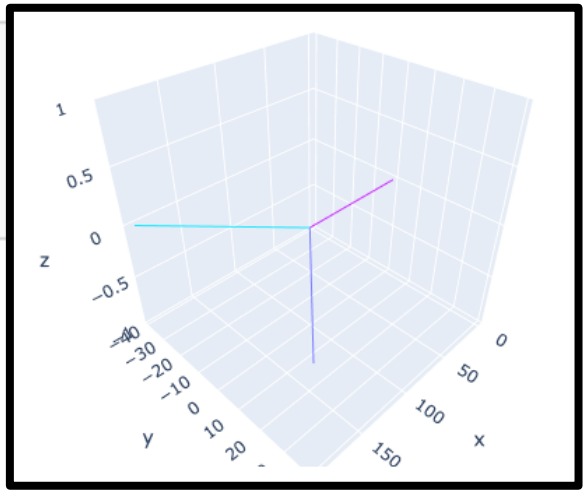
# Viewing voltage, sodium, etc…

- 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(False)
ps.variable("v")
ps.scale(-80, 80)
ps.plot(plotly).show()
```



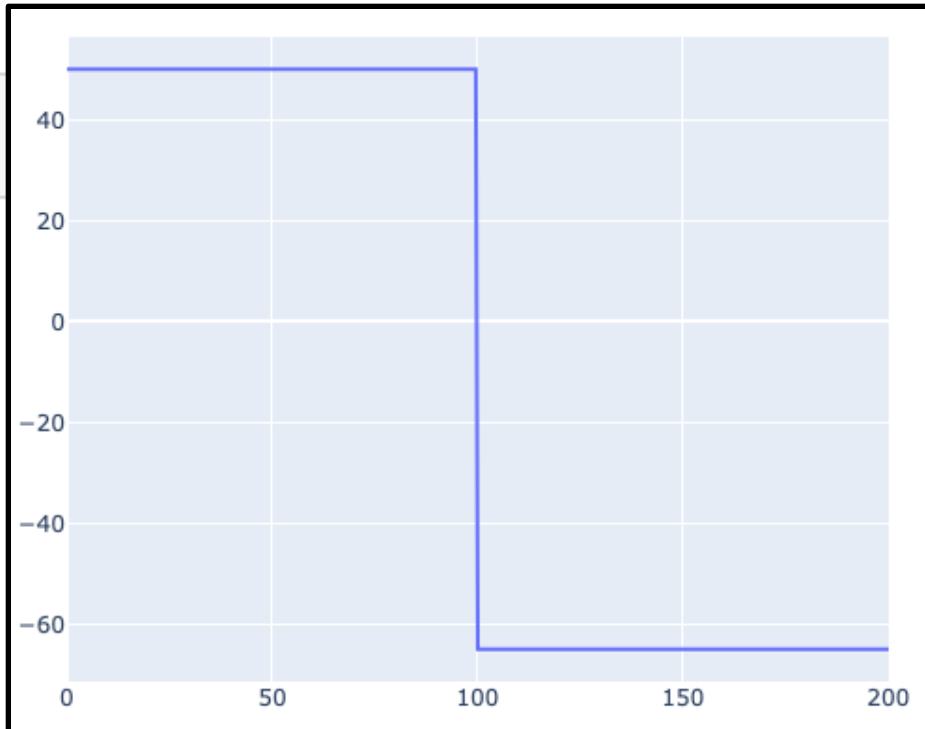
# Viewing voltage, sodium, etc…

- After increasing the spatial resolution:

```
for sec in my_cell.all: sec.nseg = 101
```

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

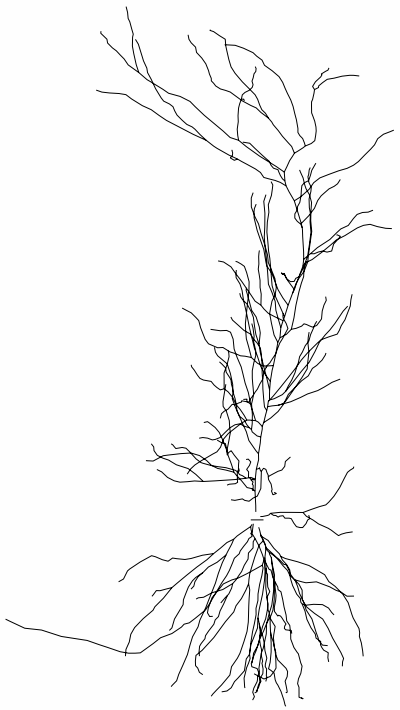
```
rvp = h.RangeVarPlot('v', my_cell.main(0), my_cell.dend2(1))
rvp.plot(plotly).show()
```



# Loading morphology from an swc file

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

```
from neuron import h
h.load_file("stdlib.hoc")
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, False)
        i3d.instantiate(self)

pyr = Pyramidal()
```



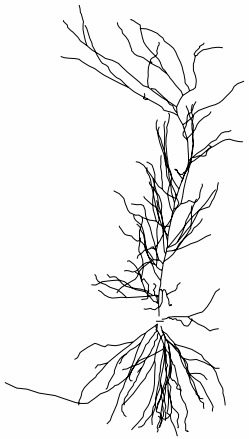
`pyr` has lists of Sections: `pyr.apic`, `.axon`, `.soma`, and `.all`.
Each Section has the appropriate `.name()` and `.cell()`

# Working with multiple cells

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

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

We then view these in a shape plot:



Where are the other 9 cells?

# Working with multiple cells

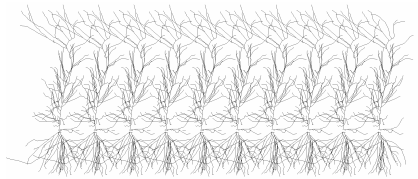
We can create a method to reposition a cell and call it from `__init__`:

```
from neuron import h
h.load_file("stdlib.hoc")
h.load_file("import3d.hoc")

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)):
            soma.pt3dchange(i, a + x, b + y, c + z, d)
```

```
    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, False)
        i3d.instantiate(self)
```

Now if we create ten while specifying offsets, the PlotShape will show all the cells separately.

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



# 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

- Insert a distributed mechanism (e.g. from a mod file) into a section or list of sections with `.insert`:

```
h.hh.insert(apical)
```

```
h.hh.insert([apical, soma, basal])
```

```
h.hh.insert(h.allsec())
```

- Mechanisms may also be inserted one-at-a-time into a single section via e.g.

```
apical.insert(h.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.*

# 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: `sec(x).MECHANISM.VARNAME` e.g.

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

- Setting variables works the same way:

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

# Setting and reading parameters

- 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 seg in apical:
    seg.hh.gkbar = 0.037
```

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

```
for sec in h.allsec():
    for seg in sec:
        seg.hh.gkbar = some_function(h.distance(seg, soma(0.5)))
```

h.allsec() is an iterable of all sections

- 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]
```

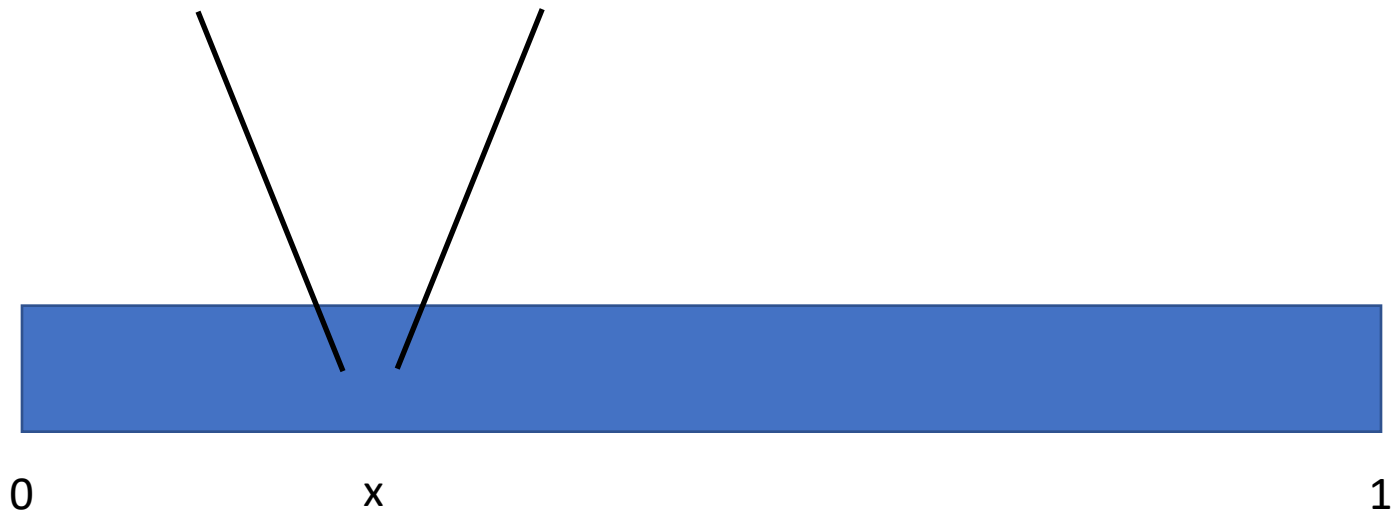
Note: looping over a Section only returns true Segments. If you want to include the voltage-only nodes at 0 and 1, iterate over, e.g. apical.allseg() instead. HOC's for (x,0) and for (x) are equivalent to looping over a section and looping over allseg, respectively.

# Recording Results

We can read the instantaneous membrane potential at a location via, e.g.

```
axon(0.5).v
```

To record this value over time, we use an `h.Vector` and pass in the pointer (prefixed with `_ref_`) to the record method.




```
v = h.Vector().record(axon(x)._ref_v)
t = h.Vector().record(h._ref_t)
```

# Running simulations: the basics

For convenience, we use a high-level simulation control functions defined in the `stdrun.hoc` library. Load this via:

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

Initialize to -65 mV:

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

Run until time 10 ms:

```
h.continuerun(10 * ms)
```

# Running simulations: the basics

For convenience, we use a high-level simulation control functions defined in the `stdrun.hoc` library. Load this via:

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

Initialize to -65 mV:

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

Advance one timestep:

```
h.fadvance()
```

# Running simulations: improving accuracy

Increase time resolution (by reducing time steps) via, e.g.

```
h.dt = 0.01 * ms
```

Enable variable step (allows error control):

```
h.CVode().active(True)
```

Set the absolute tolerance to e.g. 10−5:

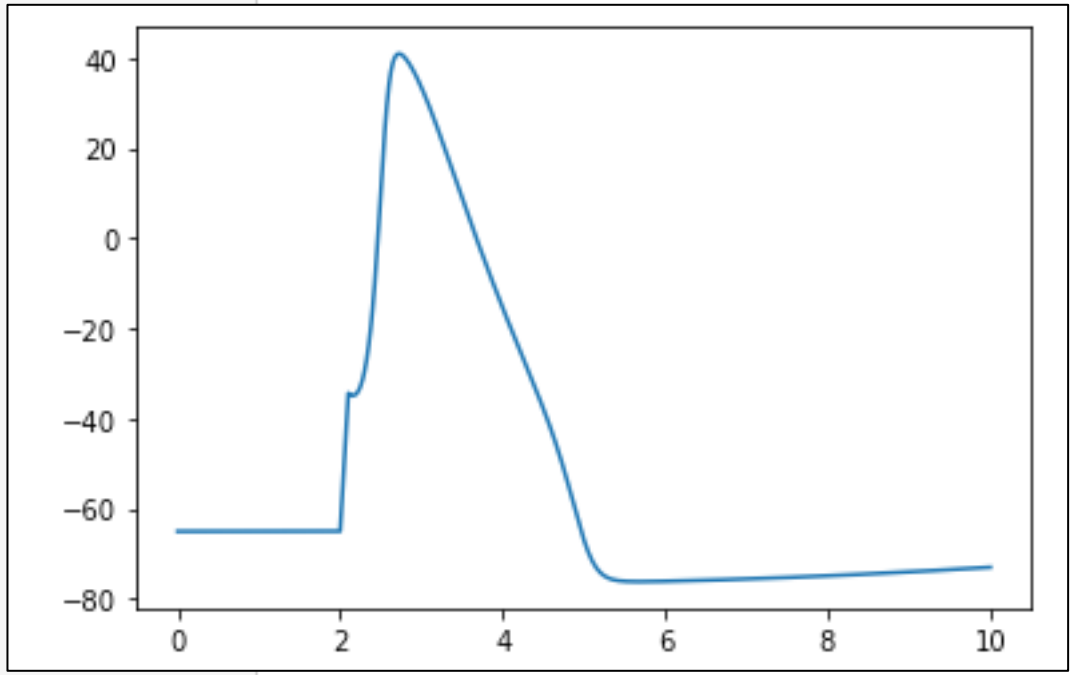
```
h.CVode().atol(1e-5)
```

Increase spatial resolution by e.g. a factor of 3 everywhere:

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

# Example: Hodgkin-Huxley

```
from neuron import h
from neuron.units import ms, mV, µm
import matplotlib.pyplot as plt
h.load_file("stdrun.hoc")

soma = h.Section(name='soma')
soma.L = soma.diam = 10 * µm
h.hh.insert(soma)

ic = h.IClamp(soma(0.5))
ic.delay = 2 * ms
ic.dur = 0.1 * ms
ic.amp = 1

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

h.finitialize(-65 * mV)
h.continuerun(10 * ms)

plt.plot(t, v)
plt.show()
```



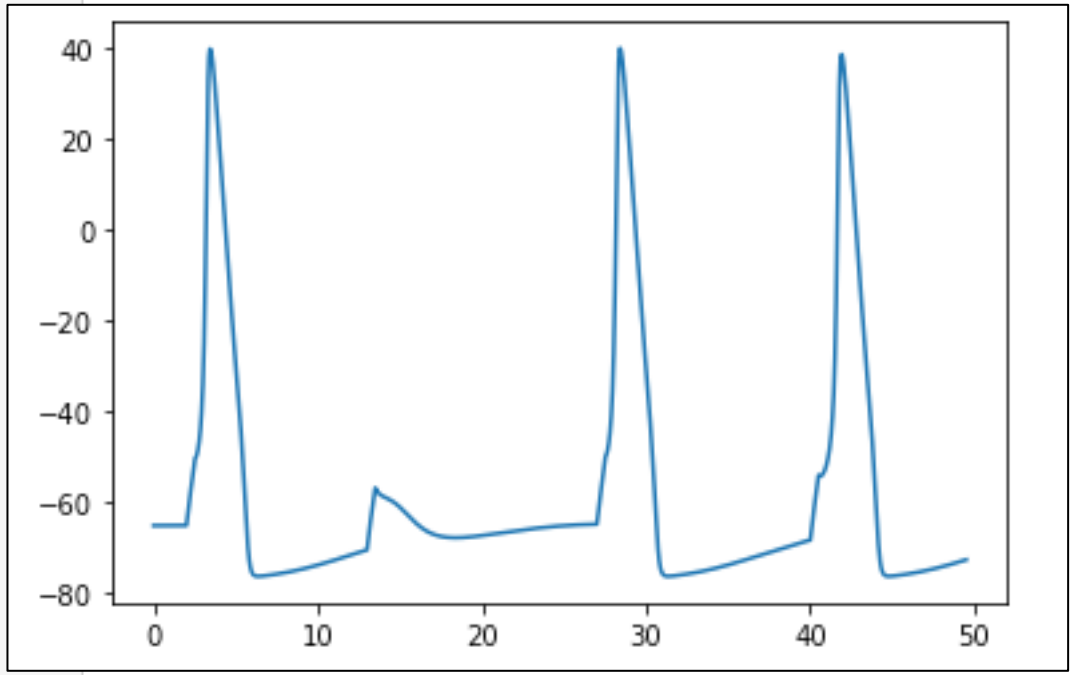
# Example: spike detection

```
from neuron import h
from neuron.units import ms, mV, µm
import matplotlib.pyplot as plt
h.load_file("stdrun.hoc")

axon = h.Section(name='axon')
h.hh.insert(axon)

iclamps = []
for input_time in [2 * ms, 13 * ms, 27 * ms, 40 * ms]:
    ic = h.IClamp(axon(0.5))
    ic.delay = input_time
    ic.dur = 0.5 * ms
    ic.amp = 50
    iclamps.append(ic)

t = h.Vector().record(h._ref_t)
v = h.Vector().record(axon(0.5)._ref_v)
nc = h.NetCon(axon(0.5)._ref_v, None, sec=axon)
spike_times = h.Vector()
nc.record(spike_times)

h.finitialize(-65 * mV)
h.continuerun(49.5 * ms)

print("spike times:", list(spike_times))
plt.plot(t, v)
plt.show()
```

Many inputs

Recording spikes

```
spike times: [3.225000000100012, 28.20000000009893, 41.70000000010092]
```



# Networks of neurons

- Suppose we have the simple model:

```
from neuron import h
from neuron.units import ms, mV

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

- and two cells:

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

# Networks of neurons

- If the first cell has a sufficient current clamp injection, we know that it will fire, but how can we get that to send a signal to another cell?
- We do this with a synapse.
- On the post-synaptic side:

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

- On the pre-synaptic side, specify a source pointer, the corresponding post-synaptic side, the transmission delay, and synaptic weight:

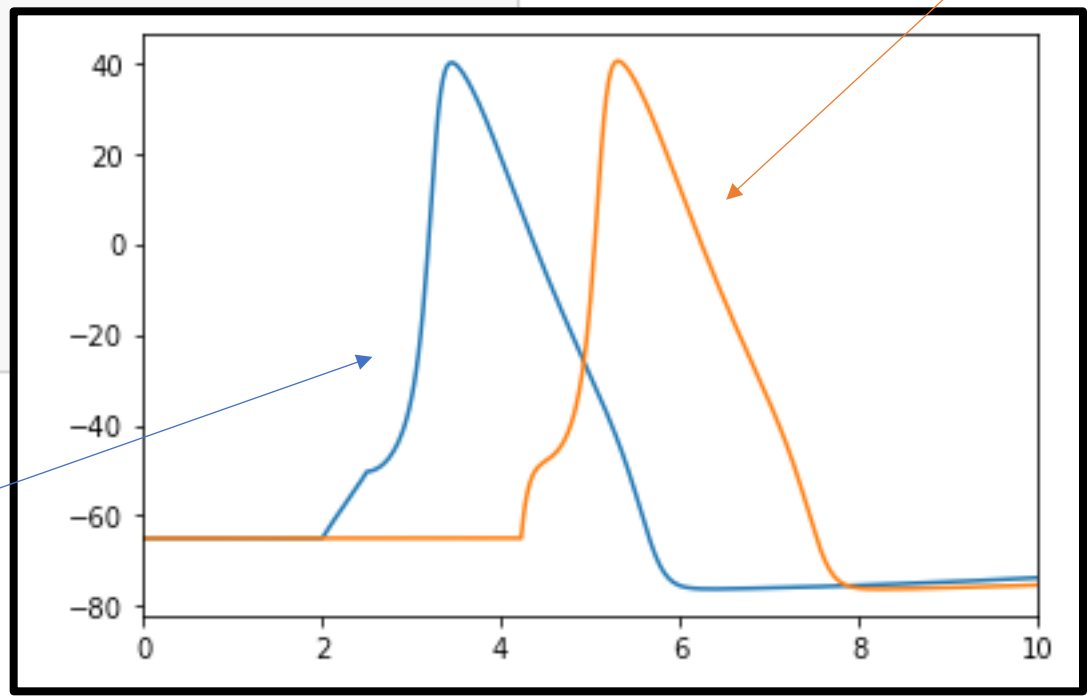
```
syn = h.NetCon(neuron1.soma(0.5)._ref_v, postsyn, sec=neuron1.soma)
syn.delay = 1
syn.weight[0] = 5
```

# Networks of neurons

Record, run, and plot as normal:

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

h.finitialize(-65 * mV)
h.continuerun(10 * ms)

plt.plot(t, v1, t, v2)
plt.xlim((0, 10))
plt.show()
```

due to the synapse




due to the iclamp
(code not shown)

# Storing and loading data with pandas

- Saving as CSV with pandas:

t and v are h.Vector instances

```
import pandas as pd
pd.DataFrame({"t": t, "v": v}).to_csv("data.csv", index=False)
```

- Loading from CSV with pandas:

```
import pandas as pd
data = pd.read_csv("data.csv")
t = h.Vector(data["t"])
v = h.Vector(data["v"])
```

```
t,v
0.0,-65.0
0.025,-64.99925452909274
0.05,-64.9985207095132
0.075,-64.99779768226396
0.09999999999999999,-64.99708468737194
0.12499999999999999,-64.9963810528078
0.15,-64.99568618464123
```

# NEURON Developer's meetings

- Third Friday of the month, 10am EDT; 16:00 CEST on zoom.
- Agenda is on the NEURON GitHub wiki: https://github.com/neuronsimulator/nrn/wiki

NEURON

Search docs

USER DOCUMENTATION:

NEURON Python documentation

NEURON HOC documentation

Python tutorials

Python RXD tutorials

DEVELOPER DOCUMENTATION:

NEURON SCM and Release

NEURON Development topics

C/C++ API

» Welcome to NEURON's documentation!

## Welcome to NEURON's d

### User documentation:

- NEURON Python documentation
  - Quick Links
  - Basic Programming
  - Model Specification
  - Simulation Control
  - Visualization
  - Analysis
- NEURON HOC documentation
  - Quick Links
  - Basic Programming
  - Model Specification
  - Simulation Control
  - Visualization
  - Analysis
- Python tutorials
  - ball-and-stick-1
  - ball-and-stick-2
  - ball-and-stick-3
  - ball-and-stick-4
  - pythontutorial

# NEURON Resources

## Unified documentation

- tinyurl.com/neuron-docs

## Forum

- tinyurl.com/neuron-forum

## NEURON models on ModelDB

- tinyurl.com/neuron-models

## CNS 2020 Tutorial

- tinyurl.com/neuron-cns2020