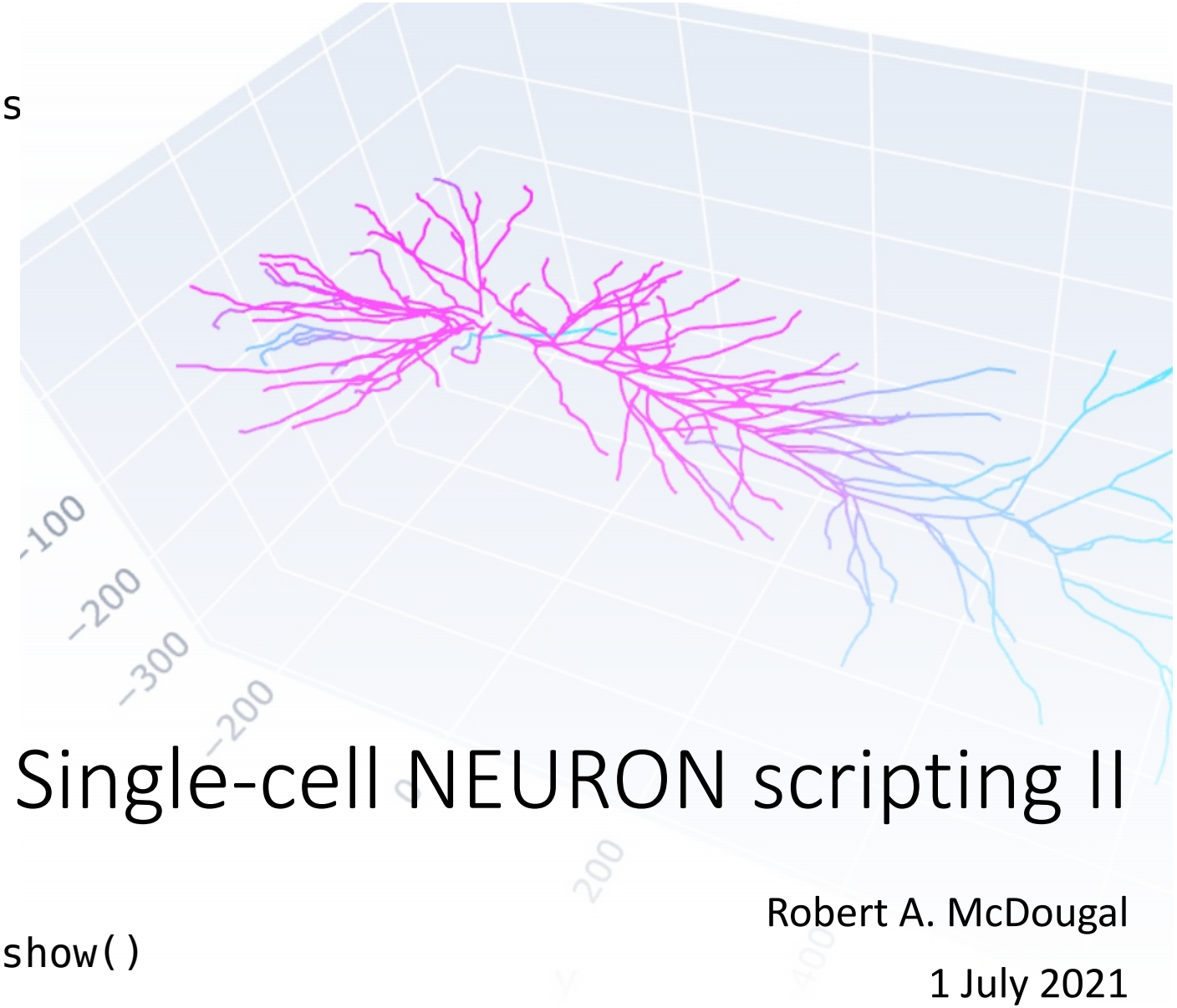
```
from neuron import h
from neuron.units import mV, ms
from matplotlib import cm
import plotly
h.load_file('stdrun.hoc')

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()
```



Single-cell NEURON scripting II

Robert A. McDougal

1 July 2021

Today

- Miscellaneous tools and techniques
  - neuron.rxd – intracellular and extracellular ion/protein/etc dynamics
  - pandas – for file storage (e.g. CSV, excel) and databases (e.g. sqlite3)
  - eFEL – electrophys feature extraction library
- Shape and position
- Events, networks, and integrate-and-fire cells (Ted)

Intra/extracellular chemical dynamics

```
from neuron import rxd
```

For more: tinyurl.com/netpyne-rxd-2021

Why use NEURON's `rxd` module?

Reduces typing

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

```
cyt = rxd.Region(soma.wholetree(), nrn_region="i")
ca = rxd.Species(cyt, name="ca", d=1, charge=2)
```

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

Allow arbitrary domains

- By default, NEURON only has two domains for chemical 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, 3D extracellular space, etc).

`rxd` module overview

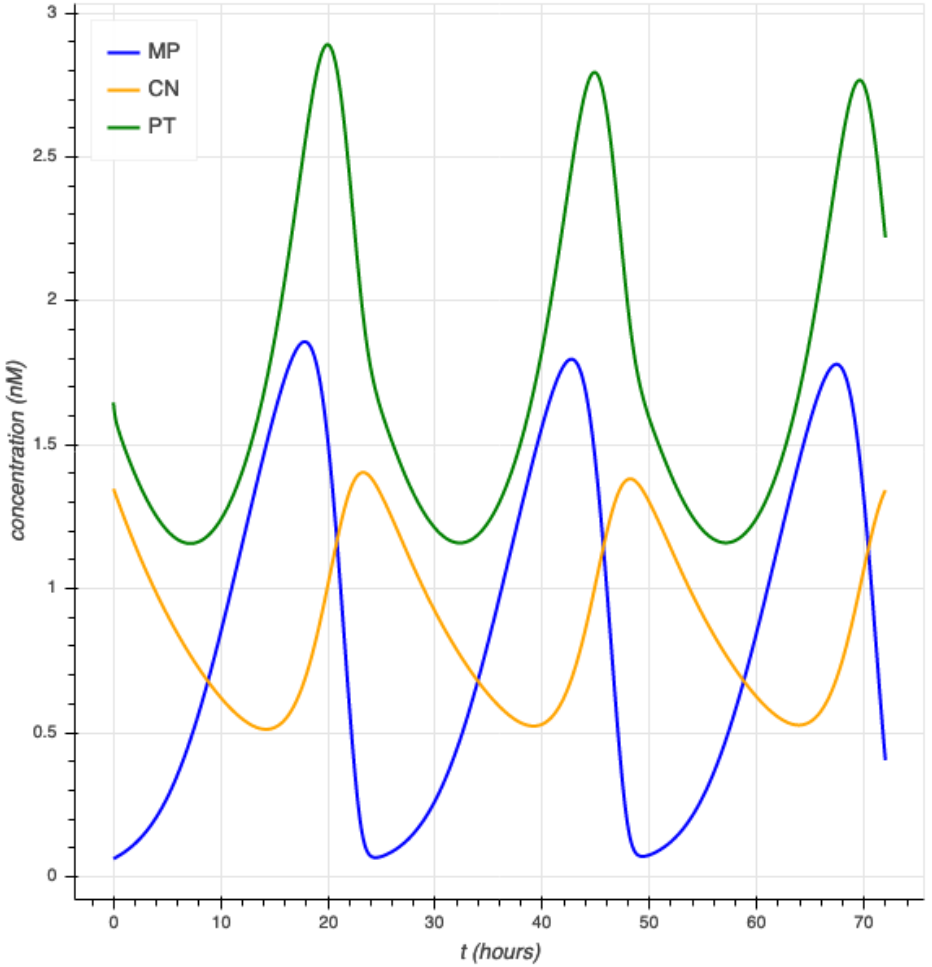
- **Where** do the dynamics occur? (`rxd.Region`, `rxd.Extracellular`)
  - Cytosol
  - Endoplasmic reticulum
  - Mitochondria
  - Extracellular Space
- **Who** are the actors? (`rxd.Species`, `rxd.State`, `rxd.Parameter`)
  - Ions
  - Proteins
- **What** are the reactions? (`rxd.Reaction`, `rxd.Rate`,…)
  - Buffering
  - Degradation
  - Phosphorylation

Interface design principle

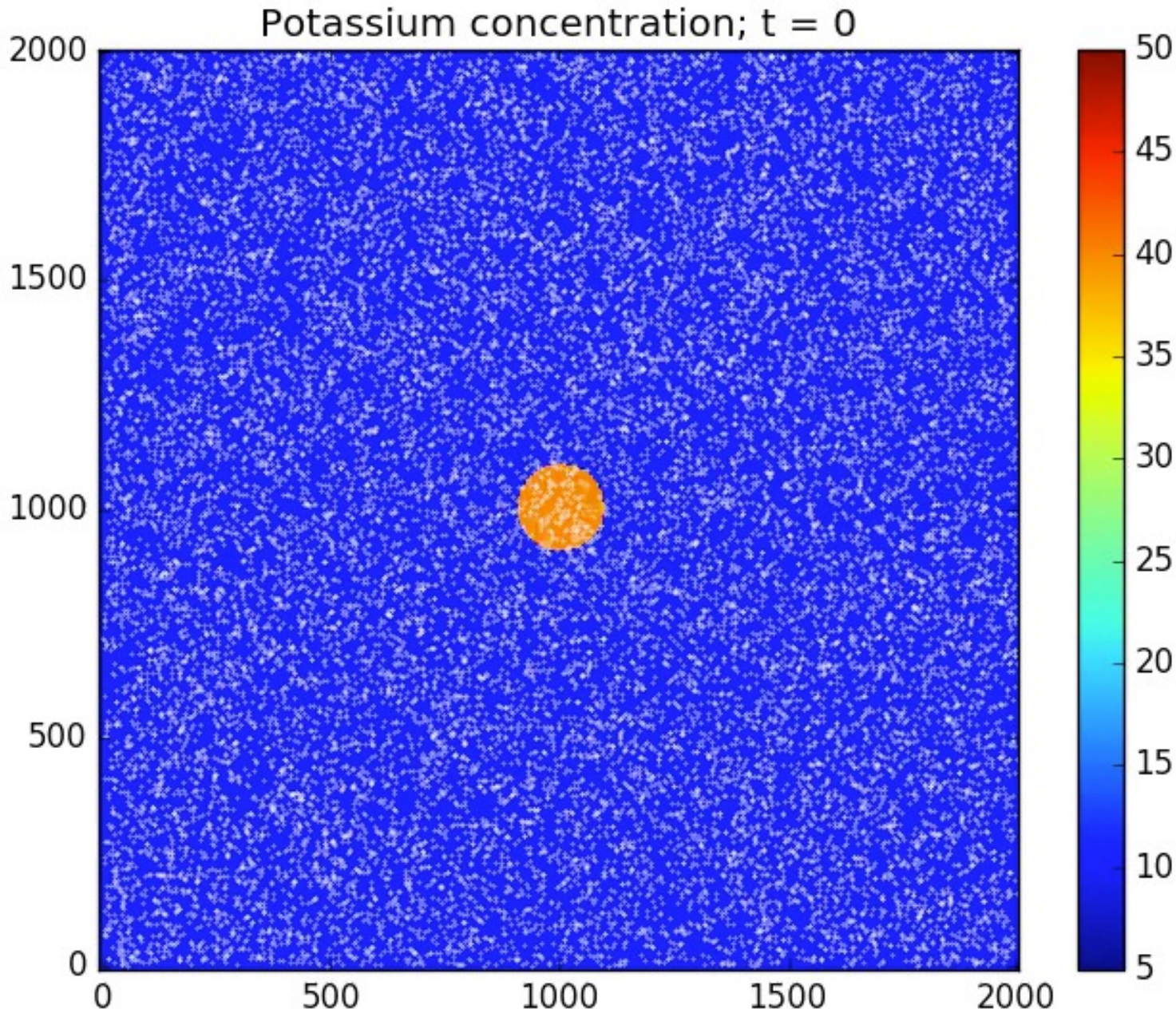
- Reaction-diffusion model specification is independent of:
  - Deterministic or stochastic
  - 1D or 3D

Example:

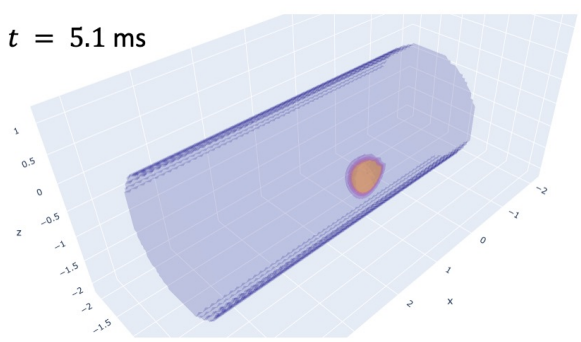
Leloup, Gonze, Goldbeter 1999

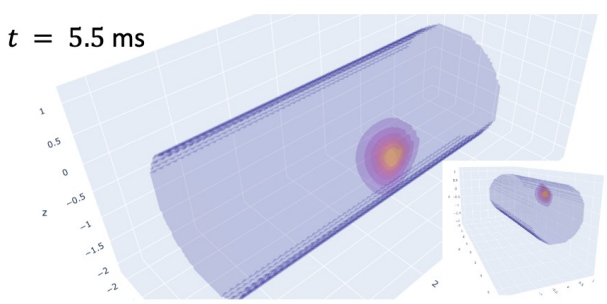


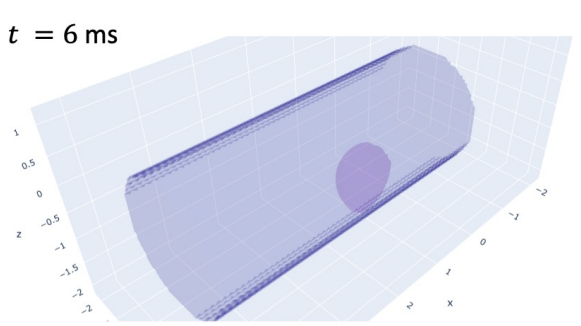
https://tinyurl.com/neuron-leloup-1999

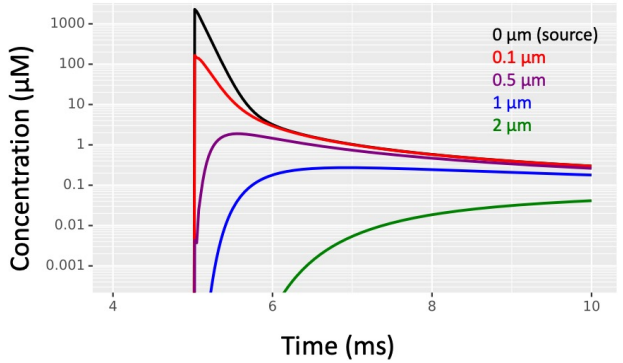

Potassium concentration; t = 0
2000
1500
1000
500
0
0
500
1000
1500
2000
50
45
40
35
30
25
20
15
10
5

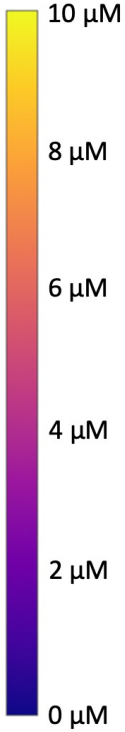
Example: 3D point source












```
NEURON {
  POINT_PROCESS RxDSyn
  RANGE tau
}

PARAMETER {
  tau = 1 (ms)
}

STATE { g }

INITIAL { g=0 }

BREAKPOINT {SOLVE state METHOD cnexp}

DERIVATIVE state { g' = -g/tau }
NET_RECEIVE(weight) {
  g = g + weight
}
```

```
node.include_flux(r._ref_g)
```

```
rxd.set_solve_type(seclist, dimension=3)
```

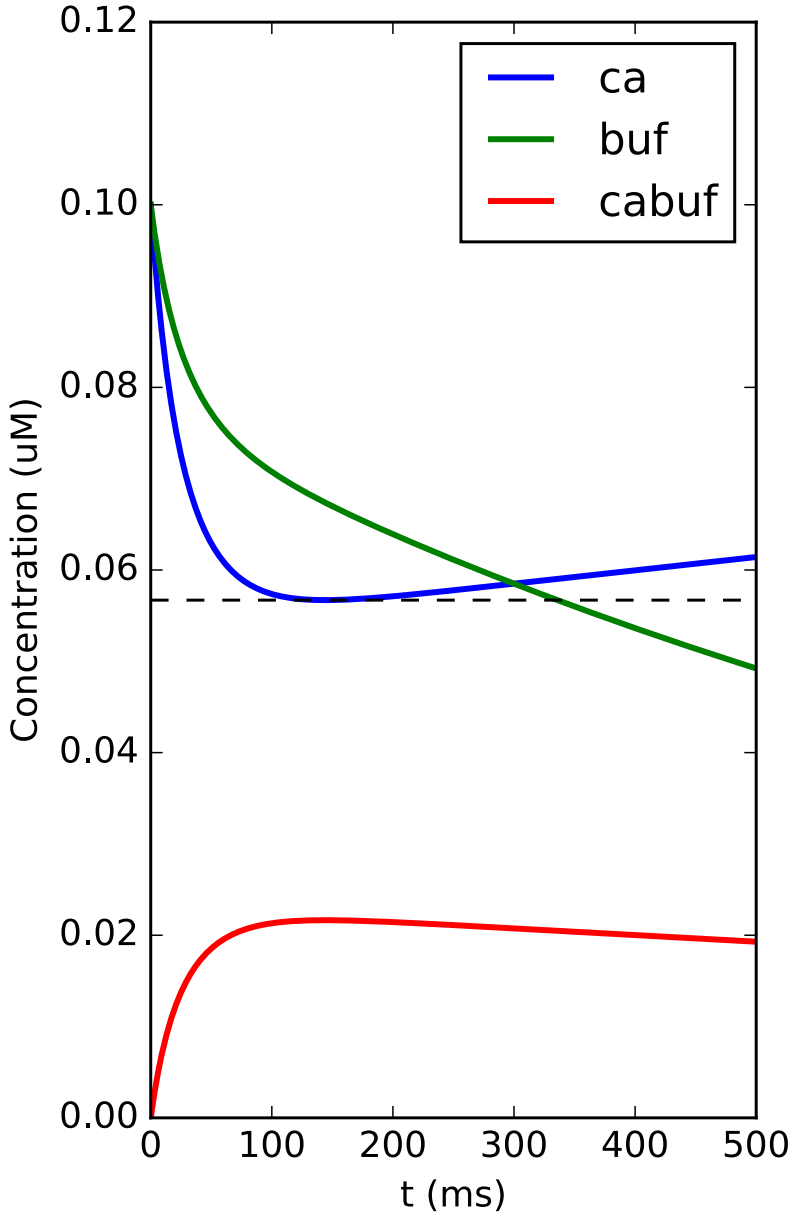
Example:

Calcium buffering*

Consider calcium buffering with a degradable buffer:

$$2Ca + Buf \leftrightarrow CaBuf, \quad Buf \rightarrow (degraded)$$

```
from neuron import h, rxd
from neuron.units import nM

# where
soma = h.Section(name="soma")
cyt = rxd.Region([soma], nrn_region="i")

# who
ca = rxd.Species(cyt, name="ca", charge=2, initial=100*nM)
buf = rxd.Species(cyt, name="buf", initial=100*nM)
cabuf = rxd.Species(cyt, name="cabuf", initial=0)

# what
buffering = rxd.Reaction(2 * ca + buf, cabuf, 1e6, 1e-2)
degradation = rxd.Rate(buf, 1e-3 * buf)
```



File storage and databases

The `pandas` library

For more: tinyurl.com/ycmi-pandas-2020

Storing and loading data with pandas

- Saving as CSV with pandas:

t and v are `h.Vector` instances

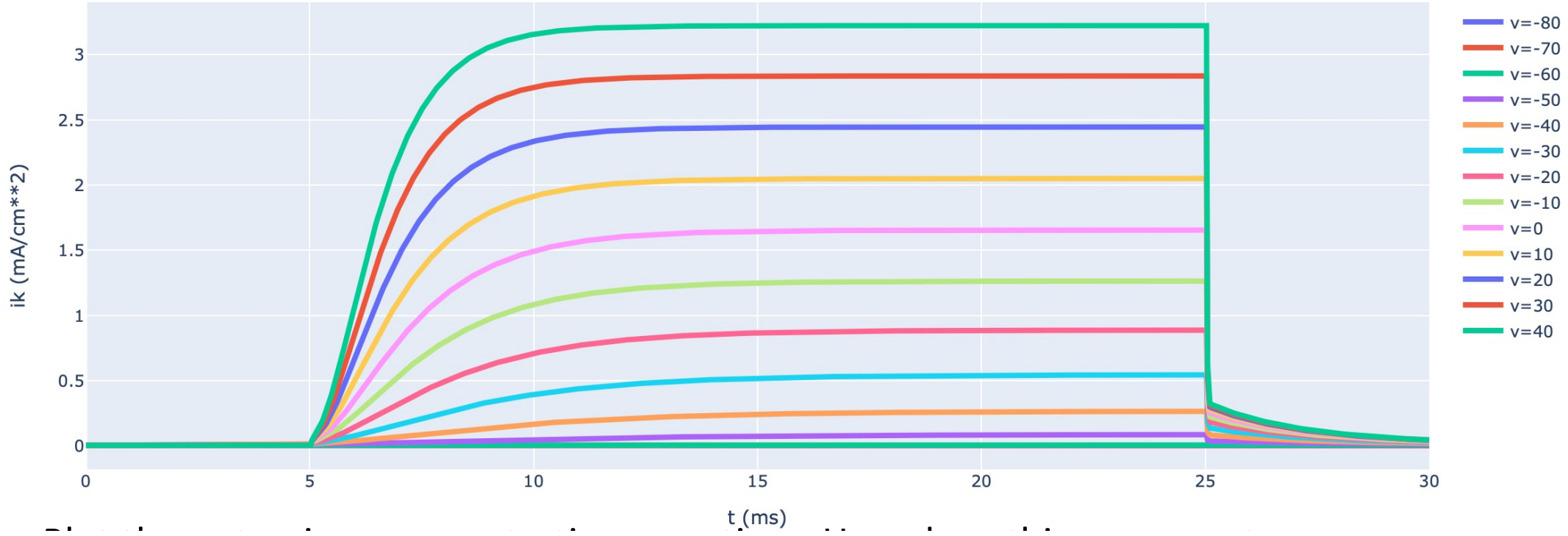
```
import pandas as pd
pd.DataFrame({"t": t, "v": v}).to_csv("data.csv", index=False)
```

- 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
```

Parameter sweeps



for loops, multiprocessing module

Saving to a database table

```
import pandas as pd
import sqlite3
import time

# do all the following inside a loop

start = time.perf_counter()
# do the simulation here

data = pd.DataFrame(
  {
    "dx": [dx],
    "L": L,
    "diam": diam,
    "alpha": alpha,
    "temperature": h.celsius,
    "spike_half_width": spike_half_width,
    "calculated_value": calculated_value,
    "runtime": time.perf_counter() - start,
  }
)
with sqlite3.connect(DB_FILENAME) as conn:
  data.to_sql("data", conn,
              if_exists="append", index=False)
```

DB_FILENAME is a string

Read from a database table

```
import pandas as pd
import sqlite3
import time

with sqlite3.connect(DB_FILENAME) as conn:
  data = pd.read_sql("SELECT * FROM data", conn)
```

```
with sqlite3.connect(DB_FILENAME) as conn:
data = pd.read_sql(
  """
    SELECT dx, L, diam, temperature, alpha FROM data
    WHERE calculated_value > 7
    ORDER BY temperature
  """, conn)
```

eFEL - Electrophys Feature Extraction Library

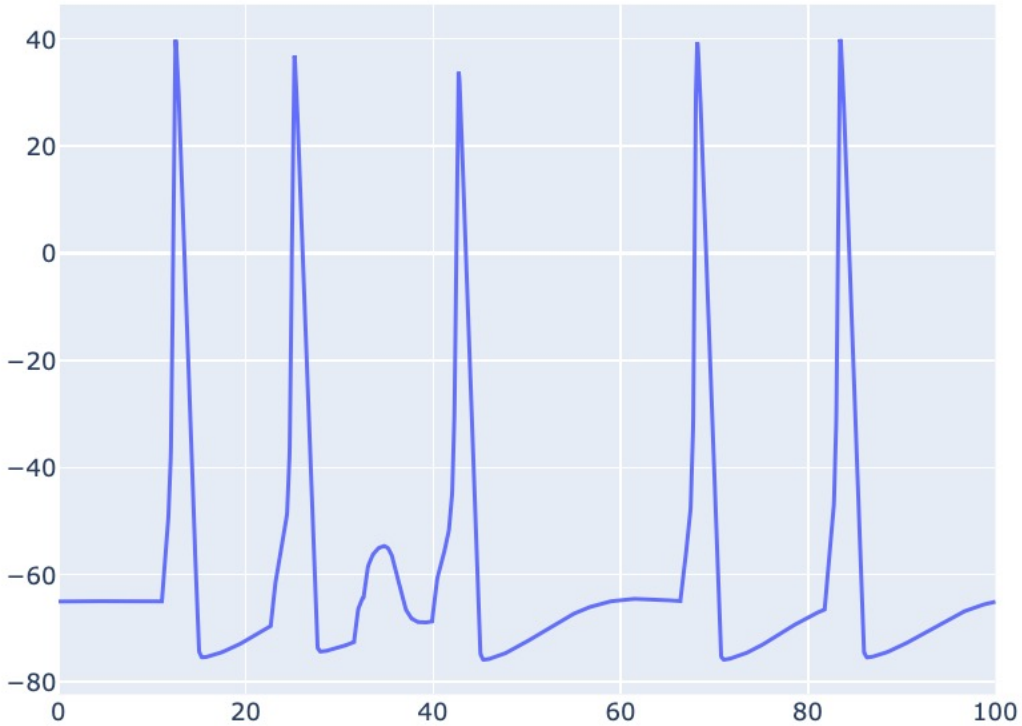
```
pip install efel
```

efel.readthedocs.io

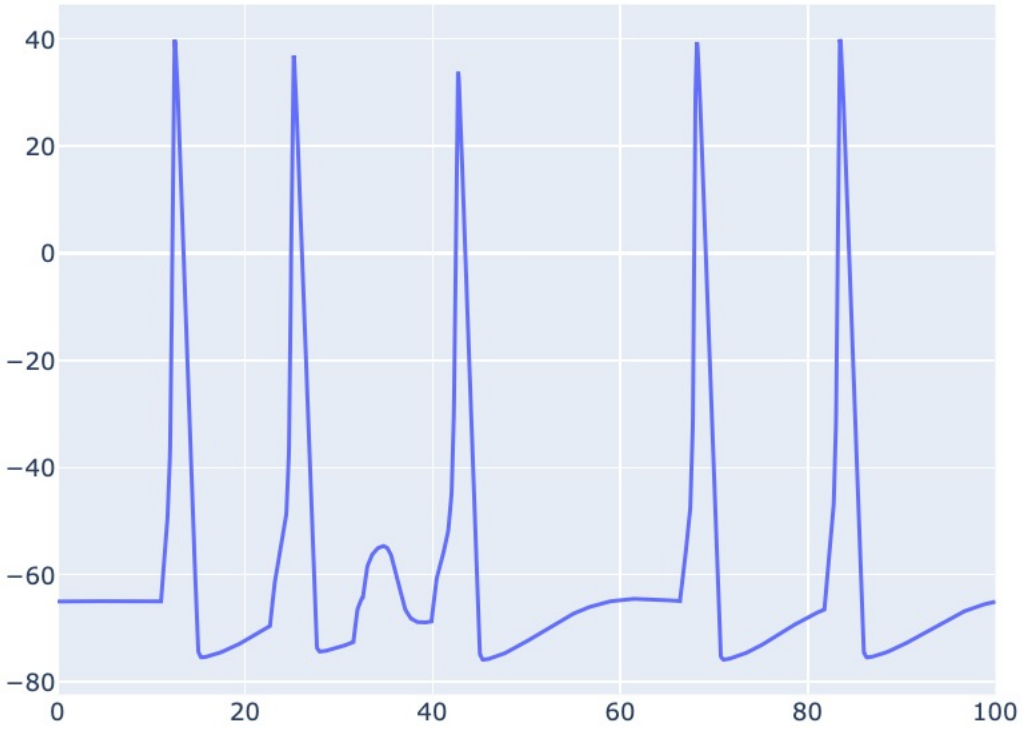
```
>>> import efel
>>> efel.getFeatureNames()
['AHP1_depth_from_peak', 'AHP2_depth_from_peak', 'AHP_depth', 'AHP_depth_abs',
'AHP_depth_abs_slow', 'AHP_depth_diff', 'AHP_depth_from_peak', 'AHP_slow_time',
'AHP_time_from_peak', 'AP1_amp', 'AP1_begin_voltage', 'AP1_begin_width', 'AP1_peak',
'AP1_width', 'AP2_AP1_begin_width_diff', 'AP2_AP1_diff', 'AP2_AP1_peak_diff',
'AP2_amp', 'AP2_begin_voltage', 'AP2_begin_width', 'AP2_peak', 'AP2_width',
'AP_amplitude', 'AP_amplitude_change', 'AP_amplitude_diff',
'AP_amplitude_from_voltagebase', 'AP_begin_indices', 'AP_begin_time',
'AP_begin_voltage', 'AP_begin_width', 'AP_duration', 'AP_duration_change',
'AP_duration_half_width', 'AP_duration_half_width_change', 'AP_end_indices',
'AP_fall_indices', 'AP_fall_rate', 'AP_fall_rate_change', 'AP_fall_time',
'AP_height', 'AP_peak_downstroke', 'AP_peak_upstroke', 'AP_phaseslope',
'AP_phaseslope_AIS', 'AP_rise_indices', 'AP_rise_rate', 'AP_rise_rate_change',
'AP_rise_time', 'AP_width', 'APlast_amp', 'APlast_width', 'BAC_maximum_voltage',
'BAC_width', 'BPAPAmplitudeLoc1', 'BPAPAmplitudeLoc2', 'BPAPHeightLoc1',
'BPAPHeightLoc2', 'BPAPatt2', 'BPAPatt3', 'E10', 'E11', 'E12', 'E13', 'E14', 'E15',
'E16', 'E17', 'E18', 'E19', 'E2', 'E20', 'E21', 'E22', 'E23', 'E24', 'E25', 'E26',
'E27', 'E3', 'E39', 'E39_cod', 'E4', 'E40', 'E5', 'E6', 'E7', 'E8', 'E9', 'ISI_CV',
'ISI_log_slope', 'ISI_log_slope_skip', 'ISI_semilog_slope', 'ISI_values',
'Spikecount', 'Spikecount_stimint', 'adaptation_index', 'adaptation_index2',
'all_ISI_values', 'amp_drop_first_last', 'amp_drop_first_second',
'amp_drop_second_last', 'burst_ISI_indices', 'burst_mean_freq', 'burst_number',
'check_AISInitiation', 'current', 'current_base', ...]
```

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

soma = h.Section(name="soma")
soma.L = soma.diam = 10 * µm
h.hh.insert(soma)
syn = h.ExpSyn(soma(0.5))
syn.e = 0 * mV
syn.tau = 2 * ms

ns = h.NetStim()
ns.start = 5 * ms
ns.interval = 10 * ms
ns.number = 10
ns.noise = True

nc = h.NetCon(ns, syn)
nc.weight[0] = 0.001

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

h.finitialize(-65 * mV)
h.continuerun(100 * ms)
```

```
import plotly.graph_objects as go

fig = go.Figure()
fig.add_trace(go.Scatter(x=t, y=v))
fig.show()
```



tinyurl.com/neuron-webinar2021-efel




```
import efel

trace = {"T": t, "V": v,
         "stim_start": [5 * ms],
         "stim_end": [100 * ms]}
traces = [trace]

features = efel.getFeatureValues(
  traces,
  ["AP_amplitude", "ISIs",
   "spike_half_width", "AP_begin_time"])
```

features:

```
[{'AP_amplitude': array([103.29412261, 105.09884958, 105.28924314, 103.58659989,
         105.59554847]),
'AP_begin_time': array([11.1, 22.7, 31.6, 66.4, 81.8]),
'ISIs': array([12.7, 17.5, 25.5, 15.2]),
'spike_half_width': array([1.52675742, 1.41003978, 1.40225347, 1.48675265,
         1.64899456])}]
```

frontiers in Neuroinformatics

TECHNOLOGY REPORT
published: 07 June 2016
doi: 10.3389/fninf.2016.00017

CrossMark

BluePyOpt: Leveraging Open Source Software and Cloud Infrastructure to Optimise Model Parameters in Neuroscience

Werner Van Geit[1], Michael Gevaert[1], Giuseppe Chindemi[1], Christian Rössert[1], Jean-Denis Courcol[1], Eilif B. Muller[1], Felix Schürmann[1], Idan Segev[2,3] and Henry Markram[1,4]**

[1] Blue Brain Project, École Polytechnique Fédérale de Lausanne, Geneva, Switzerland, [2] Department of Neurobiology, Alexander Silberman Institute of Life Sciences, The Hebrew University of Jerusalem, Jerusalem, Israel, [3] The Edmond and Lily Safra Centre for Brain Sciences, The Hebrew University of Jerusalem, Jerusalem, Israel, [4] Laboratory of Neural Microcircuitry, Brain Mind Institute, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland

At many scales in neuroscience, appropriate mathematical models take the form of complex dynamical systems. Parameterizing such models to conform to the multitude of available experimental constraints is a global non-linear optimisation problem with a

pip install bluepyopt

Neuron Morphologies

Shepherd, G. M., & Brayton, R. K. (1987). Logic operations are properties of computer-simulated interactions between excitable dendritic spines. *Neuroscience*, *21*(1), 151-165.

Shepherd, G. M., Woolf, T. B., & Carnevale, N. T. (1989). Comparisons between active properties of distal dendritic branches and spines: implications for neuronal computations. *Journal of Cognitive Neuroscience*, *1*(3), 273-286.

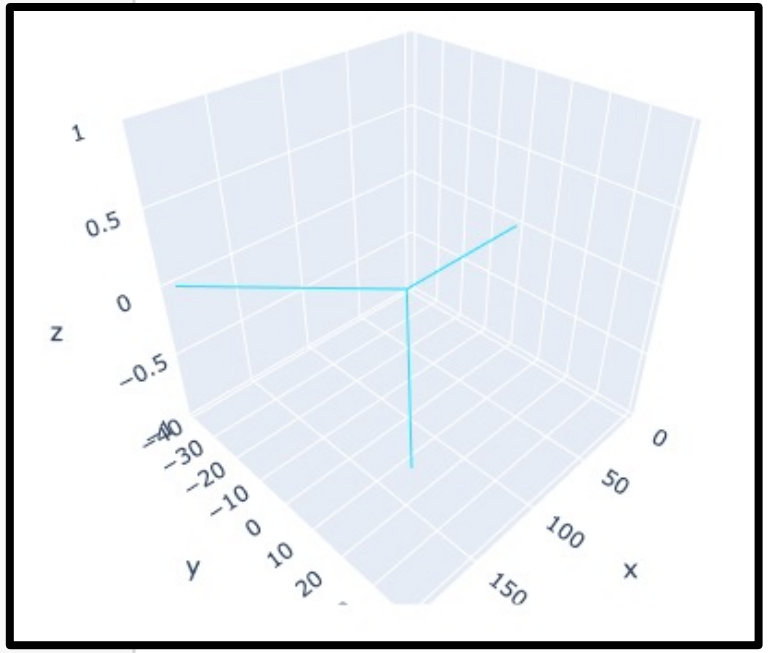
London, M., & Häusser, M. (2005). Dendritic computation. *Annu. Rev. Neurosci.*, *28*, 503-532.

Branco, T., & Häusser, M. (2010). The single dendritic branch as a fundamental functional unit in the nervous system. *Current opinion in neurobiology*, *20*(4), 494-502.

Cannon, R. C., O'Donnell, C., & Nolan, M. F. (2010). Stochastic ion channel gating in dendritic neurons: morphology dependence and probabilistic synaptic activation of dendritic spikes. *PLoS computational biology*, *6*(8), e1000886.

Viewing the morphology with h.PlotShape

```
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")
```

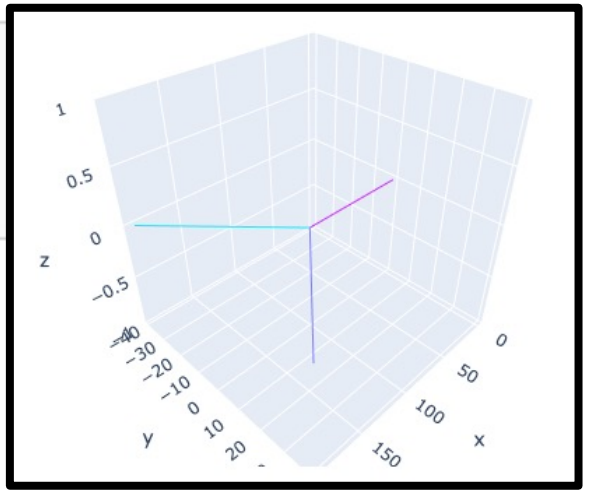
Viewing voltage, sodium, etc…

- 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
```

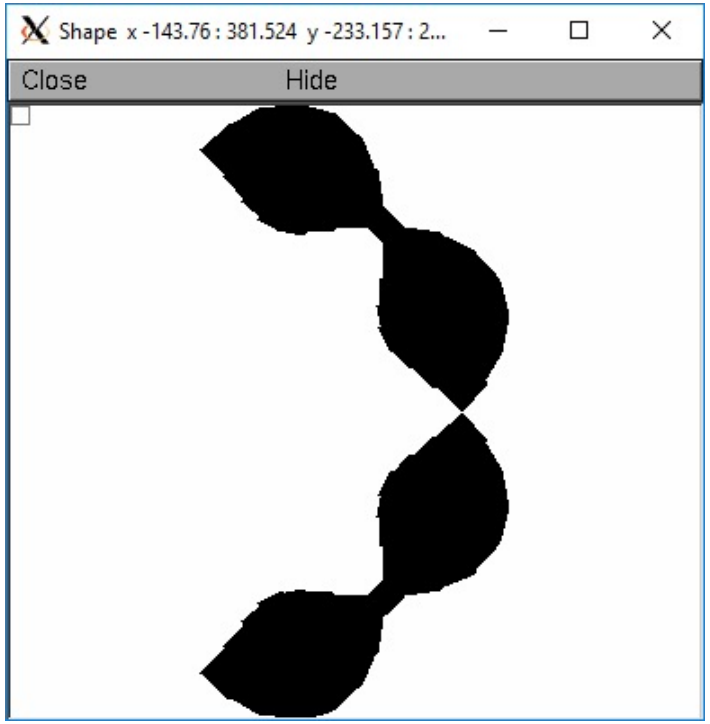
- We can create a PlotShape that color-codes the sections by voltage:

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



Manually set 3D points with sec.pt3dadd

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

sec = h.Section(name='sec')
sec.pt3dclear()
for i in range(31):
  x = h.PI * i / 30.
  sec.pt3dadd(200 * sin(x), 200 * cos(x),
              0, 100 * sin(4 * x))

s = h.Shape()
s.show(0)
```



Also: `sec.pt3dchange`, `sec.x3d(i)`, `sec.y3d(i)`, `sec.z3d(i)`, `sec.diam3d(i)`, `sec.arc3d(i)`, `sec.n3d()`, …

Example SWC file

```
1 1 0.835652 0.0943478 -1.71739 7.32349 -1
2 3 -21.43 38.81 -1.5 0.245 1
3 3 -19.94 39.29 -1.5 0.245 2
4 3 -18.71 39.05 -1.5 0.245 3
5 3 -17.72 38.81 -1.0 0.245 4
6 3 -15.74 39.05 -1.0 0.245 5
7 3 -14.5 39.05 -1.5 0.245 6
8 3 -13.51 38.09 -2.0 0.245 7
9 3 -11.53 37.36 -2.0 0.245 8
10 3 -10.04 36.88 -2.0 0.245 9
11 3 -7.82 35.68 -1.0 0.245 10
12 3 -6.58 35.19 0.0 0.245 11
13 3 -5.09 33.75 0.0 0.245 12
14 3 -3.11 33.27 -1.0 0.245 13
15 3 -0.39 33.02 0.5 0.125 14
16 3 1.09 33.02 -0.5 0.125 15
17 3 3.07 32.06 -1.0 0.125 16
18 3 4.06 32.06 1.5 0.125 17
19 3 5.05 32.06 2.0 0.125 18
20 3 6.29 31.58 0.0 0.125 19
```

- SWC is one of a variety of morphology formats that NEURON supports.
- These reconstructions are made (manually or automatically) from optical imaging.
- Each line in an SWC file has an identifier, a type, an x, y, z and diameter, as well as a parent identifier.

Trevelyan, A. J., Sussillo, D., Watson, B. O., & Yuste, R. (2006). Modular propagation of epileptiform activity: evidence for an inhibitory veto in neocortex. *Journal of Neuroscience*, *26*(48), 12447-12455

Ascoli, G. A., Donohue, D. E., & Halavi, M. (2007). NeuroMorpho. Org: a central resource for neuronal morphologies. *Journal of Neuroscience*, *27*(35), 9247-9251.

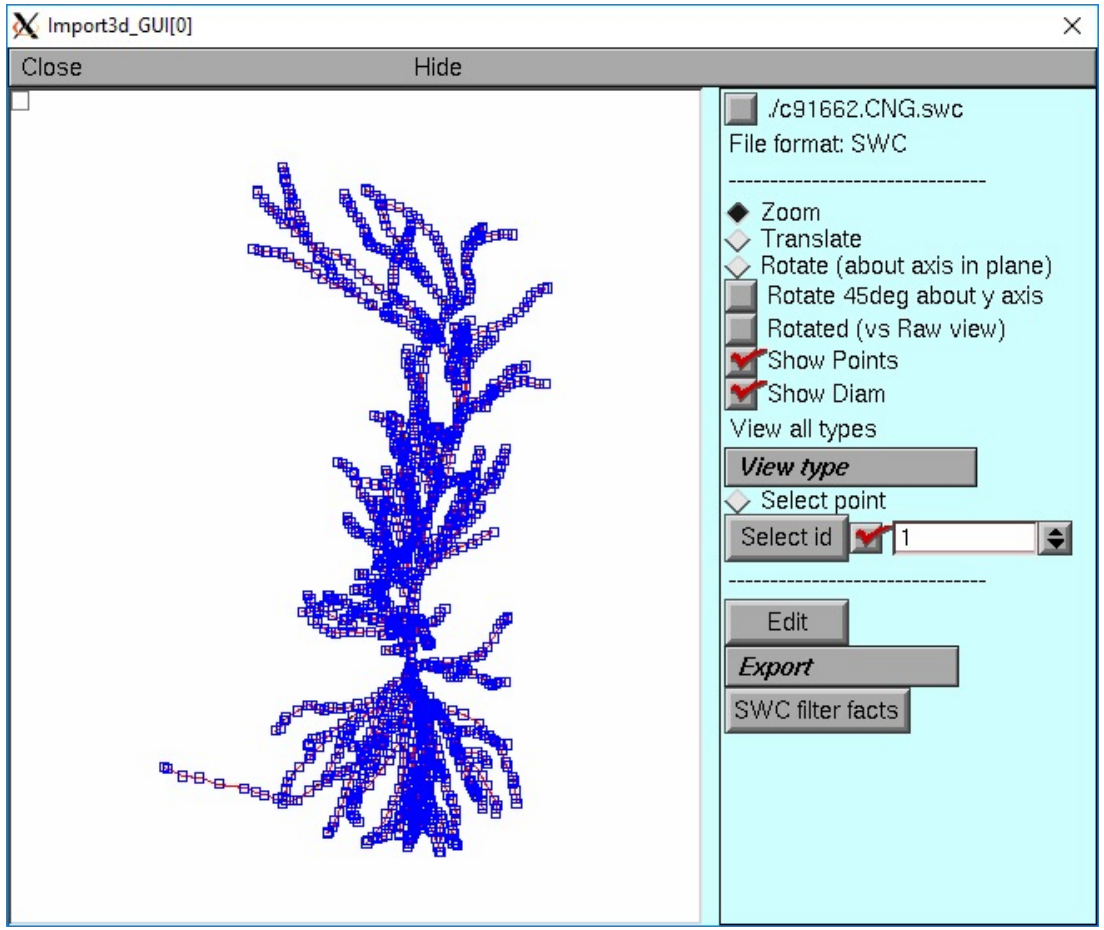
Import3D GUI

Access via:

Tools – Miscellaneous – Import 3D

Can instantiate directly into NEURON or transfer to CellBuilder.

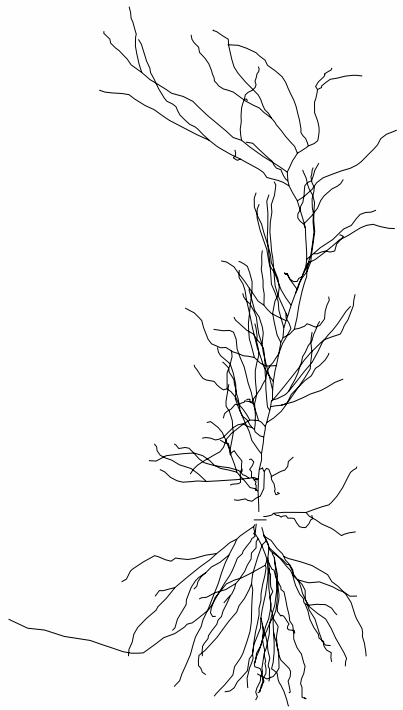
For more details, see:
neuron.yale.edu/neuron/docs/import3d



Loading morphology from an swc file

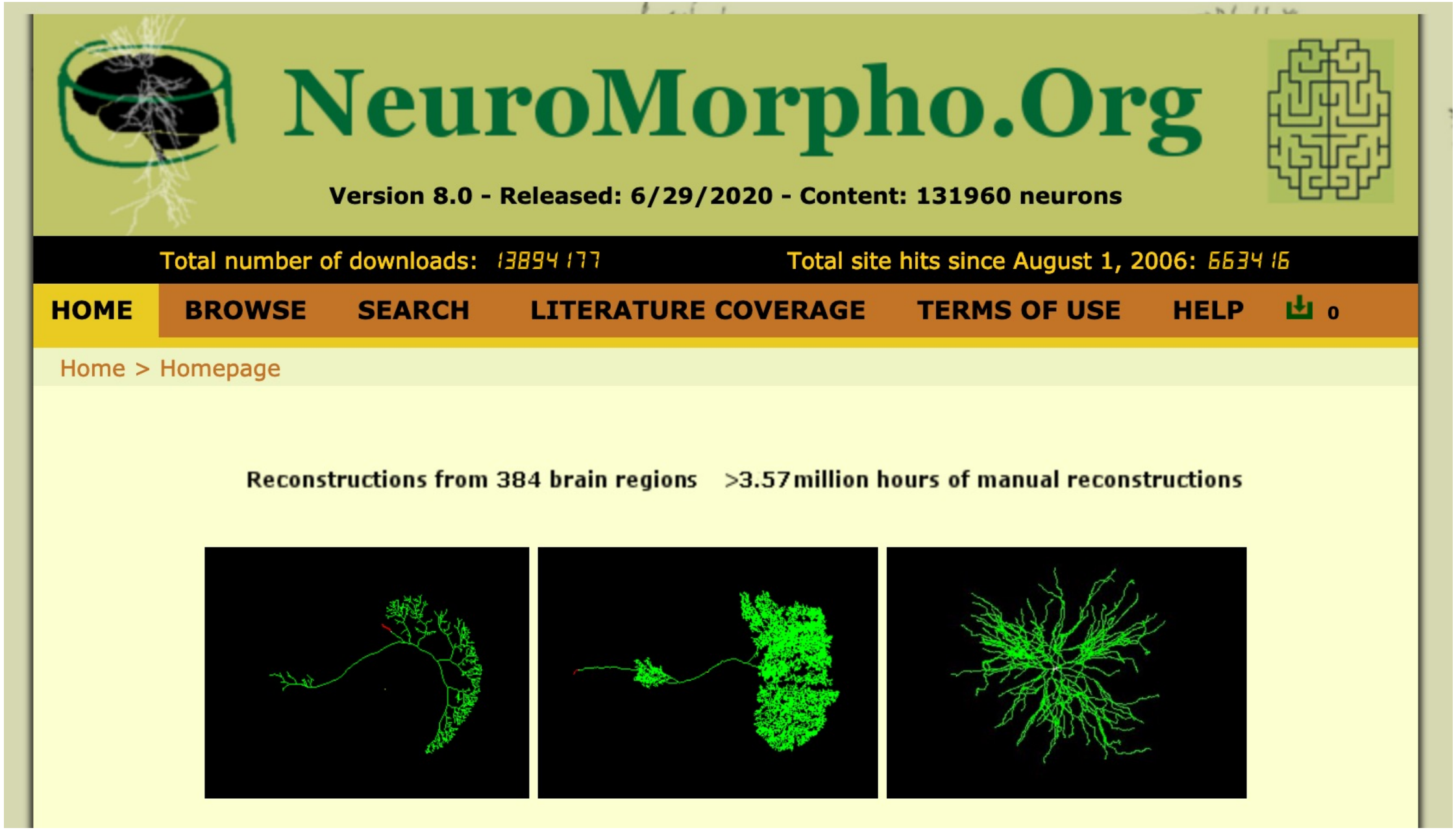
To create `pyr`, a Pyramidal cell with morphology from the file c91662.swc:

```
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()`

For Neurolucida asc files, use `h.Import3d_Neurolucida3` instead

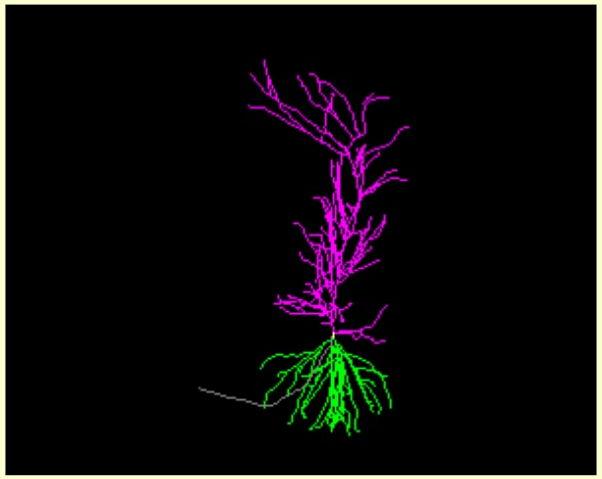


131,960 reconstructions · 641 cell types · 384 brain regions

NeuroMorpho.Org

Version 8.0 - Released: 6/29/2020 - Content: 131960 neurons

HOME BROWSE SEARCH LITERATURE COVERAGE TERMS OF USE HELP 0



Morphology File (Standardized)
Morphology File (Original)
Log File (Standardized)
Log File (Original)

Get above files zipped

3D Neuron Viewer - Java, legacy
3D Neuron Viewer - WebGL, novel
Animation

Details about selected neuron	
NeuroMorpho.Org ID :	NMO_00227
Neuron Name :	c91662
Archive Name :	Amaral
Species Name :	rat
Strain :	Sprague-Dawley
Structural Domains :	Dendrites, Soma, Axon
Physical Integrity :	Dendrites Complete, Axon Incomplete
Morphological Attributes :	Diameter, 3D, Angles
Min Age :	33.0 days

Standardized:
Always SWC

Original format:
Could be anything

Not everything was made for you

Not every morphology was reconstructed with the intent of being in a simulation.

Potential factors affecting the quality of the data:

- histology
  - staining, amputation, shrinkage
- physics
  - diameter
- spines

Before using a morphology found online, always read the associated paper(s) to make sure you understand any limitations of the reconstruction.

For example, why did they make this? Were they studying a disease (e.g. Alzheimer's) that alters morphology?

Qualitative tests

Look for orphan sections and bottlenecks.

Insert pas, set Ra and g_pas = pas.g low. Inject large depolarizing current at soma. Examine a PlotShape of v.

Look for z-axis drift and backlash.
Rotate the cell on a PlotShape and look for abrupt jumps.

Are diameters constant or varying? Are they reasonable?

For more on issues with morphologies

- Kaspirzhny, A. V., Gogan, P., Horcholle-Bossavit, G., & Tyˇc-Dumont, S. (2002). Neuronal morphology data bases: morphological noise and assesment of data quality. *Network: Computation in Neural Systems*, 13(3), 357-380.

- Scorcioni, R., Lazarewicz, M. T., & Ascoli, G. A. (2004). Quantitative morphometry of hippocampal pyramidal cells: differences between anatomical classes and reconstructing laboratories. *Journal of Comparative Neurology*, 473(2), 177-193.

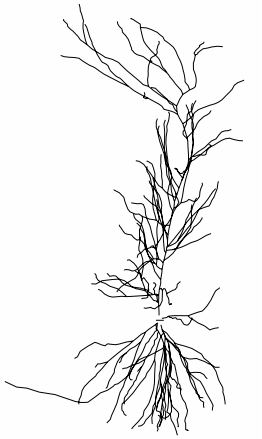
- MorphoUnit (SciUnit-based morphology testing):
  - github.com/appukuttan-shailesh/morphounit

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

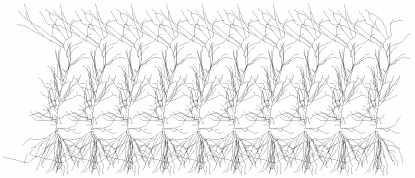
We can create a method to reposition a cell and call it from `__init__`:

```
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 or shape matter?

Sometimes.

They matter with:

- Connections based on proximity of axon to dendrite.
- Connections based on cell-to-cell proximity.
- Communicating about your model to other humans.
- Local field potentials.
- Extracellular stimulation.
- Extracellular diffusion.
- 3D intracellular chemical dynamics.

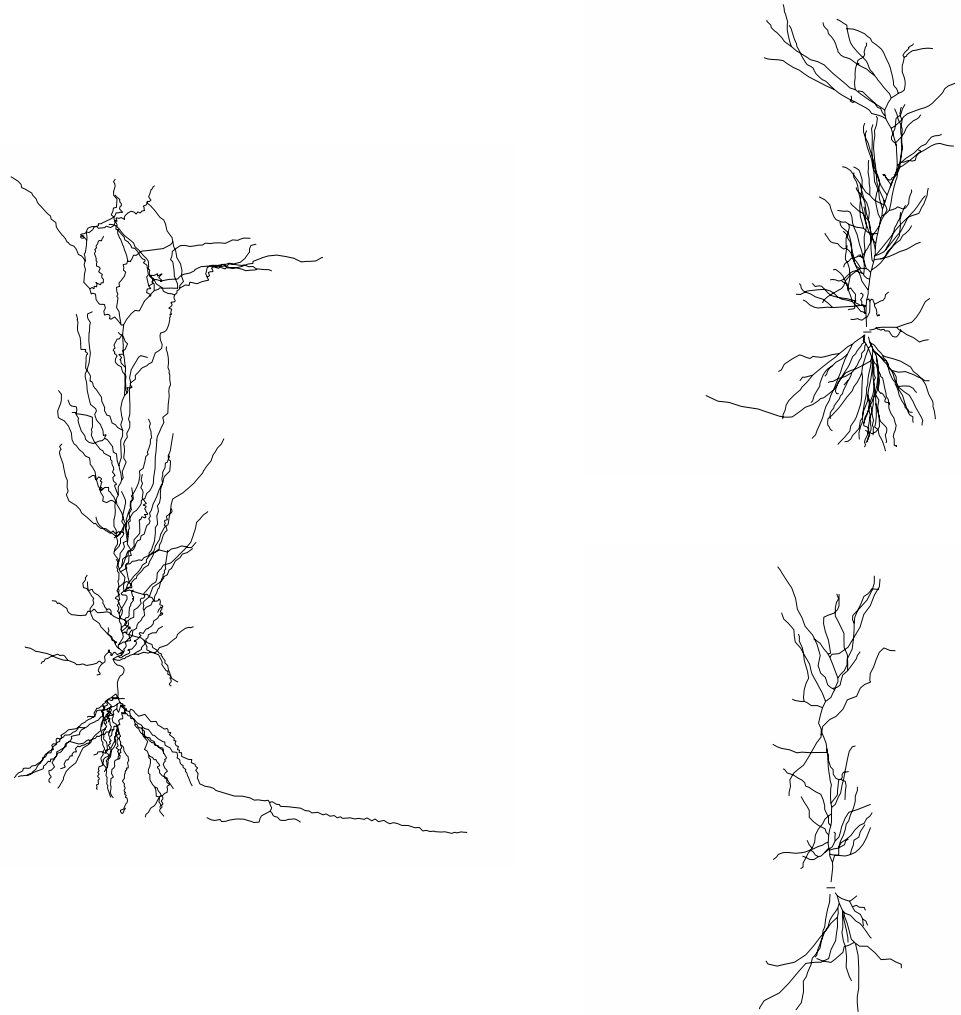
Exercise 1

- Hodgkin and Huxley did not originally define rules for how channel kinetics change with temperature, but NEURON's implementation of their dynamics does.
- Do a parameter sweep and examine how the action potential duration (`AP_duration` in eFEL) and any other electrophysiological properties you are interested in depend on `h.celsius`. Save your data to a `sqlite3` database using `pandas` and have a separate script plot the relationship.

Exercise 2

Download and examine the following three CA1 pyramidal cell morphologies (use the “standardized” version). What are your thoughts on the appropriateness of each for simulation?

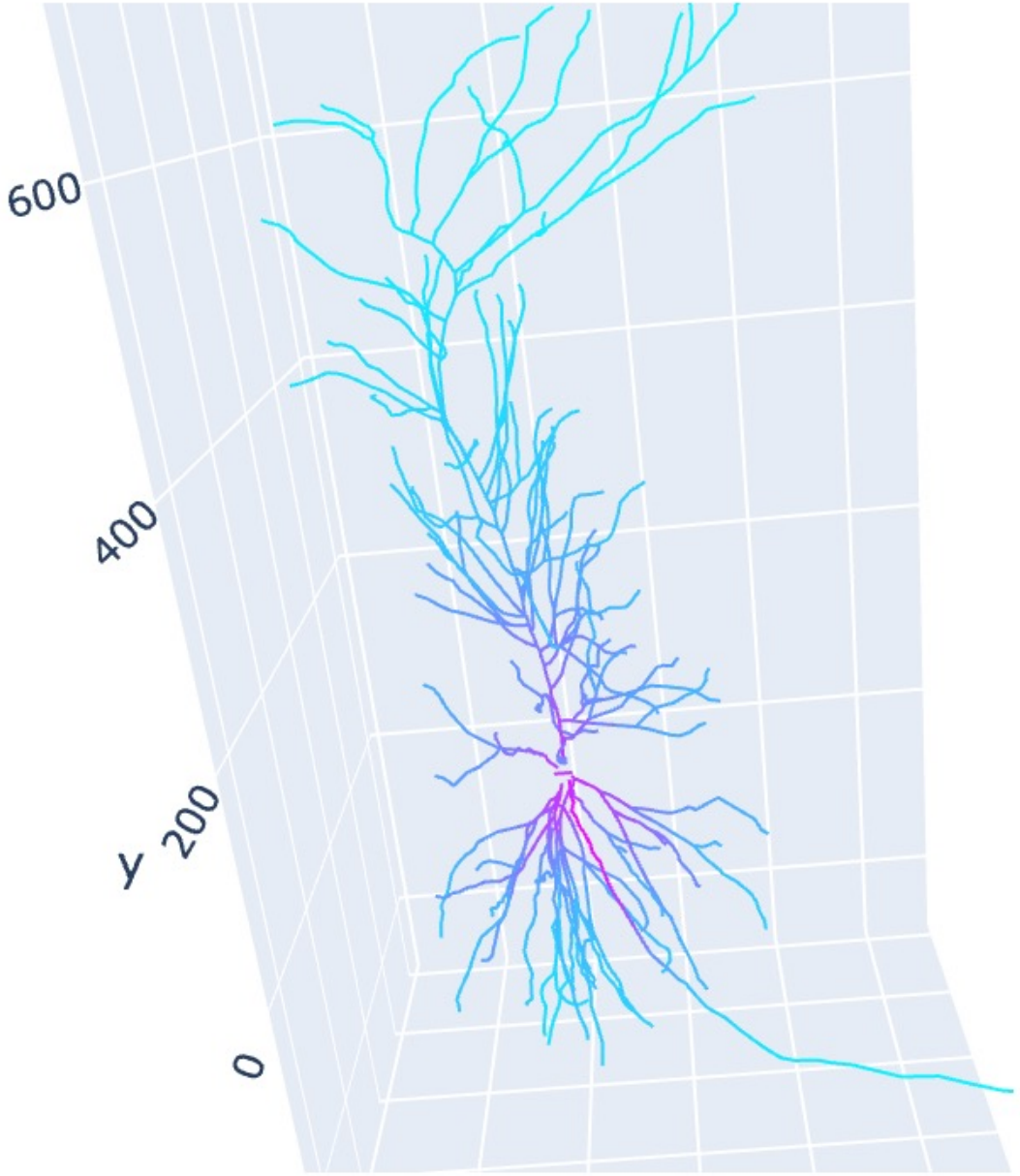
- tinyurl.com/neuromorpho-n123
- tinyurl.com/neuromorpho-c91662
- tinyurl.com/neuromorpho-calsynteninKO



Exercise 3

- Load the standardized morphology from: tinyurl.com/neuromorpho-c91662

- Insert Hodgkin-Huxley channels in the soma and axon, insert passive (h.pas) channels everywhere, set the passive conductance to 1e-6, choose a reasonable discretization, and plot the membrane potential mid-action potential.




```
!wget -O c91662.swc http://neuromorpho.org/dableFiles/amaral/CNG%20version/c91662.CNG.swc
```