Using NEURON and NEST to build a retina model in Python

1. Summary

2. Introduction

NEURON and NEST are two libraries useful to describe and simulate a population of neurons. The main difference between the two is the way a neuron is represented. In NEST, the way a neuron behaves is much like a logical module, a point process, where the neurons and their connections are like balls and sticks.
NEURON on the other side, is more realistic in a way, because it uses a more complex representation of neurons, using multiple sections and segment to mimic the compartments of a brain cell.

The reason why we chose to use NEURON in this project, is to have an accurate simulation of an electrical stimulation of a retina, taking into account not only the somas, but their axons as well, which NEST cannot allow.

This documentation is here to help getting started in this project, and to save some useful time of research through confusing web pages. Please if you need some additional information, do email me at camil.hamdane@epfl.ch

3. Download & Setup guide

Any OS

- **NeuroVM** is a virtual machine image, running Ubuntu, along with NEST, NEURON, and other useful softwares already installed. It is **the quickest way** to install, as NEURON is quite complex to install and get it going with Python. This VM saved me extra hours of research and failed install *(my reply in the thread speaks for itself)*
  NeuroVM is a virtual image which requires **VirtualBox** to run. To launch the machine, simply *import* the image (the .ova file) into VirtualBox, and run it.

Windows

Unfortunately, NEST is not currently natively supported in Windows, thus I'd recommend going for the previous option and get VirtualBox running on your Windows system. This is the way I did it.

OSX/Ubuntu

If you wish to install everything yourself, you can download every software and library from their host website:

- Python
- NEST
- NEURON will require you to "pair" it with Python in order to use them together. This operation can be confusing, but you can find some information on the official website here or you can try this more detailed guide here

To check if the installation worked properly, enter the following commands in the terminal:

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

This is what you should get:

```
NEURON -- VERSION <current version> master (6b4c19f) 2017-09-25
Duke, Yale, and the BlueBrain Project -- Copyright 1984-2016
See http://neuron.yale.edu/neuron/credits
```

4. Code documentation

4.1 NEURON Basics

The first step is to create a section:

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

A section is made out of **segments** that range from 0 to 1, being the two ends of the section. The amount of segment in a section is set with `soma.nseg = 11` *note that this value should always be odd, so that soma(0.5) is a segment with an integer index.*
You should see a section as a compartment of a cell. To get another one and connect them, input the following:

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

to check the current topology, type `h.topology`.

Now, let's setup some **Biophysics** into these section, to get the proper biological properties of neurons.

```
for sec in h.allsec():      # h.allsec() refers to all the current sections
created (soma, dendrite)
    sec.Ra = 100            # Axial resistance in Ohm * cm
    sec.cm = 1              # Membrane capacitance in microFarads / cm^2
    sec.diam = 10           # Section diameter, useful for topology & display,
in nm
```

These are the basic parameters inside the Section class. The documentation for the class can be found **here** or **here**.
The next step is to add **Mechanisms** into a section. These are *modules* that can be tweaked with various parameters, such as leak channels, Hodgkin–Huxley modules...
Let's insert a few useful modules:

```
# Insert active Hodgkin-Huxley current in the soma
soma.insert('hh')
for seg in soma:
    seg.hh.gnabar = 0.12  # Sodium conductance in S/cm2
    seg.hh.gkbar = 0.036  # Potassium conductance in S/cm2
    seg.hh.gl = 0.0003    # Leak conductance in S/cm2
    seg.hh.el = -54.3     # Reversal potential in mV

# Insert passive current in the dendrite
dend.insert('pas')
for seg in dend:
    seg.pas.g = 0.001  # Passive conductance in S/cm2
    seg.pas.e = -65    # Leak reversal potential mV
```

The documentation for point processes and mechanisms can be found here.

4.2 How to inject a specific voltage from one section to another

The next step is integrating an actual current between the two sections. This can be done multiple ways. Here we are going to see 2 ways that are the most relevant for this project:

IClamp

The first way to inject a current is through an IClamp, or a current clamp.

```
# First we need to initialize the clamp and connect it to the center of the
soma:
stim = h.IClamp(soma(0.5))
stim.delay = 5          # ms - delay after which the simulation will happen
stim.dur = 10           # ms - duration of the actual stimulation
stim.amp = 0.5          # nA - intensity of the stimulation
```

VClamp

The second way to inject a current is through a VClamp, or a voltage clamp. The good thing here with the VClamp is that it goes well with the already existing NEST code, since the connection between cells is essentially just setting the voltage value of a neuron from its base value to the base value + the input.

```
# The initialization is done the same way with a VClamp
stim = h.VClamp(soma(0.5))
# The VClamp however is a bit more complicated than the IClamp. Essentially,
there is 3 distinct
# phases where the voltage input can be set to different values for different
periods of time.
# Here let's try a stimulation with 2 ms at 0 mV, then 5 ms at 50 mV, and 10
seconds at 0mV
stim.dur[0] = 2                 # ms
stim.dur[1] = 5
stim.dur[2] = 10
stim.amp[1] = 50                # nA
stim.amp[0] = stim.amp[2] = 0
```

Thus, it would be the preferred approach, however we were unsuccessful at finding a way to get it working the way we wanted, and the only satisfying output was achieved with an IClamp for some reason.
Note that these are the basic parameters for both IClamp and VClamp and there are others you can tweak to reach a more accurate result. To check the parameters of a clamp, simply click on the desired one.

4.3 How to measure voltage

The measurement and plotting of data in NEURON uses matplotlib and specifically the **pyplot library**. This comes in quite handy since NEST uses it too for plotting, hence integrating both libraries into one plot should be doable.
The code can be found here on the official NEURON website.

4.4 RGC Class

This section is made to clarify the RGC class, located in the retinal_GC.py file.

```
This is the rough sketch of what a RGC should look like. Of course all sections
are not to scale and have various length and biophysical properties.
------
|soma|
-----\\
   \hillock\
        \\--myelin1--ranvier--myelin2
```

First the `__init__(x,y,z)` method is called when initializing a RGC entity. One should create a RGC using:

```
cell = RGC( x, y, z )
```

The geographical coordinates will become attribute of the class, that can be called with `self.x, self.y, self.z`. The initializer then calls multiple functions for building a working retinal ganglion cell.

- `create_sections()` creates the different sections contained in one cell, namely, `soma, hillock, myelin1, ranvier, myelin2`.
- `build_topology()` connects the different sections between them, as shown in the sketch above.

- `shape_3D()` can be quite confusing, because the way 3D specification is done in NEURON is not intuitive at first. Basically, after setting the **lengths** (soma.L...) I used `h.pt3dadd()` to build the 3D model of a cell. Each section is rendered in 3D separately.
  Let us take the example of `self.soma` here:

  ```
  h.pt3dclear(sec=self.soma)
  h.pt3dadd(0+self.x, lensoma+self.y, 0+self.z, self.soma.diam, sec =
  self.soma)
  # this method adds a point in the 3D space that will form the future
  section,
  # and setting multiple points will form segments of various lengths, to
  # represent a cell in 3D.
  ```

- `define_biophysics()` is in charge of setting all the biophysical properties of the different sections. This is currently here that the VClamp is initialized.
  We can see a recurring scheme here with a `for` loop running across every segment in each section.
- `build_subsets()` is an auxiliary method, defining a useful attribute: `self.all` which is used to iterate over all the section of the cell.
- `move()` moves the cell in the 3D space across a predefined vector.
- `stimulate(voltage, duration)` stimulates the VClamp according to the `voltage` and `duration` parameters.
  I have included in the function the code for the IClamp too, if you wish to use it instead of a VClamp.

5. Useful Links

- NEURON Main page
- NEST Main page
- NEURON Forum

  There are a few useful boards inside the forum that are worth checking and posting to:

  - NEURON + Python integration
  - Getting Started
- NEURON + Python step by step guide
- NEURON + Python detailed Documentation
  - Point processes & mechanisms
- The NEURON book, by Hines and Carnevale This is the paper presenting the simulation environment, unfortunately it does not go into detail about the integration with Python, but rather with hoc.
- Naïg's repository, with the retina.py file