NEURON - tutorial C of Gillies & Sterratt (part 1)

http://web.mit.edu/neuron_v7.4/nrntuthtml/index.html

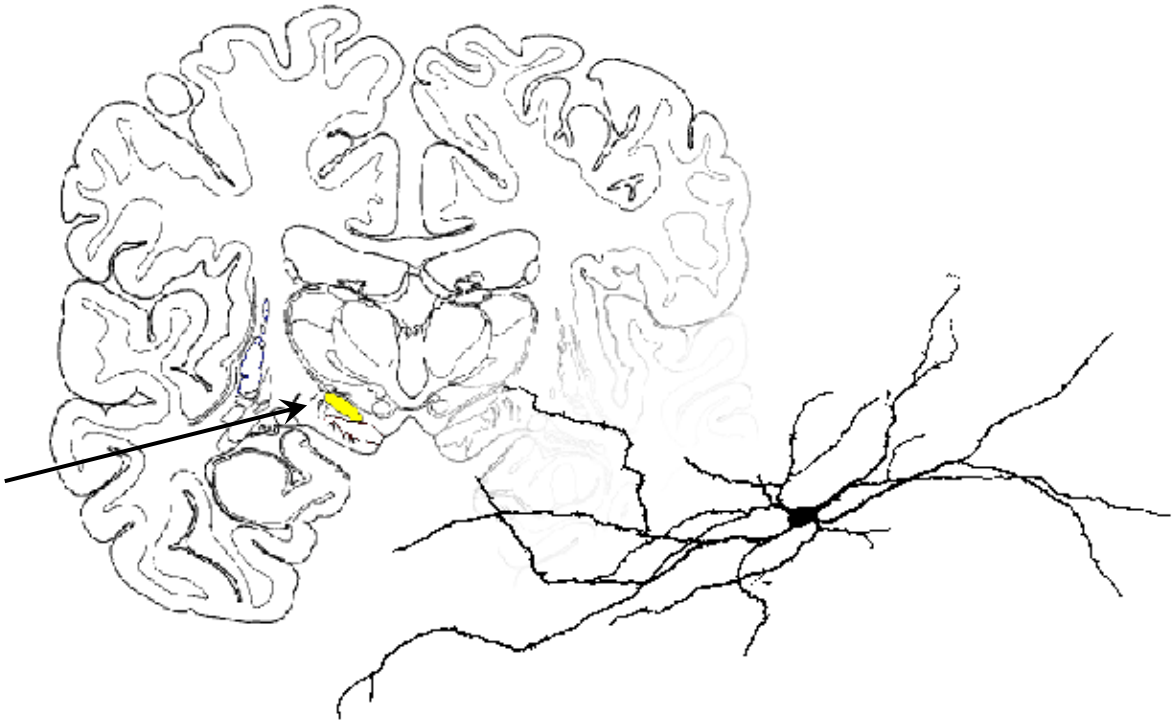
Lubica Benuskova

Lecture 5

How to define multiple neurons using templates

Centre for Cognitive Science Bratislava MEi: Cog Sci

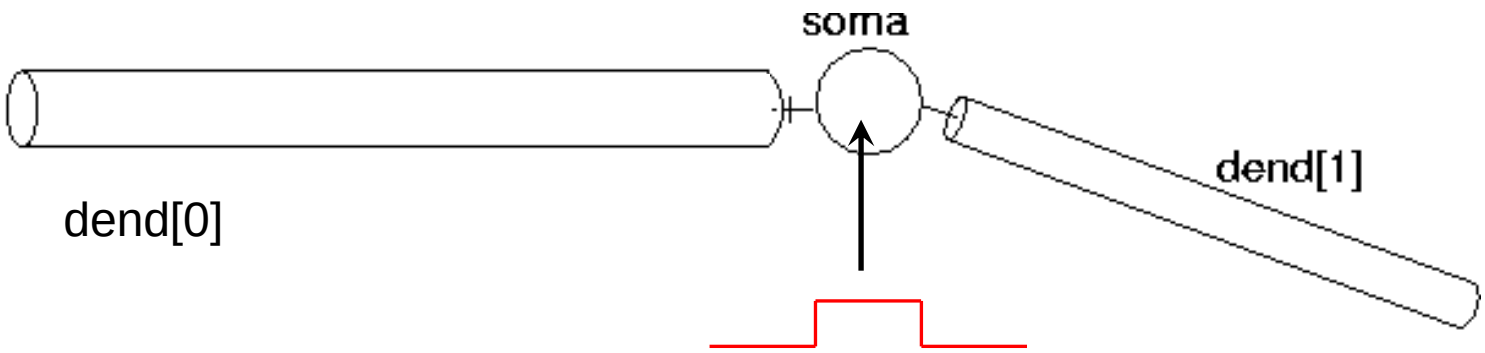
Our goal



- Our goal is to model and study small network of neurons in the rat subthalamic nucleus.
- So far, we created only one model neuron. In this lecture we will create more model neurons.

Our model so far **`sthB.hoc`**

- We have a soma with two dendrites (dend[0] and dend[1]) and there is a stimulating electrode in the soma, which injects a rectangular current pulse into the soma lasting 100 ms with the delay of 100 ms.



- We want to create a small network of such neurons. In this lecture we will create only 4 neurons, but we will create them in a way that increasing the number of neurons in the network is easy later.

Templates

- A *template* is an object definition – it defines a *prototype* of an object, from which we can create multiple copies.
- After defining the template, we must declare the *object variable* that we will use to reference these objects.
- Then, we can create a *new instance* of the object from the template.
  - The new object is an exact copy of the template.
- After we create the object from the template, we can either use it as it is, or we can modify it to fit our needs.

Template: definition

- The structure of a *template*:

```
begintemplate name
[public names]
create names
proc init() {
…
}
endtemplate name
```

- Notation: ***name*** is the name of the template for future reference.

- Square brackets **[public *names*]** mean this statement is optional.

Template: example

- The structure of *our template*:

```
begintemplate SThCell
public soma, dend
create soma, dend[1]
proc init() {
…
}
endtemplate SthCell
```

- Name **`SThCell`** is the name of the template for any future reference.
- Templates have a special procedure called **`init`**. It is automatically called when a new object is created from the template

The **`public`** statement

- The **`public`** statement is used to tell NEURON which parts of the template can be accessed from outside of the template definition.

- If there are no public names, then the code inside the template is completely private and nothing, aside from the name of the template itself, is accessible from the rest of the program code.

- For example, if we create a neuron template and we want to be able to put a current clamp in the soma of the neuron we create, we need to give access to the soma section via the public command, i.e., we need to type **`public soma.`**

Declarations before the **`init`** procedure

- In general, the two rules we need to follow are:

1) A section or object must be created / declared before the **`init`** (or any other) procedure, in which it is re-created.

2) When creating / declaring an array of sections or objects that will be re-created inside a procedure, create an array of dimension 1 before the procedure **`init`**.

```
create soma, dend[1]
```

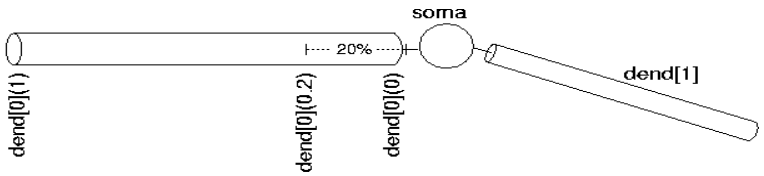
What's the code in the **`init`** procedure?

```
begintemplate SThCell

public soma, dend

create soma, dend[1]

proc init() {

ndend = 2

create soma, dend[ndend]

soma {
        nseg = 1
        diam = 18.8
        L = 18.8
        Ra = 123.0
        insert hh
}
```

```
dend[0] {
          nseg = 5
          diam = 3.18
          L = 701.9
          Ra = 123
          insert pas
        }

dend[1] {…}

// Connect things

connect dend[0](0),soma(0)
connect dend[1](0),soma(1)

}

endtemplate SThcell
```

The **`init`** procedure

- The templates have a special procedure called **`init()`** which is automatically called when a new object is created from the template.
- This procedure is used to initialise the newly created object.
- In our **`init()`** procedure above, we have created and defined *all the sections* of our model neuron and connected them together.
- Thus, when a new neuron object is created from the template, with the **`new`** command, an *entire* subthalamic neuron is built.



Creating new neurons from a template

- First, we define an *array* of *object variables*:

```
nSThcells = 4
objectvar SThcells[nSThcells]
```

- Second, we create four model neurons using the **new** command:

```
SThcells[0] = new SThcell()
SThcells[1] = new SThcell()
SThcells[2] = new SThcell()
SThcells[3] = new SThcell()
```

- Each model neuron is an exact copy of the template.
- We can create as many neurons as our computer can handle.

Creating new neurons in a loop

- After defining an *array* of *object variables*:

```
nSThcells = 4
objectvar SThcells[nSThcells]
```

- Instead of creating each neuron with a separate command, we use the **new** command within the so-called **for** loop:

```
for i = 0, nSThcells-1 {
    SThcells[i] = new SThcell()
}
```

- Letter "i" denotes an index within an array. It can be any letter (a, b, c, d, …) or even a word (index, loop, instance, cell, etc.).

Arguments to the **`init`** procedure

- Arguments can be passed to **`init()`** like to any other procedure.
- This can be used to affect the properties of the object via the parameters that you pass to the **`new`** command.
- As an example of passing arguments to **`init()`** procedure, suppose we wanted to have neurons with different numbers of segments in their dendrites, i.e., variable **`nsegden`** will have different values for different individual neurons.
- We can do this in a single population of neurons creating multiple copies of neurons with different **`nsegden`** values.

Arguments to the **`init`** procedure: example

- Let's write **`init()`** with an argument for **`nsegdend`**
- Code for a variable argument is **`$1`**
- If we had more than one argument, we will use **`$1, $2, $3,`** etc.

```
proc init() {
        nsegdend = $1

        ndend = 2
        create soma,dend[ndend]
        ...

        dend[0] {
        nseg = nsegdend
        ...
        }

        dend[1] {
        nseg = nsegdend
        ...
        }
}
```

Arguments to the **`init`** procedure: example

- To create a neuron (say neuron 0) with dendritic sections containing, for example, 13 segments, we do it using the command:

```
SThcells[0] = new SThcell(13)
```

- to create all of our four cells with 3, 6, 9, and 12 dendritic segments respectively we could type:

```
SThcells[0] = new SThcell(3)
SThcells[1] = new SThcell(6)
SThcells[2] = new SThcell(9)
SThcells[3] = new SThcell(12)
```

Arguments to the **`init`** procedure: example

- In order to create all of our four cells with 3, 6, 9, and 12 dendritic segments respectively we can use the **`for`** loop:

```
for i = 0, nSThcells-1 {
        SThcells[i] = new SThcell(3*(i+1))
}
```

- Finally, we need to remember to set a default section so that graphing works:

```
access SThcells[0].soma
```

- There must be at least one **`access`** statement in the code!

Accessing parts of neurons from outside

- After declaring the array with the **`objectvar`** command and creating the objects with the **`new`** command, we can access the sections using the dot notation (provided they are **`public !`**).

- E.g, we can insert current clamps into all of four somas as follows:

```
objectvar stim[nSThcells]

for i = 0, nSThcells-1 SThcells[i].soma {
      stim[i] = new IClamp(0.5)
      stim[i].del = 100
      stim[i].dur = 100
      stim[i].amp = 0.1
}
```

- Note: the NEURON code up to now is in the file **`SthC1.hoc`**

SthC1.hoc

```
load_file("nrngui.hoc")

begintemplate SThCell

public soma, dend

create soma, dend[1]

proc init() {

ndend = 2

create soma, dend[ndend]

soma { nseg = 1
       diam = 18.8
       L = 18.8
       Ra = 123.0
       insert hh
       gnabar_hh=0.25
       gl_hh = .0001666
       el_hh = -60.0 }
```

```
dend[0] {nseg = 5
         diam = 3.18
         L = 701.9
         Ra = 123
         insert pas
         g_pas = 0.0001666
         e_pas = -60.0 }

dend[0] {nseg = 5
         diam = 2.0
         L = 549.1
         Ra = 123
         insert pas
         g_pas = 0.0001666
         e_pas = -60.0 }

// Connect things together
connect dend[0](0),soma(0)
connect dend[1](0),soma(1)

}

endtemplate SThcell
```

SthC1.hoc contd.

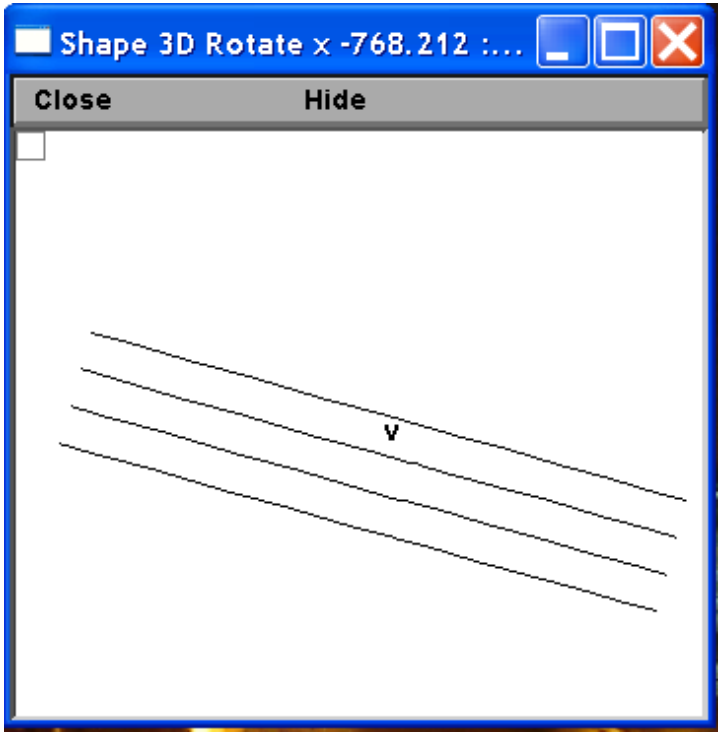
```
tstop = 300
nSThcells = 4

objectvar SThcells[nSThcells]

for i = 0, nSThcells-1 {
        SThcells[i] = new SThcell()

}

objectvar stim[nSThcells]

for i = 0, nSThcells-1 SThcells[i].soma {
    stim[i] = new IClamp(0.5)
    stim[i].del = 100
    stim[i].dur = 100
    stim[i].amp = 0.1

}

access SThcells[0].soma
```

Positioning neurons in 3D space

- Each time we create a new section and connect it to others, NEURON places the section in a 3-D space and assigns automatically X, Y and Z coordinates to each end of the sections.

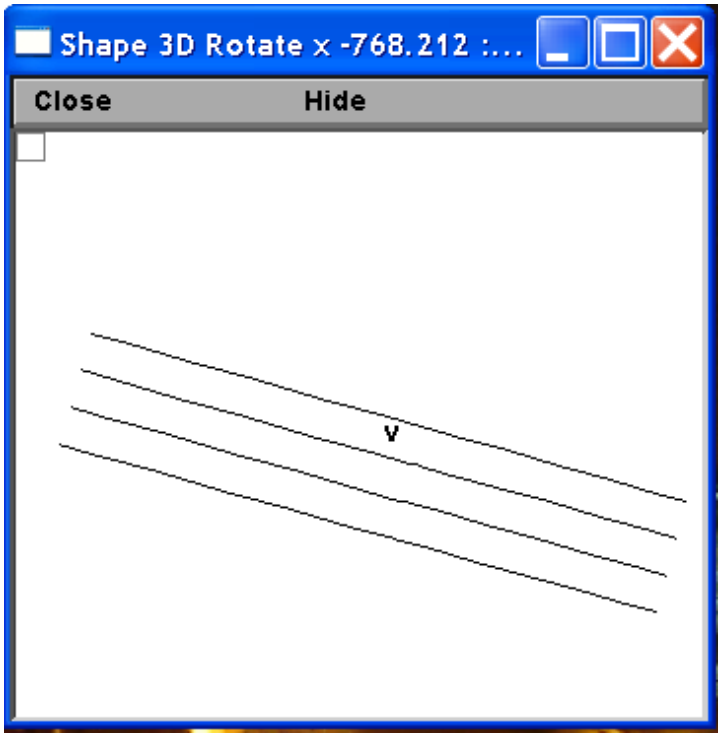
- When creating more than one neuron, each neuron is given a different Z coordinate for all of its sections.

- To see the default position of neurons, **open a space or shape plot** (under the Graph menu) right click on it and choose 3D rotate.



Re-positioning neurons in 3D space

- The default X and Y coordinates of each neuron are determined by how the individual sections are connected.
- This makes viewing the neurons difficult since they are not arranged how they are in reality.
- NEURON has two inbuilt functions to reposition each section: **`pt3dclear()`** and **`pt3dadd()`**.



Re-positioning neurons in 3D space

- The first function, **`pt3dclear(),`** will erase any 3D positioning information associated with the section.

- The second, **`pt3dadd(),`** takes four arguments (X, Y, Z, and diam) and will add a new coordinate to the section.

- Usually there are coordinates for each end of the section, which can be set by making two calls to **`pt3dadd()`** – once for the "0" end of the section and once for the "1" end of the section.

- We will demonstrate the action of these functions on a more complex dendritic trees in the next lecture.