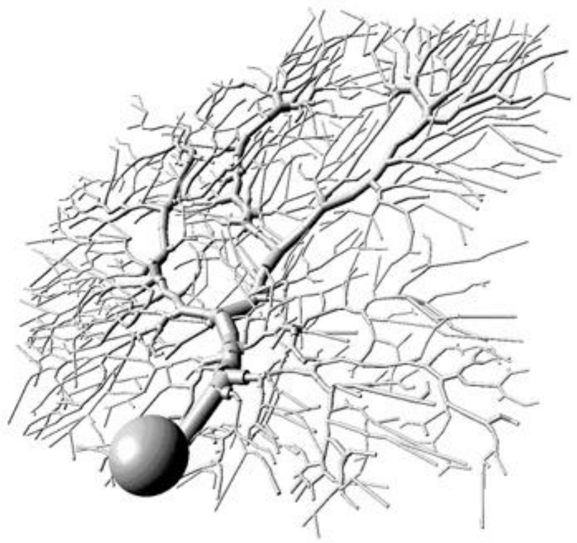
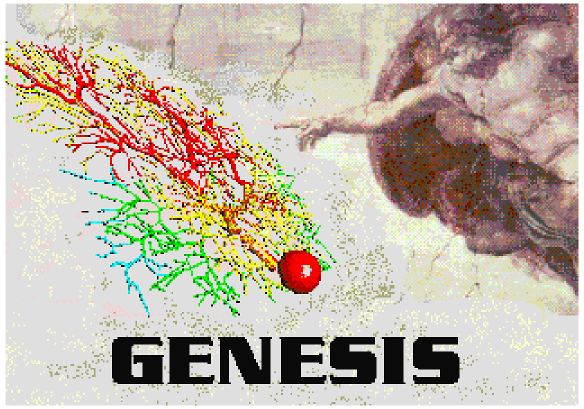
# Introducción a NEURON / NetPyNE

## Built-in solvers de neuronas detalladas

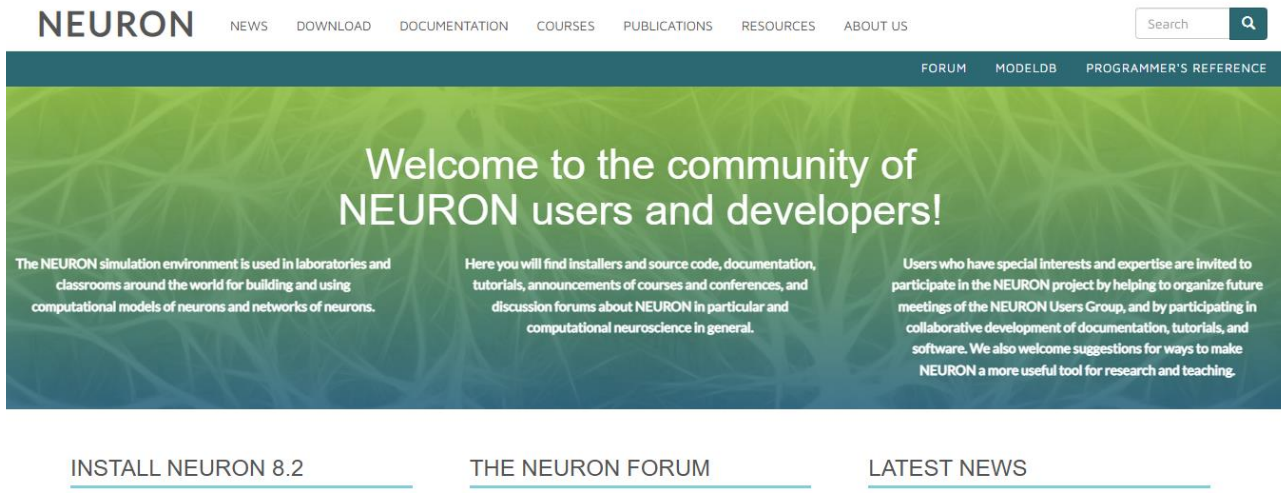
- **NEURON's CVODE:** NEURON provides the CVODE solver, which is an adaptive, variable-order solver designed for stiff and non-stiff systems. It allows accurate time-stepping while adapting to the system's needs.
- **GENESIS Solvers:** GENESIS also provides several solvers optimized for neural simulations, including implicit methods and adaptive time-stepping.



**GENESIS 2.4 Reference Manual**



**May 2019 Update**



NEURON NEWS DOWNLOAD DOCUMENTATION COURSES PUBLICATIONS RESOURCES ABOUT US Search

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.

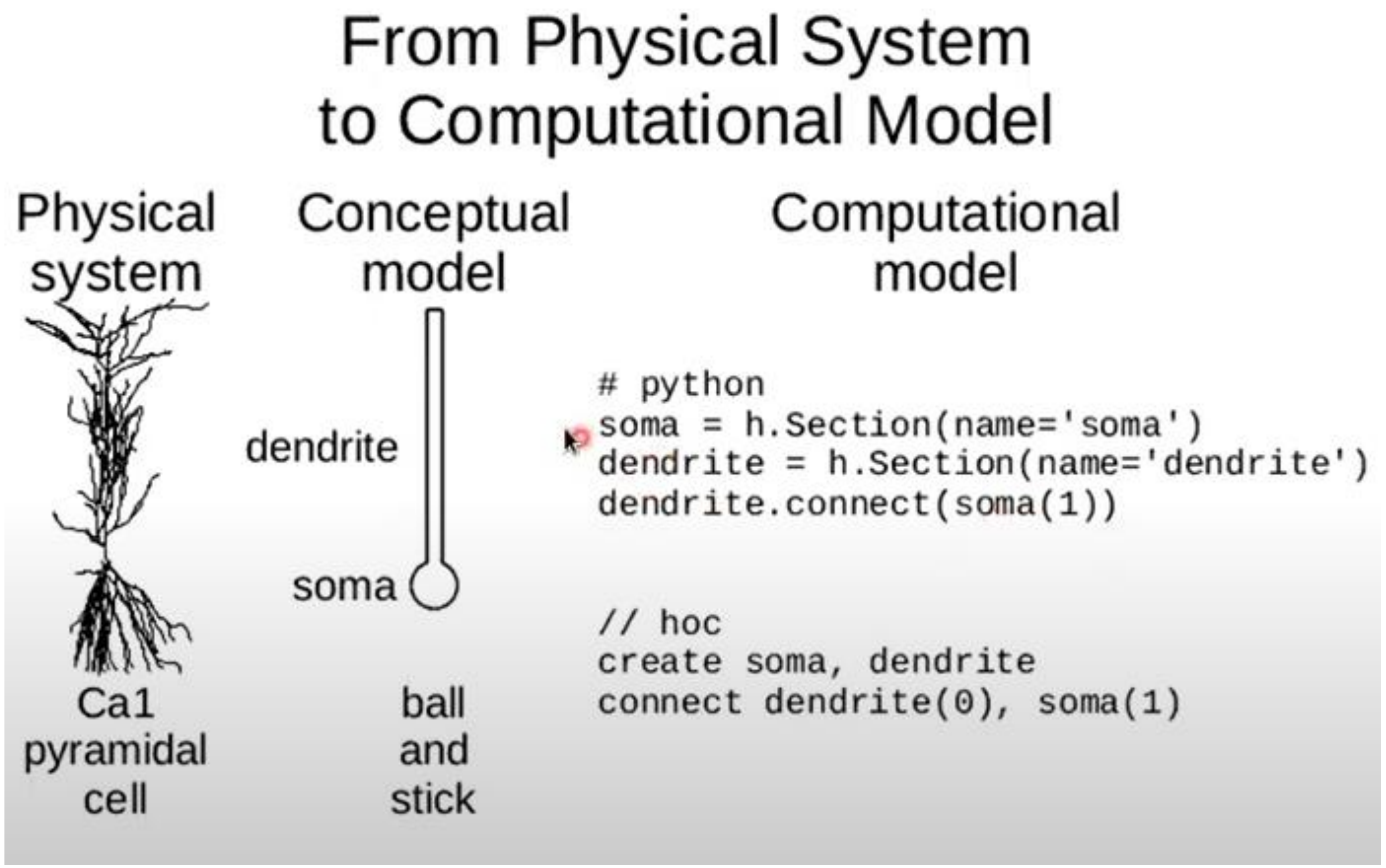
Users who have special interests and expertise are invited to participate in the NEURON project by helping to organize future meetings of the NEURON Users Group, and by participating in collaborative development of documentation, tutorials, and software. We also welcome suggestions for ways to make NEURON a more useful tool for research and teaching.

INSTALL NEURON 8.2 THE NEURON FORUM LATEST NEWS

## Introducción a NEURON / NetPyNE

Buscamos modelos de neuronas / circuitos neuronales donde:

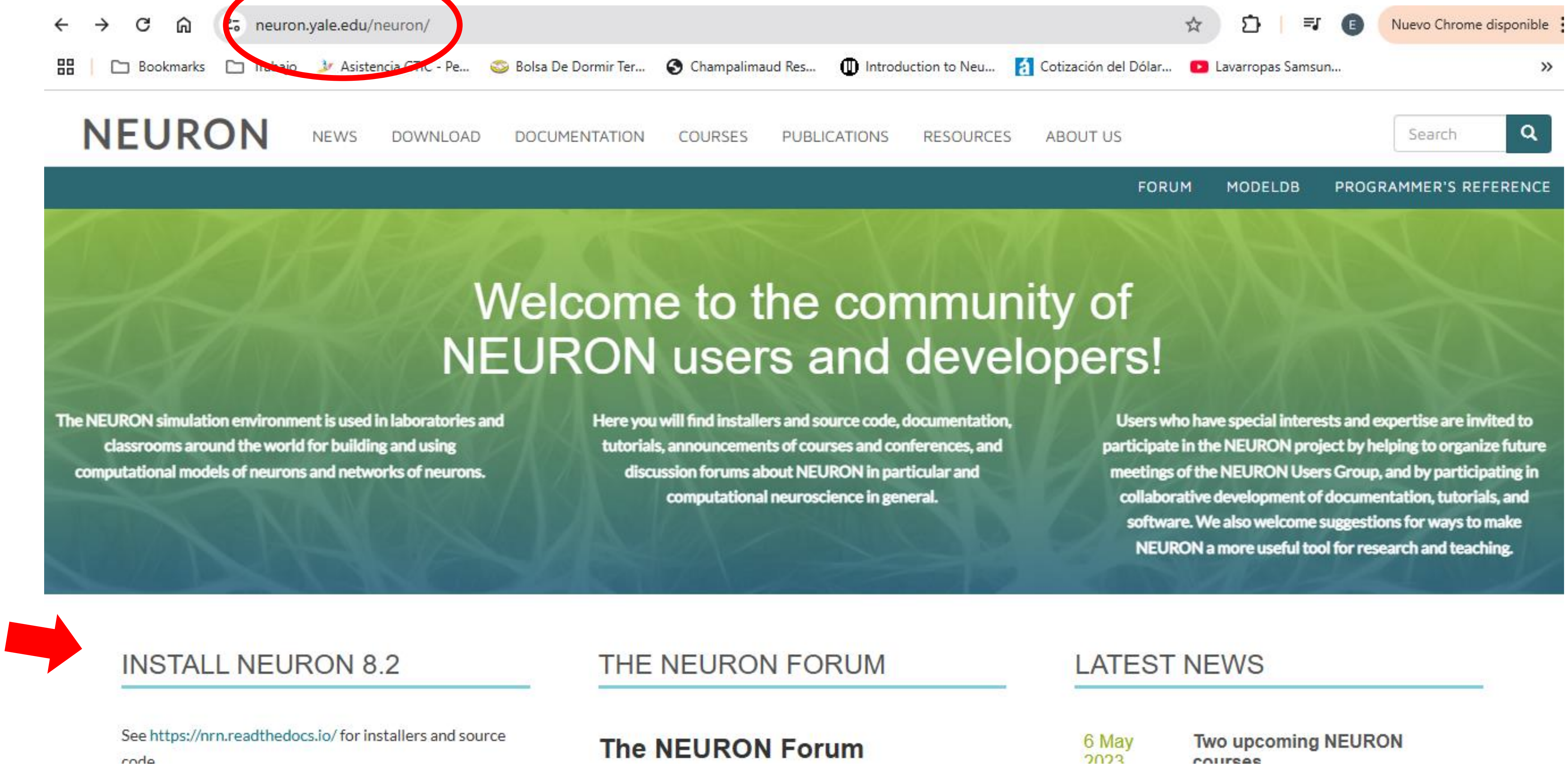
- Hay propiedades morfológicas y biofísicas complejas en juego
- Hay comunicación eléctrica y/o química
- Los modelos están fuertemente asociados a observaciones experimentales



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

// hoc
create soma, dendrite
connect dendrite(0), soma(1)
```

# Introducción a NEURON / NetPyNE

## Python y NEURON instalados



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.

INSTALL NEURON 8.2

See https://nrn.readthedocs.io/ for installers and source code

THE NEURON FORUM

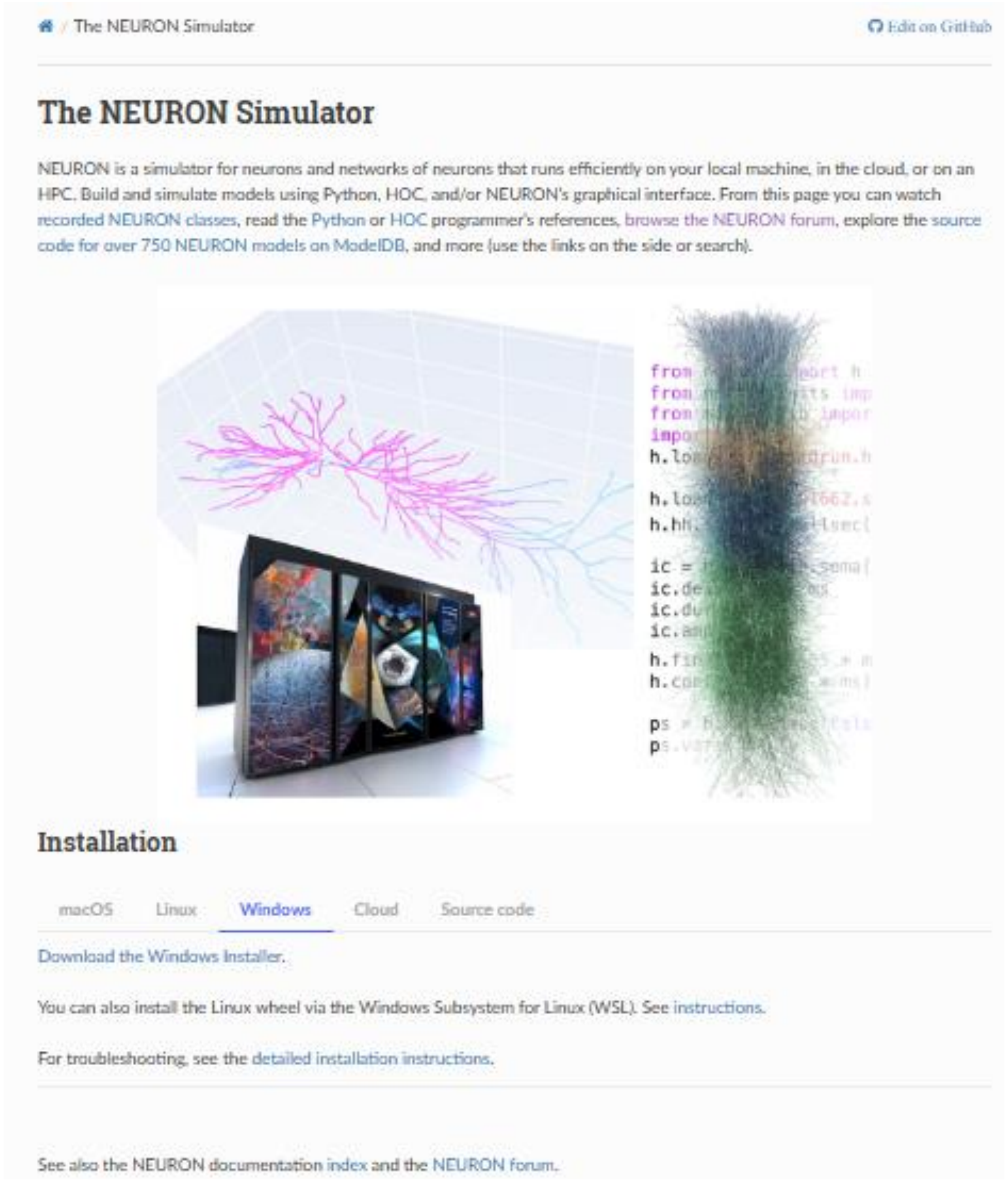
The NEURON Forum

LATEST NEWS

6 May 2023 Two upcoming NEURON courses

# Introducción a NEURON / NetPyNE

Python y NEURON instalados



The NEURON Simulator

Edit on GitHub

## The NEURON Simulator

NEURON is a simulator for neurons and networks of neurons that runs efficiently on your local machine, in the cloud, or on an HPC. Build and simulate models using Python, HOC, and/or NEURON's graphical interface. From this page you can watch recorded NEURON classes, read the Python or HOC programmer's references, browse the NEURON forum, explore the source code for over 750 NEURON models on ModelDB, and more (use the links on the side or search).

## Installation

macOS Linux Windows Cloud Source code

Download the Windows Installer.

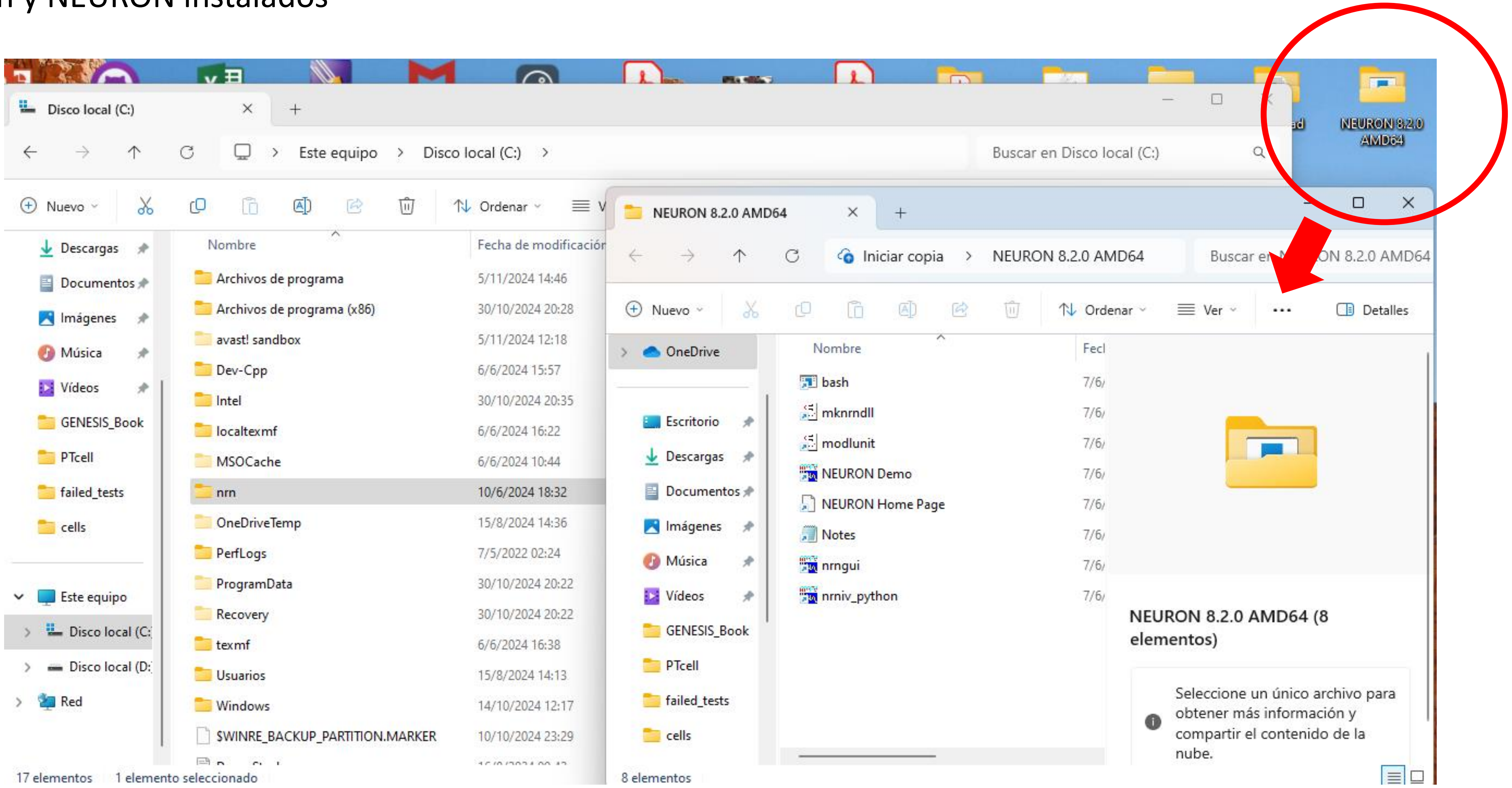
You can also install the Linux wheel via the Windows Subsystem for Linux (WSL). See instructions.

For troubleshooting, see the detailed installation instructions.

See also the NEURON documentation index and the NEURON forum.

# Introducción a NEURON / NetPyNE

## Python y NEURON instalados



Disco local (C:)

Este equipo > Disco local (C:) >

Buscar en Disco local (C:)

Nuevo · Ordenar

| Nombre | Fecha de modificación |
|---|---|
| Archivos de programa | 5/11/2024 14:46 |
| Archivos de programa (x86) | 30/10/2024 20:28 |
| avast! sandbox | 5/11/2024 12:18 |
| Dev-Cpp | 6/6/2024 15:57 |
| Intel | 30/10/2024 20:35 |
| localtexmf | 6/6/2024 16:22 |
| MSOCache | 6/6/2024 10:44 |
| nrn | 10/6/2024 18:32 |
| OneDriveTemp | 15/8/2024 14:36 |
| PerfLogs | 7/5/2022 02:24 |
| ProgramData | 30/10/2024 20:22 |
| Recovery | 30/10/2024 20:22 |
| texmf | 6/6/2024 16:38 |
| Usuarios | 15/8/2024 14:13 |
| Windows | 14/10/2024 12:17 |
| $WINRE_BACKUP_PARTITION.MARKER | 10/10/2024 23:29 |

Descargas · Documentos · Imágenes · Música · Vídeos · GENESIS_Book · PTcell · failed_tests · cells · Este equipo · Disco local (C:) · Disco local (D:) · Red

17 elementos 1 elemento seleccionado

NEURON 8.2.0 AMD64

Iniciar copia > NEURON 8.2.0 AMD64

Buscar en NEURON 8.2.0 AMD64

Nuevo · Ordenar · Ver · ··· · Detalles

OneDrive · Escritorio · Descargas · Documentos · Imágenes · Música · Vídeos · GENESIS_Book · PTcell · failed_tests · cells

| Nombre | Fecl |
|---|---|
| bash | 7/6/ |
| mknrndll | 7/6/ |
| modlunit | 7/6/ |
| NEURON Demo | 7/6/ |
| NEURON Home Page | 7/6/ |
| Notes | 7/6/ |
| nrngui | 7/6/ |
| nrniv_python | 7/6/ |

NEURON 8.2.0 AMD64 (8 elementos)

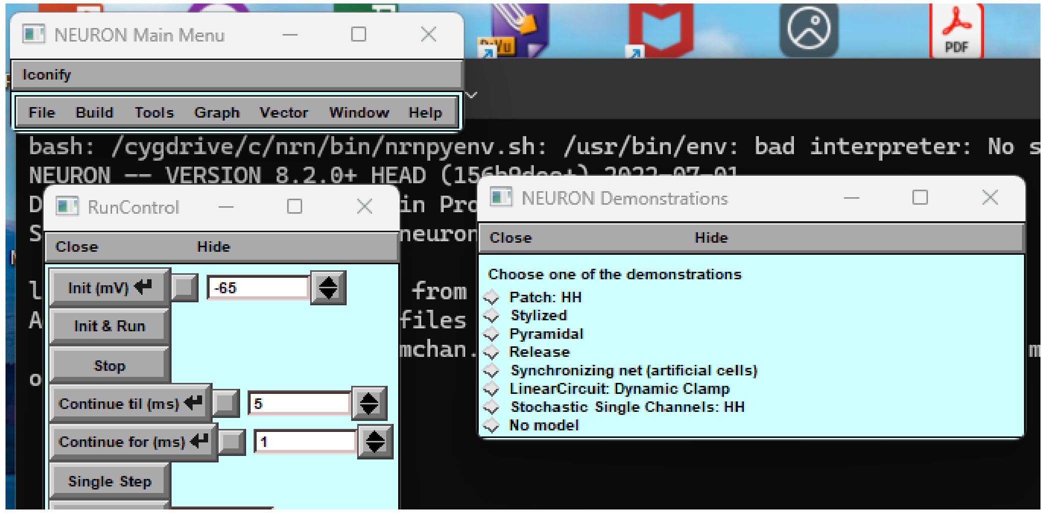
Seleccione un único archivo para obtener más información y compartir el contenido de la nube.

8 elementos

# Introducción a NEURON / NetPyNE

-> Anaconda Python terminal + “neuron”: Ejecuta y carga NEURON GUI (graphical user interface)

-> NEURON Folder: Ejecutar “NEURON Demo” -> Carga modelos ya implementados (Demostraciones)



```
bash: /cygdrive/c/nrn/bin/nrnpyenv.sh: /usr/bin/env: bad interpreter: No s
NEURON -- VERSION 8.2.0+ HEAD (15
D                                in Pro
S                                neuron

l                                 from
A                                files
                                mchan.
o
```

Choose one of the demonstrations
- Patch: HH
- Stylized
- Pyramidal
- Release
- Synchronizing net (artificial cells)
- LinearCircuit: Dynamic Clamp
- Stochastic Single Channels: HH
- No model

Mas fácil para construir modelos rápidos y tener feedback

$\uparrow$

-> NEURON as a Python module. System prompt + “python” . Then, >>> from neuron import h, gui

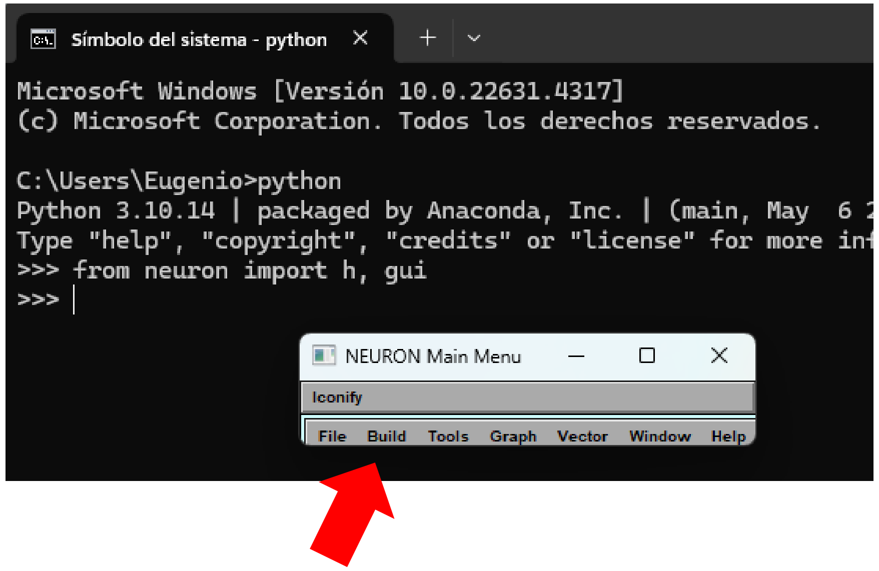
$\downarrow$

Acceso desde Python a toda la maquinaria computacional de NEURON

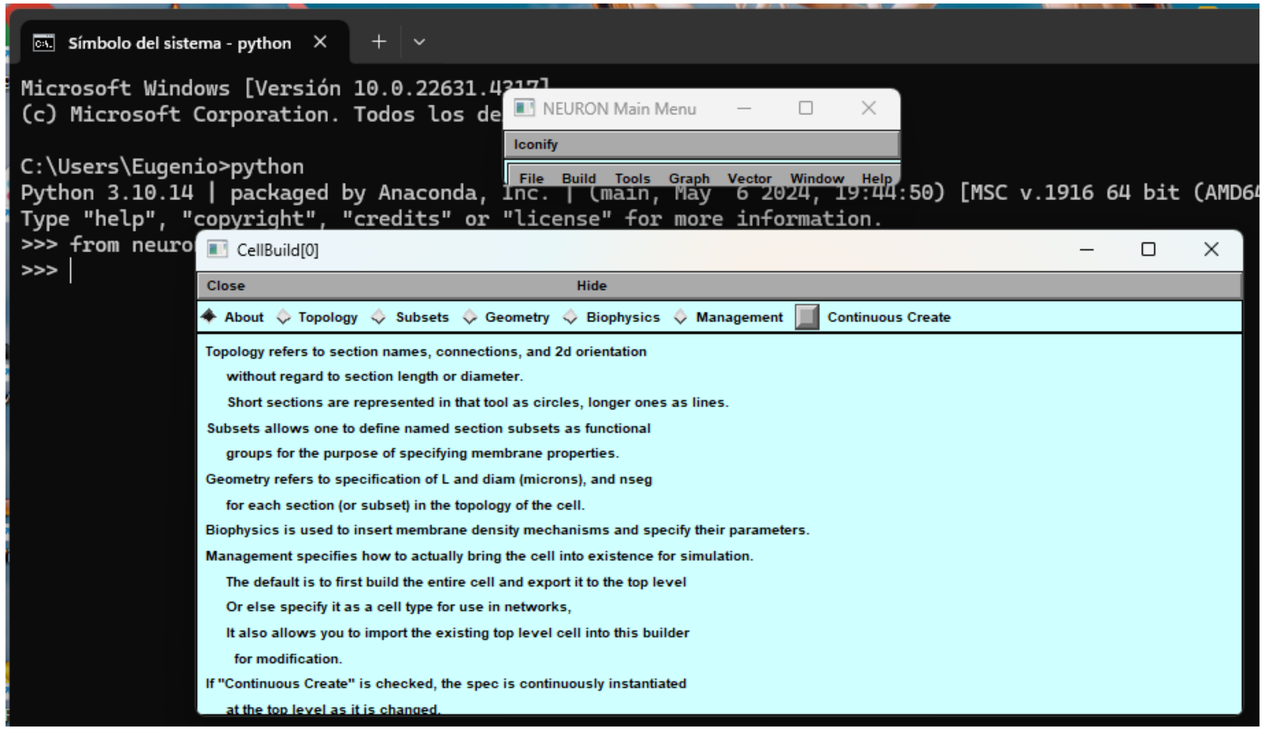
# Introducción a NEURON / NetPyNE

-> Construyendo un modelo simple de compartimiento único (soma)

- System prompt + “python” . Then, >>> from neuron import h, gui
- Hacer un modelo de célula: Main menú -> Build -> CellBuilder



```
Microsoft Windows [Versión 10.0.22631.4317]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\Eugenio>python
Python 3.10.14 | packaged by Anaconda, Inc. | (main, May  6 2
Type "help", "copyright", "credits" or "license" for more inf
>>> from neuron import h, gui
>>> |
```



```
Microsoft Windows [Versión 10.0.22631.4317]
(c) Microsoft Corporation. Todos los de

C:\Users\Eugenio>python
Python 3.10.14 | packaged by Anaconda, Inc. | (main, May  6 2024, 19:44:50) [MSC v.1916 64 bit (AMD64
Type "help", "copyright", "credits" or "license" for more information.
>>> from neuro
>>> |
```

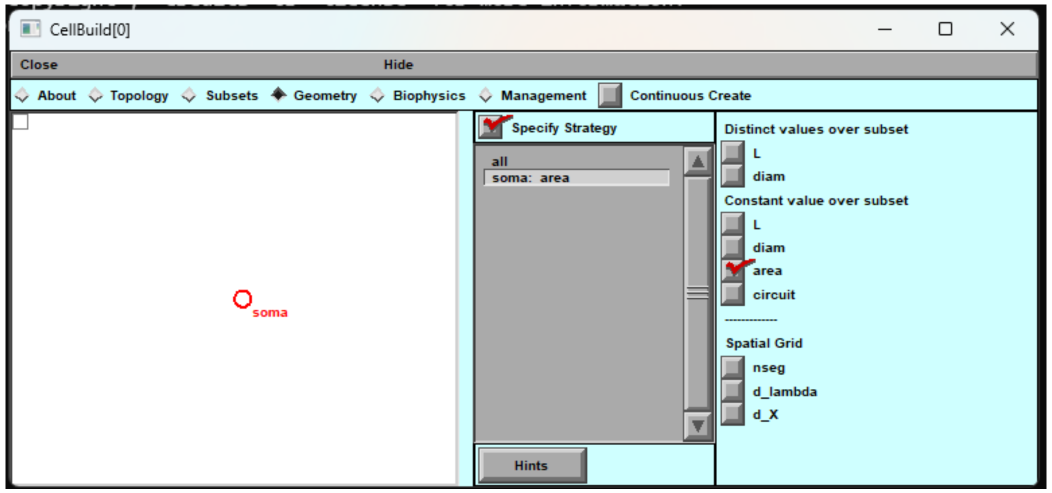
Topology refers to section names, connections, and 2d orientation
  without regard to section length or diameter.
  Short sections are represented in that tool as circles, longer ones as lines.
Subsets allows one to define named section subsets as functional
  groups for the purpose of specifying membrane properties.
Geometry refers to specification of L and diam (microns), and nseg
  for each section (or subset) in the topology of the cell.
Biophysics is used to insert membrane density mechanisms and specify their parameters.
Management specifies how to actually bring the cell into existence for simulation.
  The default is to first build the entire cell and export it to the top level
  Or else specify it as a cell type for use in networks,
  It also allows you to import the existing top level cell into this builder
   for modification.
If "Continuous Create" is checked, the spec is continuously instantiated
  at the top level as it is changed.

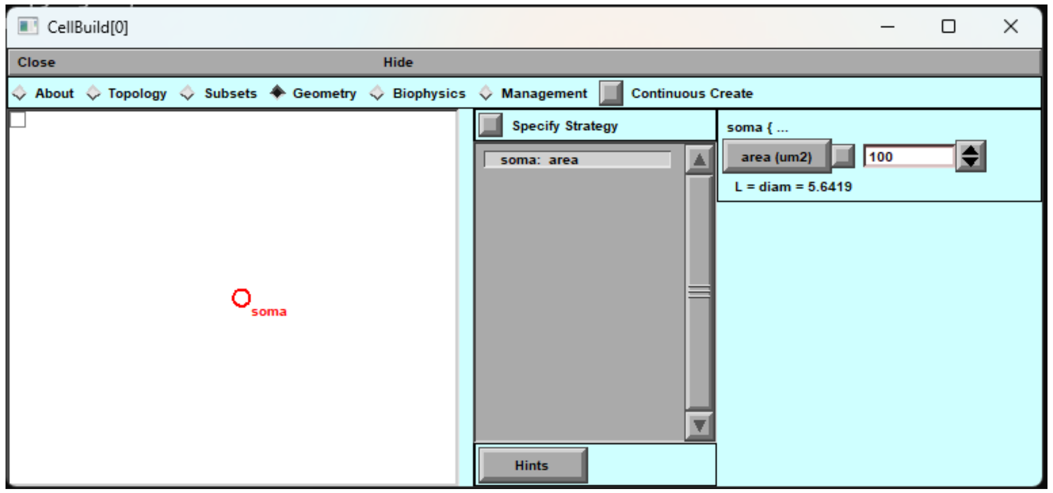
# Introducción a NEURON / NetPyNE

-> Construyendo un modelo simple de compartimiento único (soma)

- Topology -> Hay un elemento/sección (no está creado todavía, está “pedido”)

- Geometry -> Cliqueamos en “soma”. Cliqueamos en la propiedad que queramos definir (por ejemplo, área). Para modificarla, (des)cliquear “Specify Strategy”



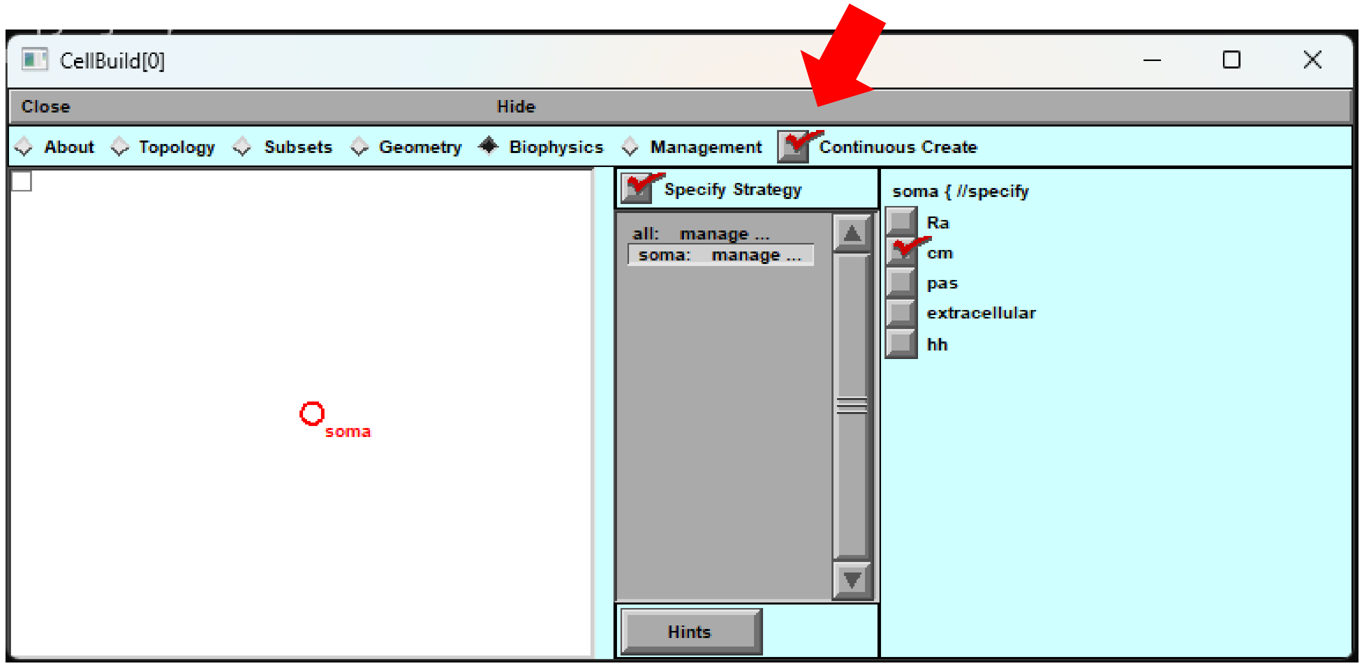


soma { ...
area (um2) 100
L = diam = 5.6419

# Introducción a NEURON / NetPyNE

-> Construyendo un modelo simple de compartimiento único (soma)

- Biophysics -> Cliqueando “cm” -> crea sólo la membrana como un capacitor
- Continuous create -> Cliqueando, instanciamos en memoria el modelo especificado

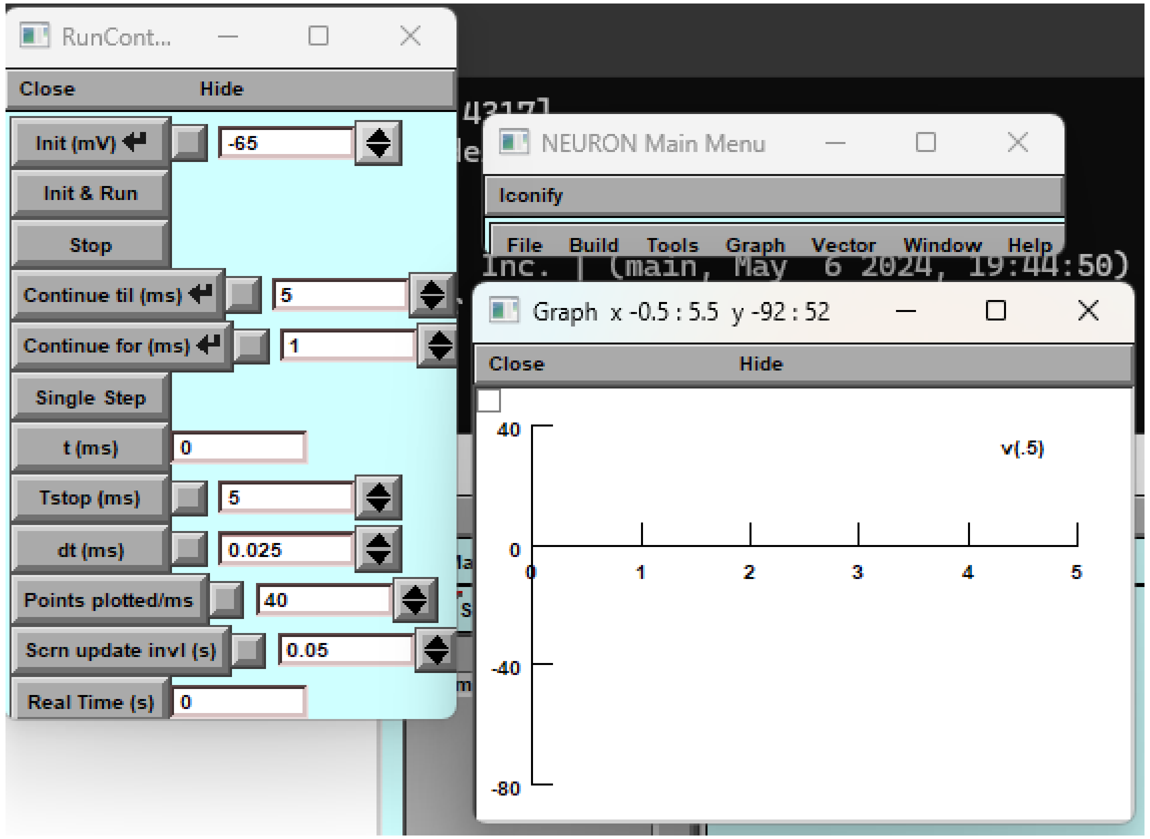


# Introducción a NEURON / NetPyNE

-> Construyendo un modelo simple de compartimiento único (soma)

- Establecer una simulación: Main menu -> Tools -> RunControl

- Visualizar un resultado: Main menu -> Graph -> Voltage axis



Init (mV) -65
Continue til (ms) 5
Continue for (ms) 1
t (ms) 0
Tstop (ms) 5
dt (ms) 0.025
Points plotted/ms 40
Scrn update invl (s) 0.05
Real Time (s) 0

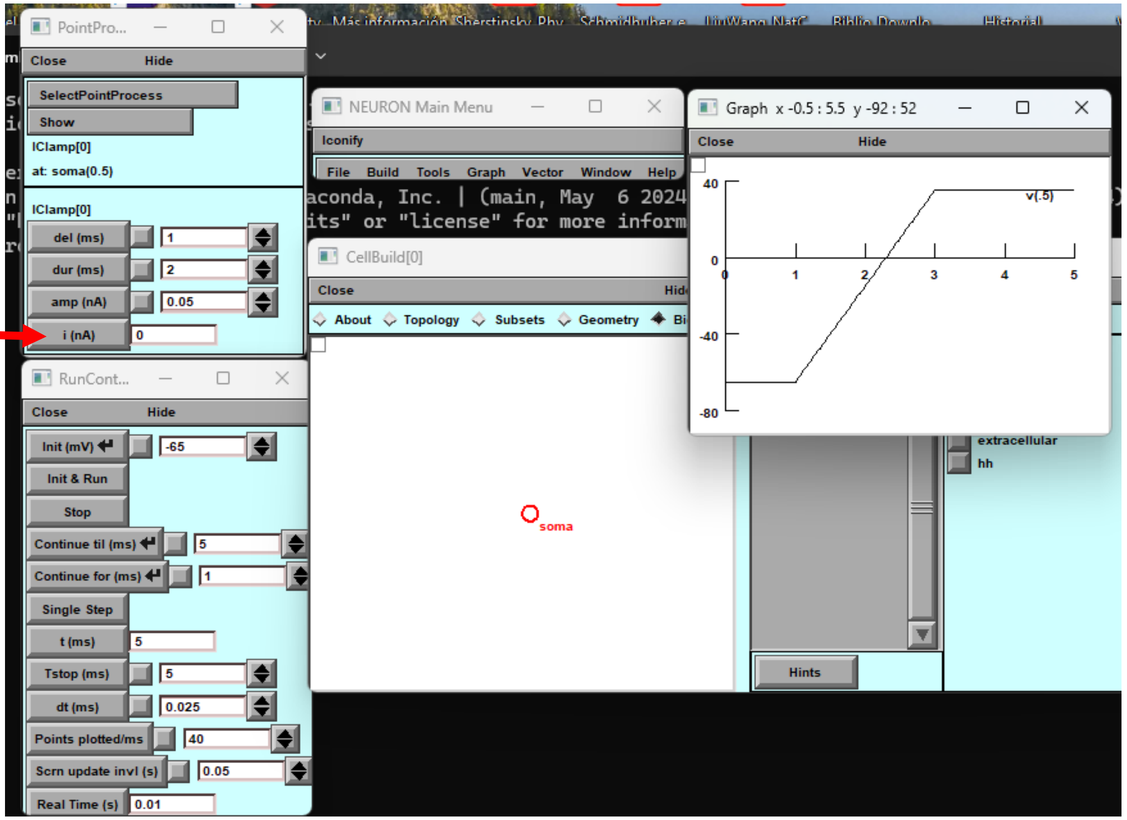
```
Inc. | (main, May  6 2024, 19:44:50)
```

# Introducción a NEURON / NetPyNE

-> Construyendo un modelo simple de compartimiento único (soma)

- Estimular a la célula: Main menu -> Tools -> Point Processes -> Managers -> Point manager
- Seleccionar “IClamp” en SelectPointProcess y establecer parámetros

Nombre de la variable allocada por el objeto



IClamp[0]
at: soma(0.5)

IClamp[0]

```
del (ms)     1
dur (ms)     2
amp (nA)     0.05
i (nA)       0
```

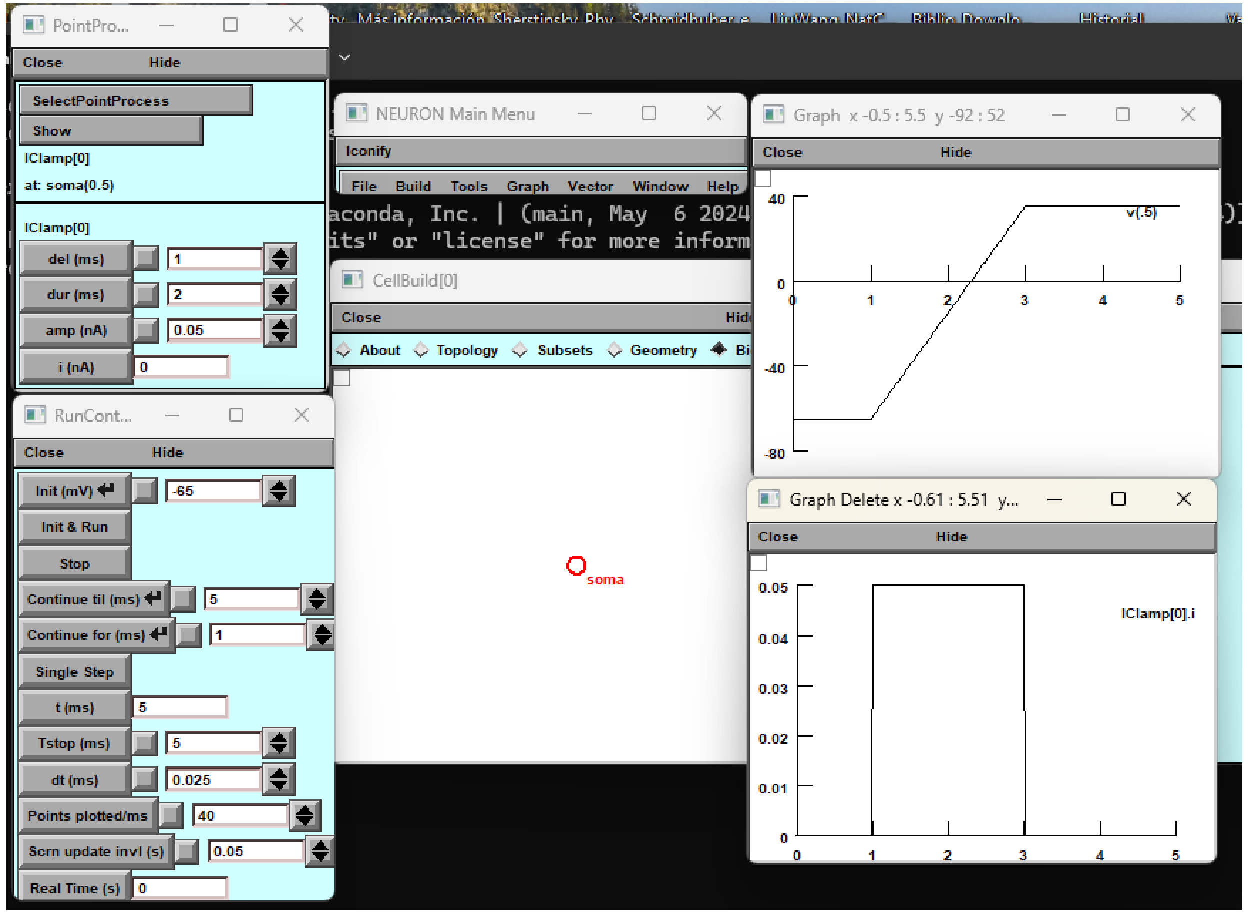
```
Init (mV)              -65
Continue til (ms)      5
Continue for (ms)      1
t (ms)                 5
Tstop (ms)             5
dt (ms)                0.025
Points plotted/ms      40
Scrn update invl (s)   0.05
Real Time (s)          0.01
```

```
aconda, Inc. | (main, May  6 2024
its" or "license" for more inform
```

# Introducción a NEURON / NetPyNE

-> Construyendo un modelo simple de compartimiento único (soma)

- Visualizar la estimulación: Main menu -> Graph -> Current axis -> Plot what? –> Iclamp[0].i



IClamp[0]
at: soma(0.5)

IClamp[0]

| | |
|---|---|
| del (ms) | 1 |
| dur (ms) | 2 |
| amp (nA) | 0.05 |
| i (nA) | 0 |

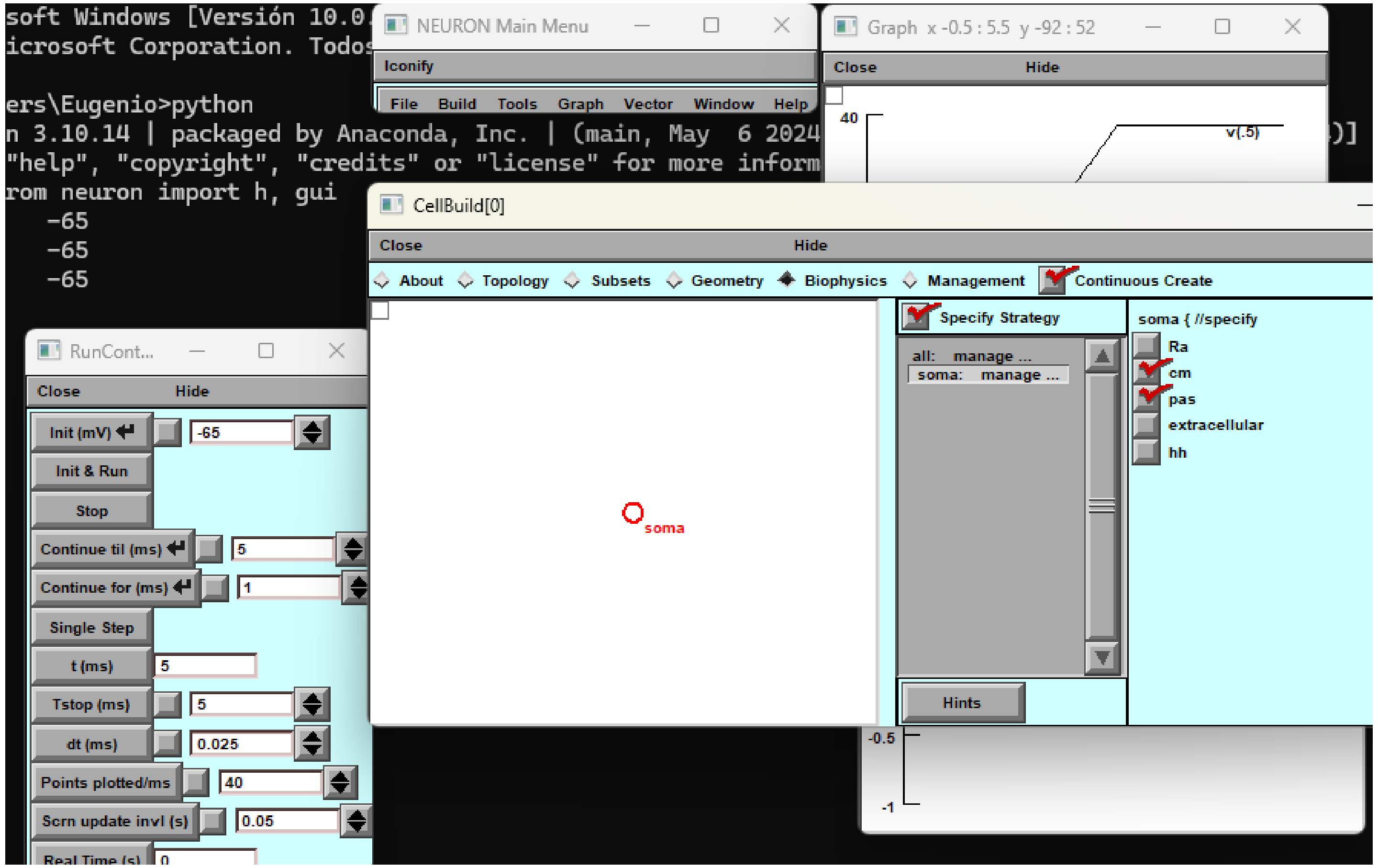
| | |
|---|---|
| Init (mV) | -65 |
| Continue til (ms) | 5 |
| Continue for (ms) | 1 |
| t (ms) | 5 |
| Tstop (ms) | 5 |
| dt (ms) | 0.025 |
| Points plotted/ms | 40 |
| Scrn update invl (s) | 0.05 |
| Real Time (s) | 0 |

```
aconda, Inc. | (main, May  6 2024
its" or "license" for more inform
```

# Introducción a NEURON / NetPyNE

-> Construyendo un modelo simple de compartimiento único (soma)

- Ponemos canales pasivos a la membrana: Main menu -> Tools -> CellBuilder -> Biophysics -> Cliquear “pas”



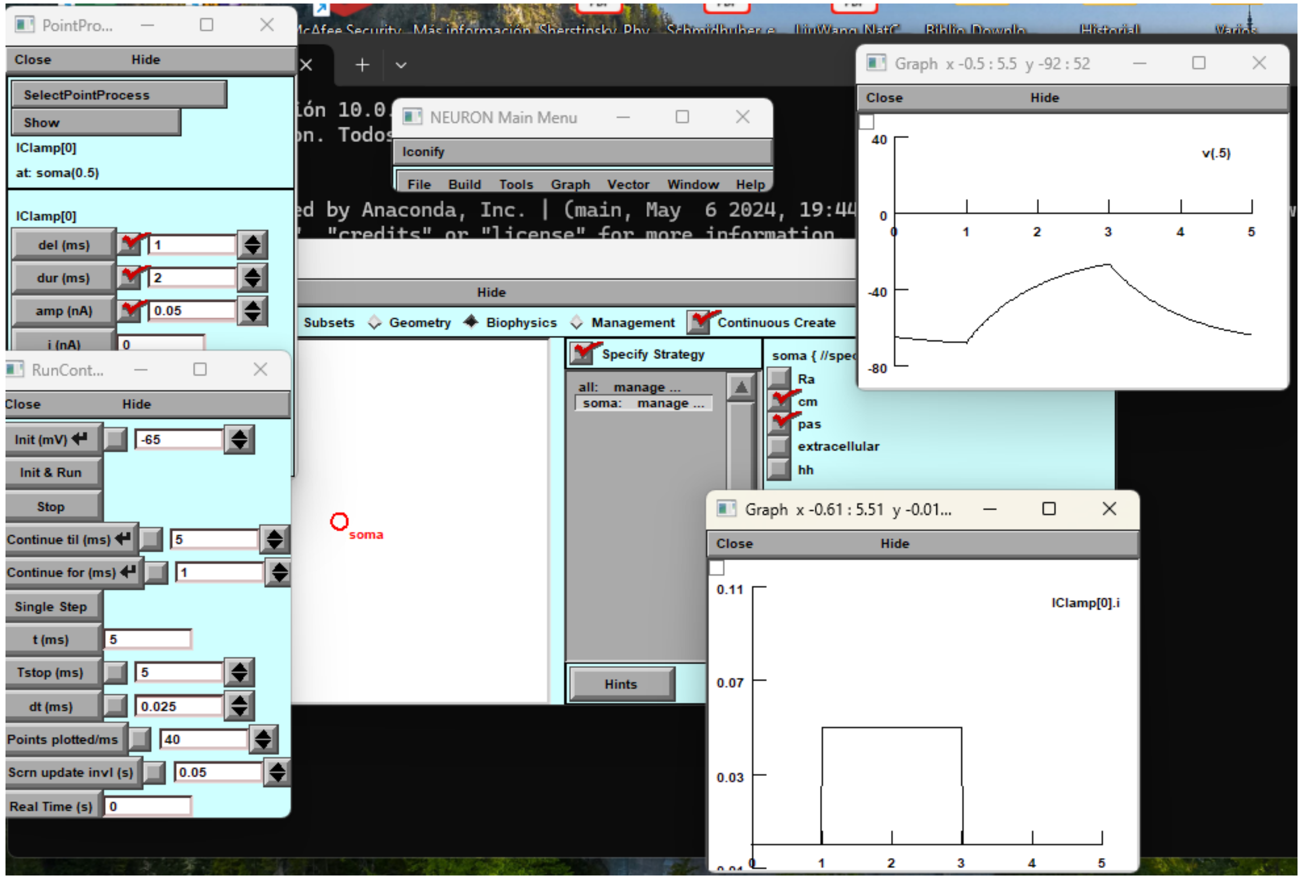
```
soft Windows [Versión 10.0.
icrosoft Corporation. Todos

ers\Eugenio>python
n 3.10.14 | packaged by Anaconda, Inc. | (main, May  6 2024
"help", "copyright", "credits" or "license" for more inform
rom neuron import h, gui
   -65
   -65
   -65
```

| | | |
|---|---|---|
| Init (mV) | -65 |
| Init & Run | | |
| Stop | | |
| Continue til (ms) | 5 |
| Continue for (ms) | 1 |
| Single Step | | |
| t (ms) | 5 |
| Tstop (ms) | 5 |
| dt (ms) | 0.025 |
| Points plotted/ms | 40 |
| Scrn update invl (s) | 0.05 |
| Real Time (s) | 0 |

# Introducción a NEURON / NetPyNE

-> Construyendo un modelo simple de compartimiento único (soma)

- Ponemos canales pasivos a la membrana: Main menu -> Tools -> CellBuilder -> Biophysics -> Cliquear “pas”



IClamp[0]
at: soma(0.5)

IClamp[0]

| | |
|---|---|
| del (ms) | 1 |
| dur (ms) | 2 |
| amp (nA) | 0.05 |
| i (nA) | 0 |

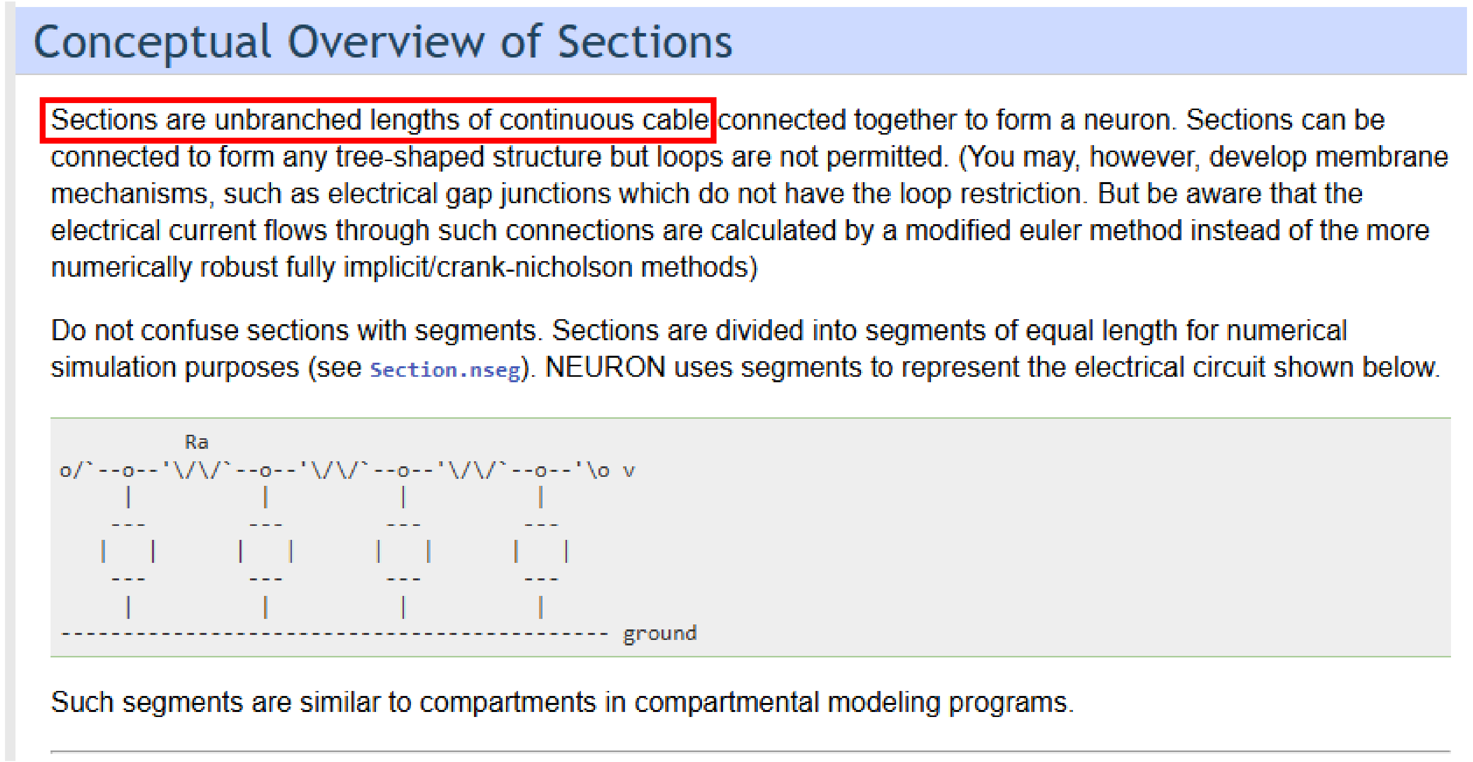
| | |
|---|---|
| Init (mV) | -65 |
| Continue til (ms) | 5 |
| Continue for (ms) | 1 |
| t (ms) | 5 |
| Tstop (ms) | 5 |
| dt (ms) | 0.025 |
| Points plotted/ms | 40 |
| Scrn update invl (s) | 0.05 |
| Real Time (s) | 0 |

```
ión 10.0
on. Todos
ed by Anaconda, Inc. | (main, May  6 2024, 19:44
"credits" or "license" for more information
```

## Introducción a NEURON / NetPyNE

-> Construyendo un modelo con **extensión espacial**

- Elemento central -> Sección (h.Section)



### Conceptual Overview of Sections

Sections are unbranched lengths of continuous cable connected together to form a neuron. Sections can be connected to form any tree-shaped structure but loops are not permitted. (You may, however, develop membrane mechanisms, such as electrical gap junctions which do not have the loop restriction. But be aware that the electrical current flows through such connections are calculated by a modified euler method instead of the more numerically robust fully implicit/crank-nicholson methods)

Do not confuse sections with segments. Sections are divided into segments of equal length for numerical simulation purposes (see `Section.nseg`). NEURON uses segments to represent the electrical circuit shown below.

```
          Ra
o/`--o--'\/\/`--o--'\/\/`--o--'\/\/`--o--'\o v
     |          |          |          |
    ---        ---        ---        ---
   |   |      |   |      |   |      |   |
    ---        ---        ---        ---
     |          |          |          |
-------------------------------------------- ground
```

Such segments are similar to compartments in compartmental modeling programs.

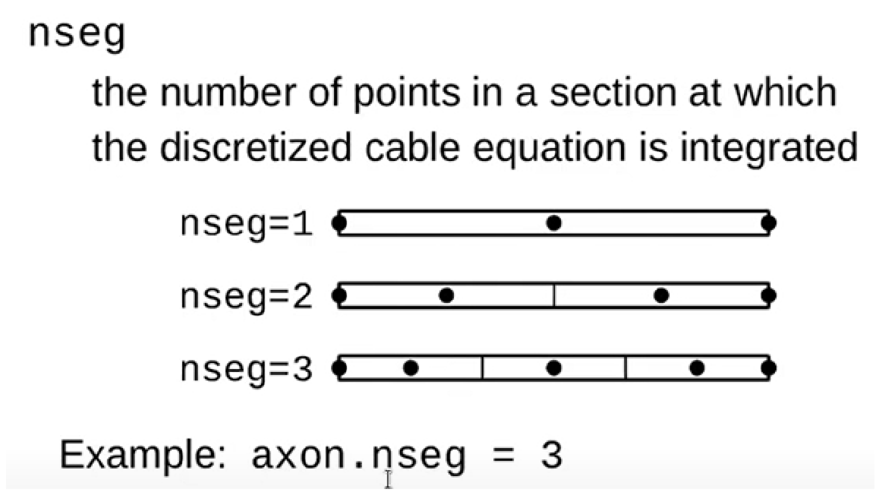
- Normalized distance (0-1) + Range variables (v, diam, cm, g_pas or pas.g, etc) –> variables y parámetros que pueden variar a lo largo de la distancia normalizada -> asociados a cada nodo

```
secname(range).rangevar

dend(0.5).v
```

# Introducción a NEURON / NetPyNE

-> Construyendo un modelo con **extensión espacial**

- Resolución espacial de la sección -> Se controla/define con una propiedad



nseg

the number of points in a section at which
the discretized cable equation is integrated

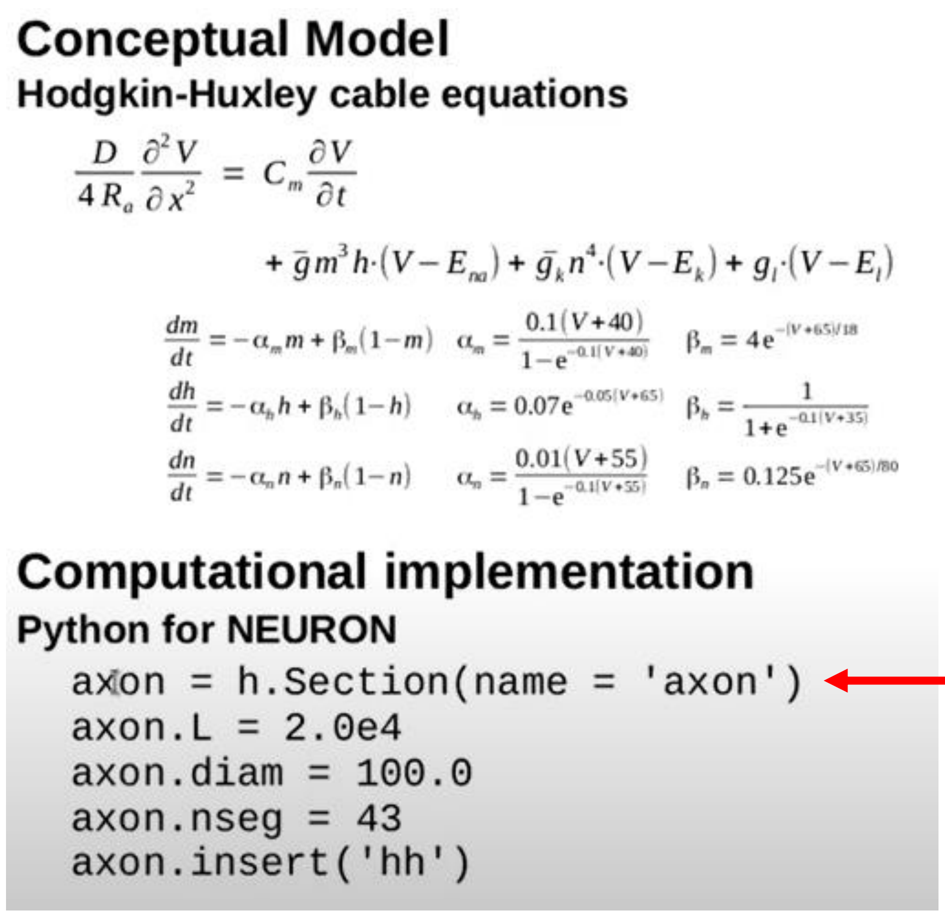
Example: `axon.nseg = 3`

-> Usar números impares (así el nodo central está representado)

## Introducción a NEURON / NetPyNE

-> Construyendo un modelo con **extensión espacial**

- Ejemplo



**Conceptual Model**

**Hodgkin-Huxley cable equations**

$$\frac{D}{4R_a}\frac{\partial^2 V}{\partial x^2} = C_m \frac{\partial V}{\partial t}$$

$$+ \bar{g} m^3 h \cdot (V - E_{na}) + \bar{g}_k n^4 \cdot (V - E_k) + g_l \cdot (V - E_l)$$

$$\frac{dm}{dt} = -\alpha_m m + \beta_m (1-m) \quad \alpha_m = \frac{0.1(V+40)}{1-e^{-0.1(V+40)}} \quad \beta_m = 4e^{-(V+65)/18}$$

$$\frac{dh}{dt} = -\alpha_h h + \beta_h (1-h) \quad \alpha_h = 0.07e^{-0.05(V+65)} \quad \beta_h = \frac{1}{1+e^{-0.1(V+35)}}$$

$$\frac{dn}{dt} = -\alpha_n n + \beta_n (1-n) \quad \alpha_n = \frac{0.01(V+55)}{1-e^{-0.1(V+55)}} \quad \beta_n = 0.125e^{-(V+65)/80}$$

**Computational implementation**

**Python for NEURON**

```
axon = h.Section(name = 'axon')
axon.L = 2.0e4
axon.diam = 100.0
axon.nseg = 43
axon.insert('hh')
```

Implementa un trozo de membrana (capacitor)

## Introducción a NEURON / NetPyNE

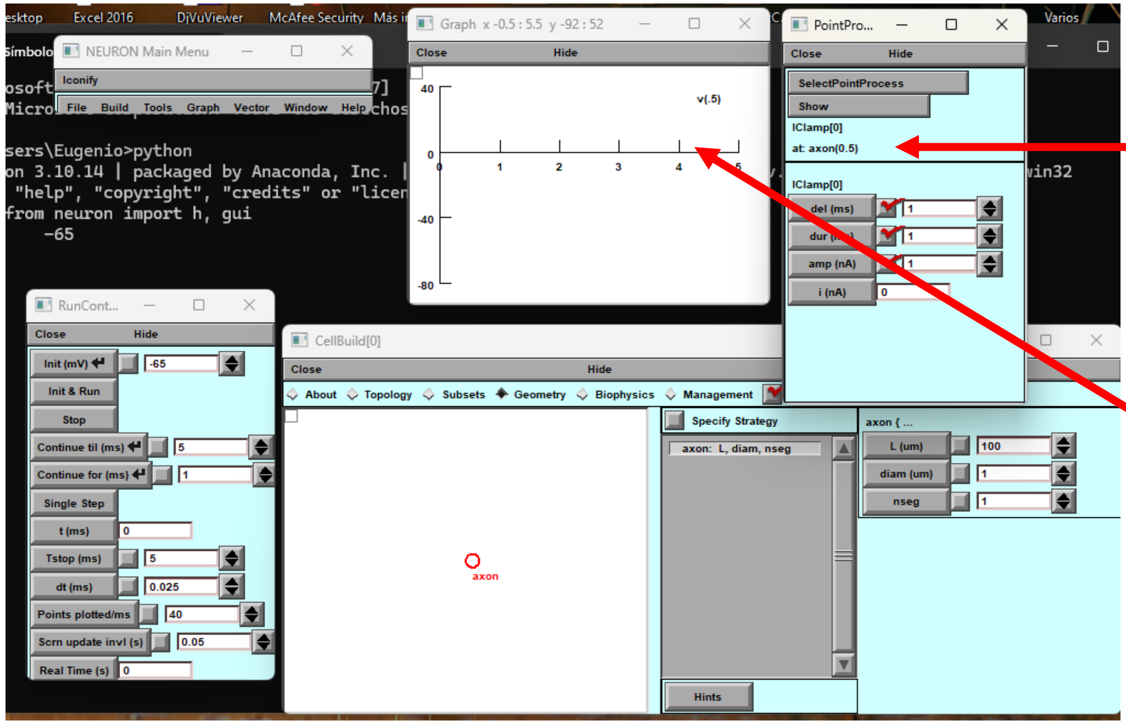
-> Construyendo un modelo con **extensión espacial**

- En GUI: Main menu -> Build / CellBuilder
  - Topology -> Cambiar nombre a “axon” (Basename -> axon)
  - Geometry -> Cliquear “axon”. Seleccionar “L”, “diam”, “nseg” y descliquer “Specify strategy”
  - Poner 20000, 100, 43.
  - Biophysics -> Cliquear “axon”. Seleccionar “Ra”, “cm”, “hh” y descliquer “Specify strategy”
  - Dejar valores por defecto
  - “Continuous Create”

# Introducción a NEURON / NetPyNE

-> Construyendo un modelo con **extensión espacial**

- En GUI: Main menu -> Tools / RunControl
- En GUI: Main menu -> Graph / Voltage axis
- En GUI: Main menu -> Tools / Point Processes /Managers / Point Manager -> Set IClamp & specify (1, 1, 1)



```
sers\Eugenio>python
on 3.10.14 | packaged by Anaconda, Inc. |
 "help", "copyright", "credits" or "licen
from neuron import h, gui
   -65
```

Está inyectando en el medio del axón:

-> Seleccionar "Show" -> "Shape" -> Mover el punto

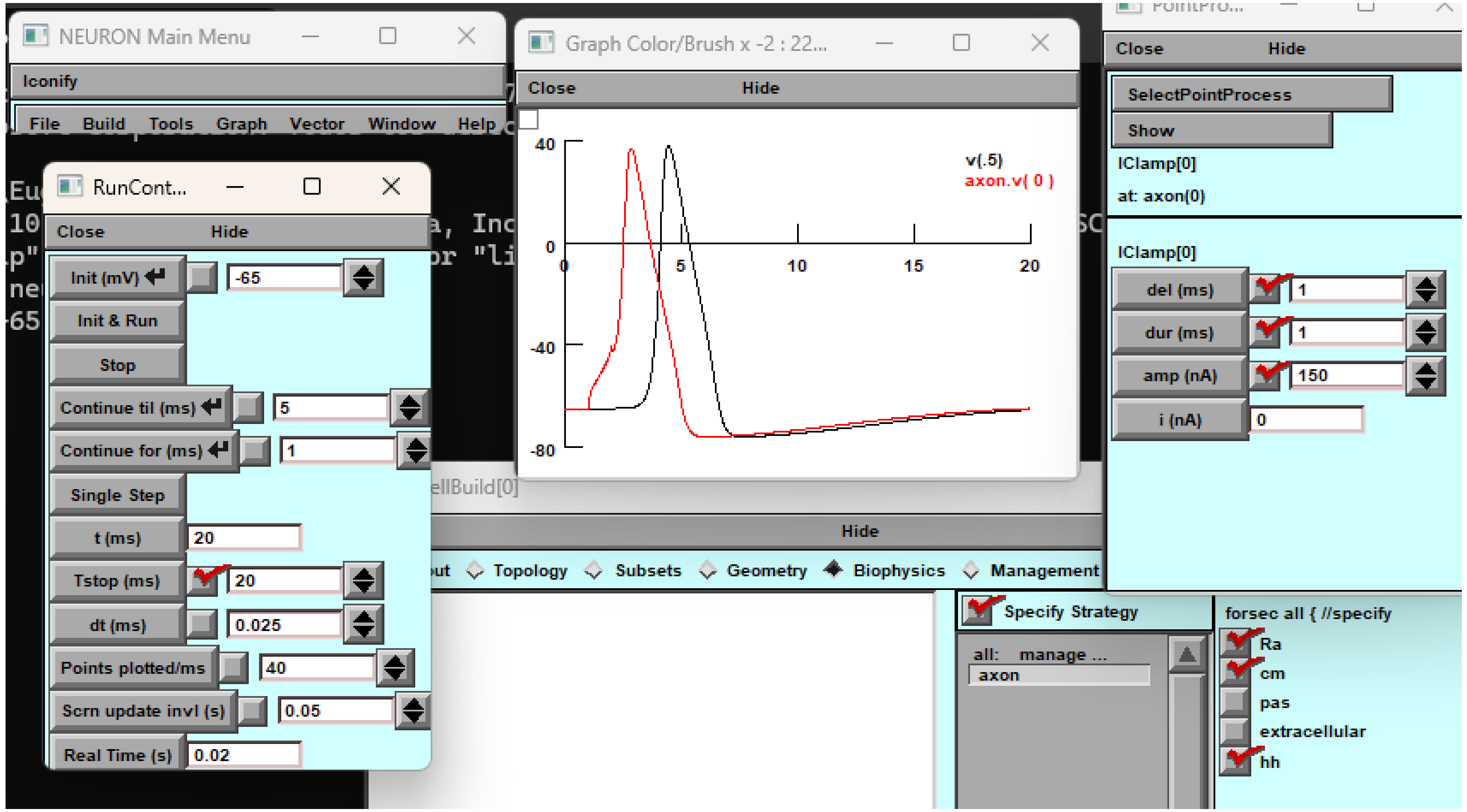
Agregar el potencial en otras ubicaciones

-> Plot what? -> axon.v(0) y ponerle otro color

# Introducción a NEURON / NetPyNE

-> Construyendo un modelo con **extensión espacial**

- IClamp -> poner 150 nA
- RunControl -> Tstop 50 ms



IClamp[0]
at: axon(0)

IClamp[0]

| | |
|---|---|
| del (ms) | 1 |
| dur (ms) | 1 |
| amp (nA) | 150 |
| i (nA) | 0 |

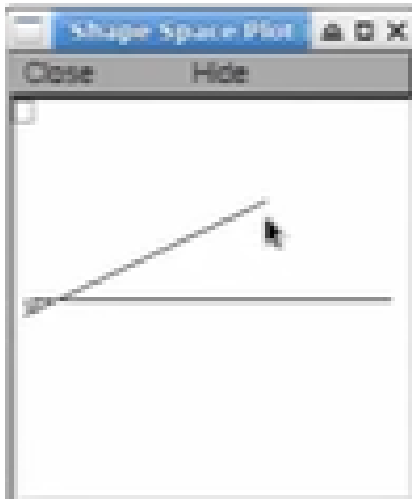
| | |
|---|---|
| Init (mV) | -65 |
| Continue til (ms) | 5 |
| Continue for (ms) | 1 |
| t (ms) | 20 |
| Tstop (ms) | 20 |
| dt (ms) | 0.025 |
| Points plotted/ms | 40 |
| Scrn update invl (s) | 0.05 |
| Real Time (s) | 0.02 |

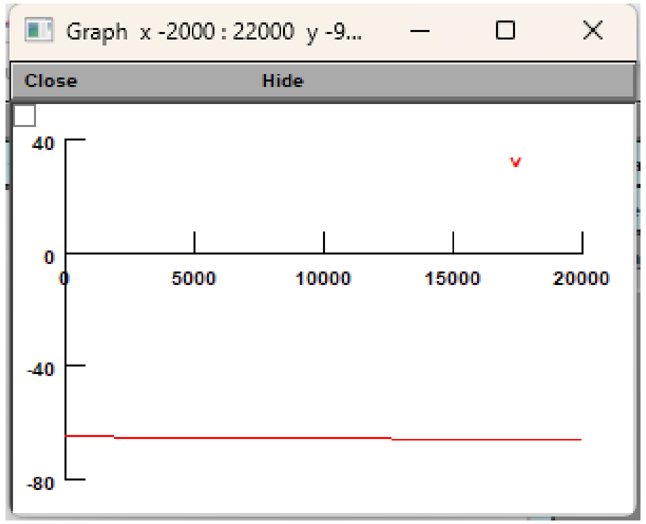
forsec all { //specify
[✓ Ra]
[✓ cm]
pas
extracellular
[✓ hh]

# Introducción a NEURON / NetPyNE

-> Construyendo un modelo con **extensión espacial**

- Crear un gráfico del potencial a lo largo de la sección
- Main menu -> Graph -> Shape Plot
- Menu del gráfico -> Especificar SpacePlot
- Cliquear desde el extremo izquierdo hasta el derecho en la representación de la sección
- Abre un gráfico donde el eje x es la distancia a lo largo de la sección

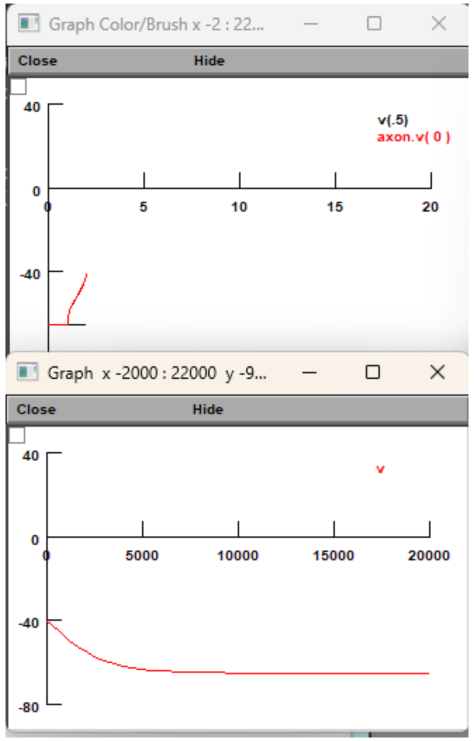




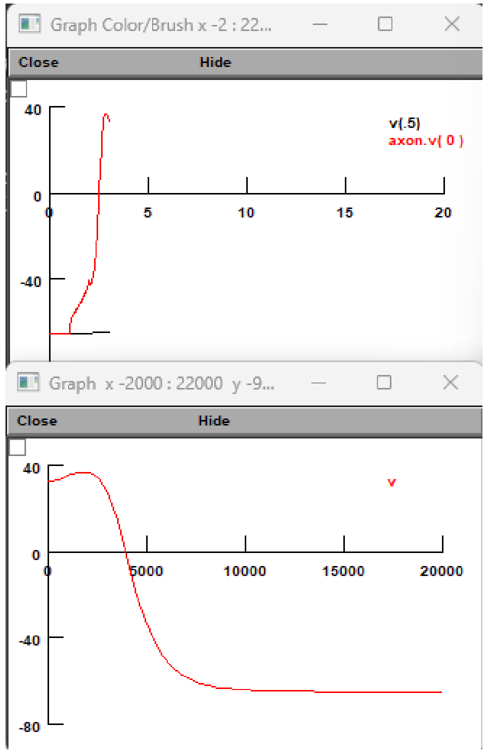
## Introducción a NEURON / NetPyNE

-> Construyendo un modelo con **extensión espacial**

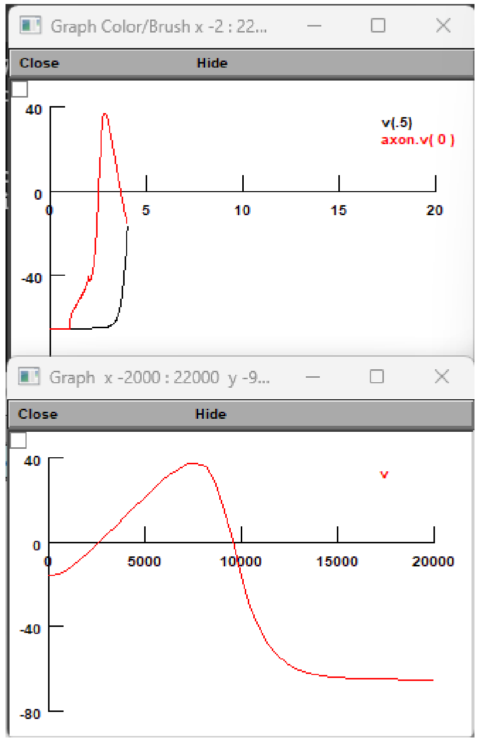
- En Run Control, puedo ir cliqueando “Continue for 1ms” y el gráfico se va actualizando con la solución



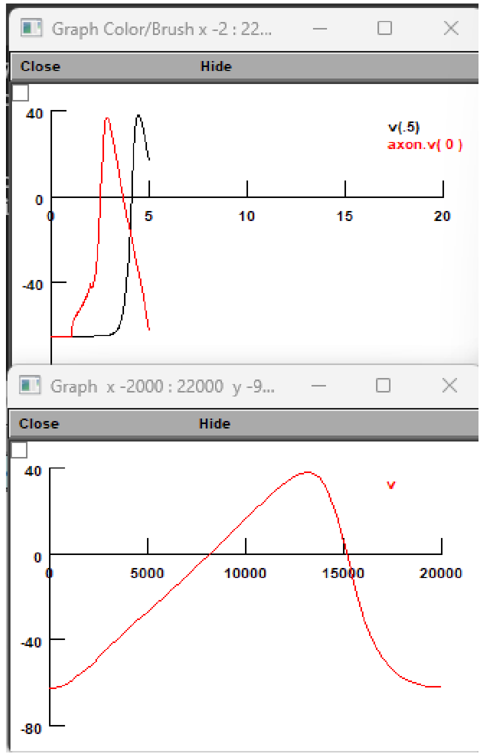
A los 2 ms



A los 3 ms



A los 4 ms



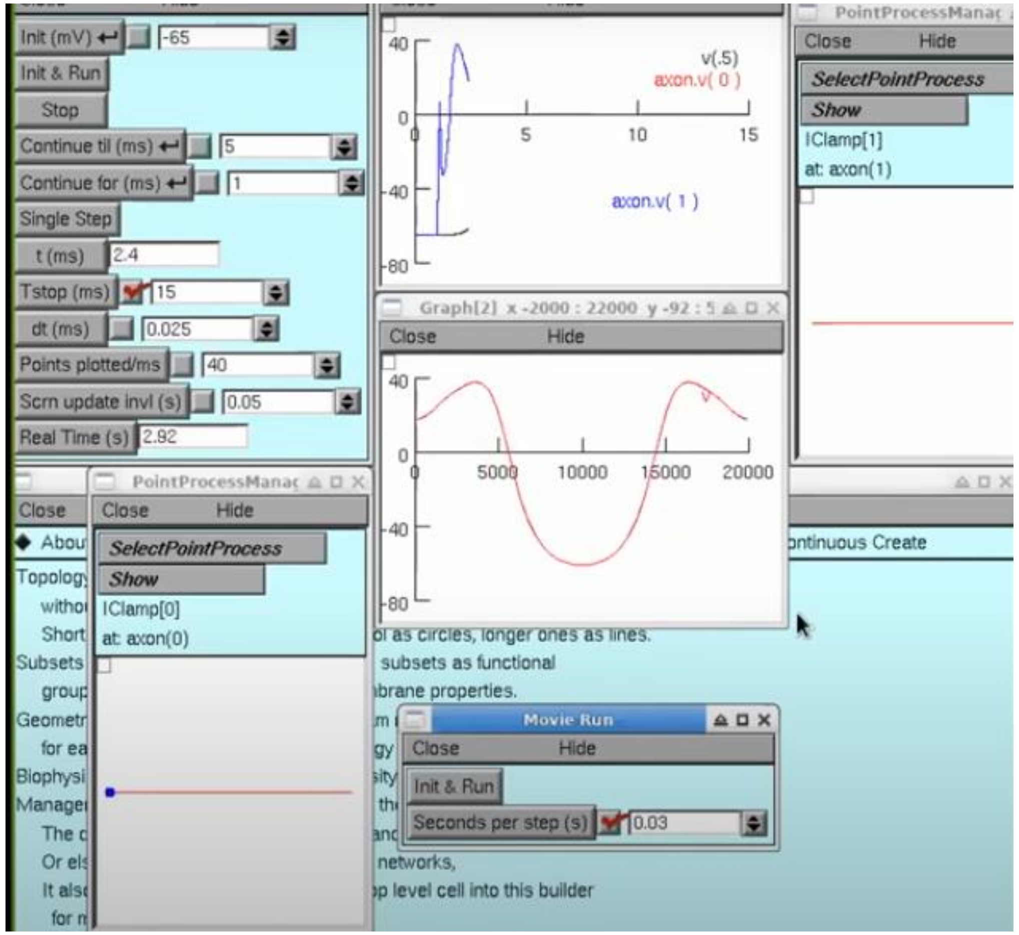
A los 5 ms

- Tools -> Movie Run -> Genera una versión mas lenta de lo anterior

## Introducción a NEURON / NetPyNE

-> Construyendo un modelo con **extensión espacial**

- Hacer un modelo con estimulación en ambos extremos (dos Iclamps)



Init (mV) -65
Continue til (ms) 5
Continue for (ms) 1
t (ms) 2.4
Tstop (ms) ✓ 15
dt (ms) 0.025
Points plotted/ms 40
Scrn update invl (s) 0.05
Real Time (s) 2.92

IClamp[1]
at: axon(1)

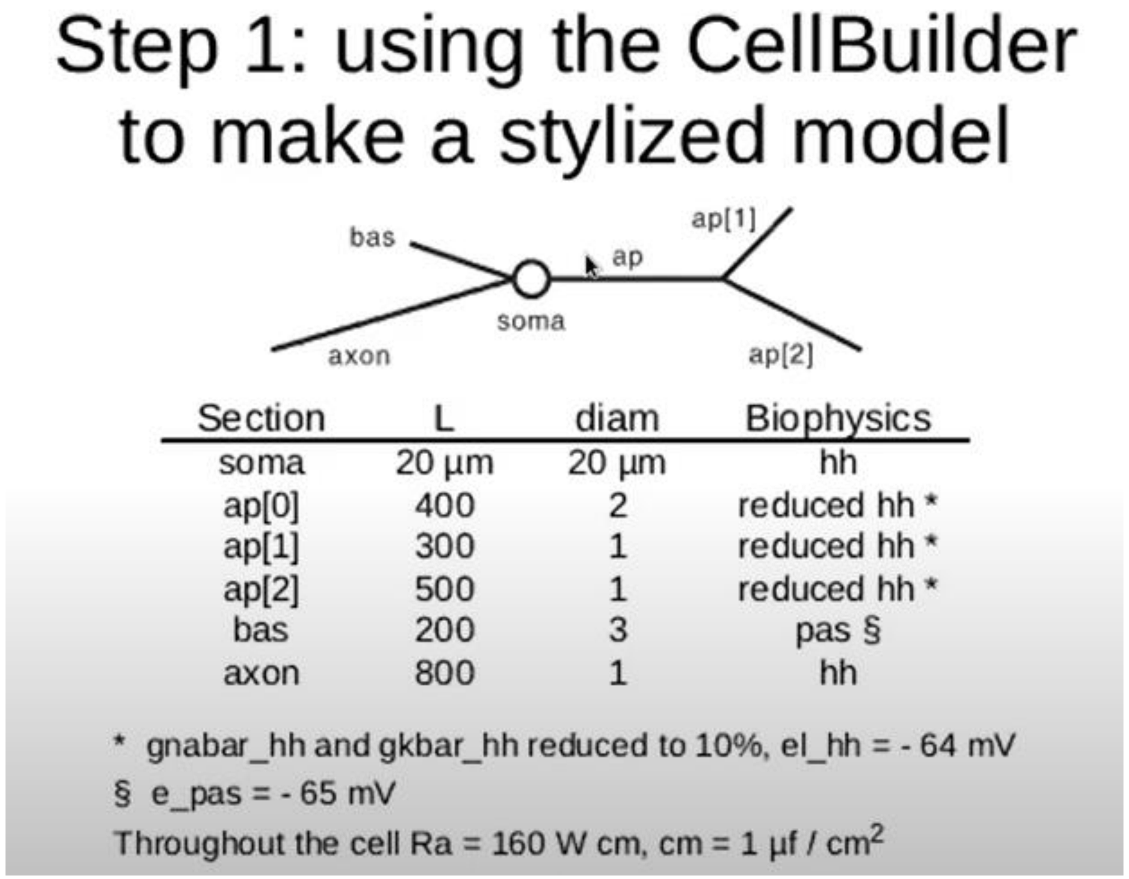
IClamp[0]
at: axon(0)

Seconds per step (s) ✓ 0.03

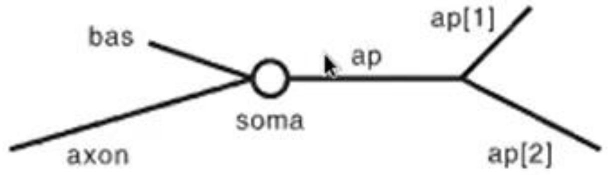
## Introducción a NEURON / NetPyNE

-> Construyendo un modelo con **extensión espacial**

- Modelo de varios compartimientos



Step 1: using the CellBuilder to make a stylized model



| Section | L | diam | Biophysics |
|---|---|---|---|
| soma | 20 µm | 20 µm | hh |
| ap[0] | 400 | 2 | reduced hh * |
| ap[1] | 300 | 1 | reduced hh * |
| ap[2] | 500 | 1 | reduced hh * |
| bas | 200 | 3 | pas § |
| axon | 800 | 1 | hh |

* gnabar_hh and gkbar_hh reduced to 10%, el_hh = - 64 mV

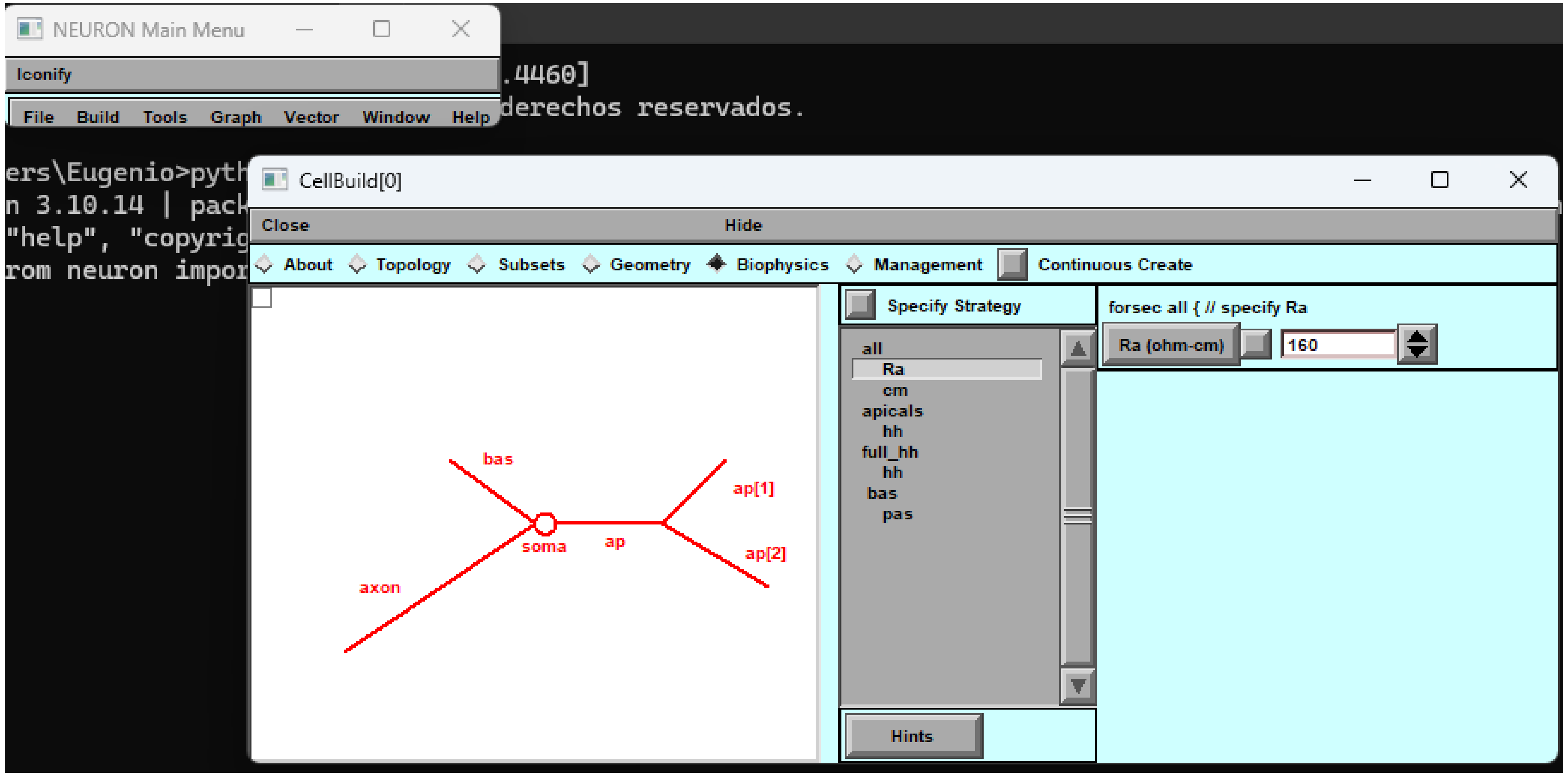
§ e_pas = - 65 mV

Throughout the cell Ra = 160 W cm, cm = 1 µf / cm$^2$

# Introducción a NEURON / NetPyNE

-> Construyendo un modelo con **extensión espacial**

- Modelo de varios compartimientos



```
.4460]
derechos reservados.

ers\Eugenio>pyth
n 3.10.14 | pack
"help", "copyrig
rom neuron impor
```

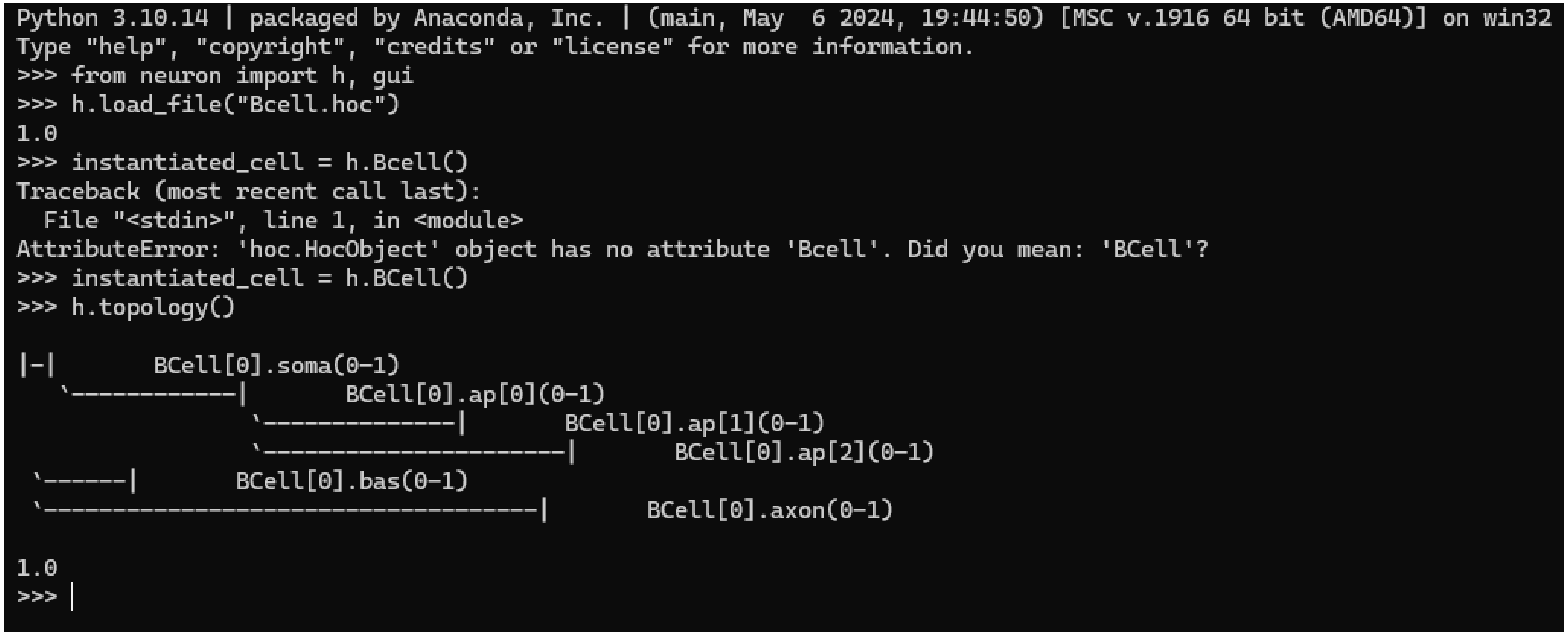
```
forsec all { // specify Ra
Ra (ohm-cm) 160
```

Introducción a NEURON / NetPyNE

-> Construyendo un modelo con **extensión espacial**

- Exportar como .hoc (en CellBuilder -> Management: Nombrar Bcell a la Classname y “Save hoc code in file”)

- Luego en Python se puede hacer:

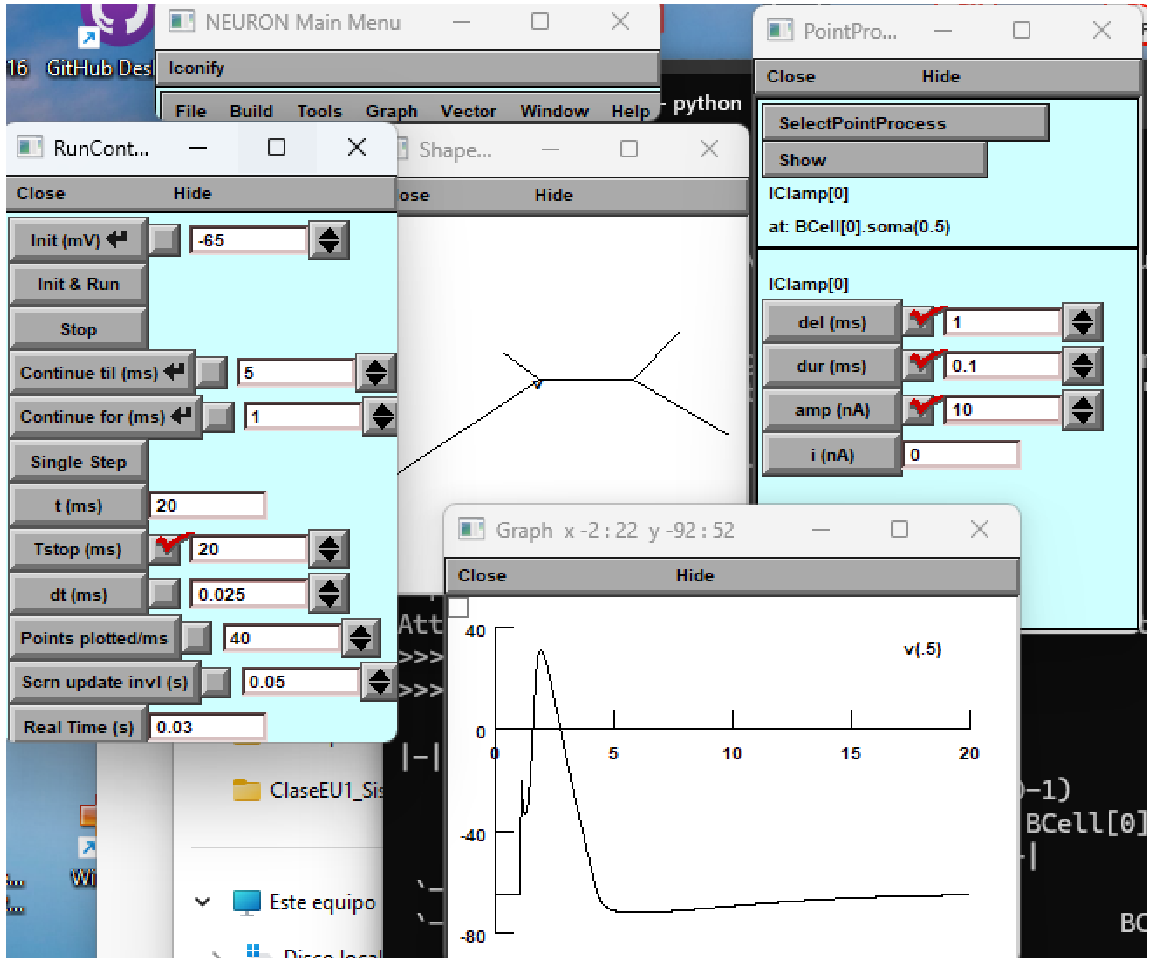


```
Python 3.10.14 | packaged by Anaconda, Inc. | (main, May  6 2024, 19:44:50) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from neuron import h, gui
>>> h.load_file("Bcell.hoc")
1.0
>>> instantiated_cell = h.Bcell()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'hoc.HocObject' object has no attribute 'Bcell'. Did you mean: 'BCell'?
>>> instantiated_cell = h.BCell()
>>> h.topology()

|-|       BCell[0].soma(0-1)
   `------------|       BCell[0].ap[0](0-1)
                 `---------------|       BCell[0].ap[1](0-1)
                 `-----------------------|       BCell[0].ap[2](0-1)
 `------|       BCell[0].bas(0-1)
 `-------------------------------------|       BCell[0].axon(0-1)

1.0
>>> |
```

# Introducción a NEURON / NetPyNE

-> Construyendo un modelo con **extensión espacial**

- Generemos un experimento:
  - Main menu: Tools -> RunControl
  - Main menu: Graph -> Voltage
  - Main menu: Tools -> Point Processes
    - > Managers
    -> Point Manager
    -> IClamp (1, 0.1, 10)



RunCont...

Close Hide

Init (mV) -65
Init & Run
Stop
Continue til (ms) 5
Continue for (ms) 1
Single Step
t (ms) 20
Tstop (ms) ✓ 20
dt (ms) 0.025
Points plotted/ms 40
Scrn update invl (s) 0.05
Real Time (s) 0.03

PointPro...

Close Hide

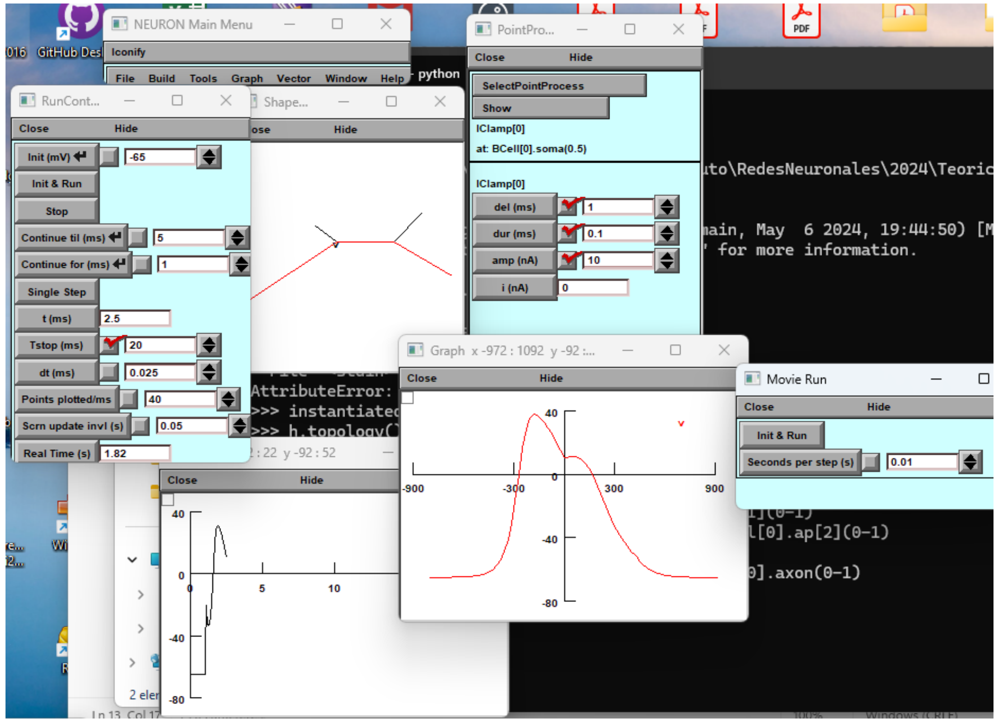
SelectPointProcess
Show
IClamp[0]
at: BCell[0].soma(0.5)

IClamp[0]
del (ms) ✓ 1
dur (ms) ✓ 0.1
amp (nA) ✓ 10
i (nA) 0

# Introducción a NEURON / NetPyNE

-> Construyendo un modelo con **extensión espacial**

- Hacemos un ShapePlot y creamos un SpacePlot a lo largo de una secuencia de secciones (del axón al ap[2])
- Mirar la simulación con "Continue for" o con Tools -> MovieRun



Init (mV) -65
Continue til (ms) 5
Continue for (ms) 1
t (ms) 2.5
Tstop (ms) 20
dt (ms) 0.025
Points plotted/ms 40
Scrn update invl (s) 0.05
Real Time (s) 1.82

IClamp[0]
at: BCell[0].soma(0.5)

IClamp[0]
del (ms) 1
dur (ms) 0.1
amp (nA) 10
i (nA) 0

Seconds per step (s) 0.01

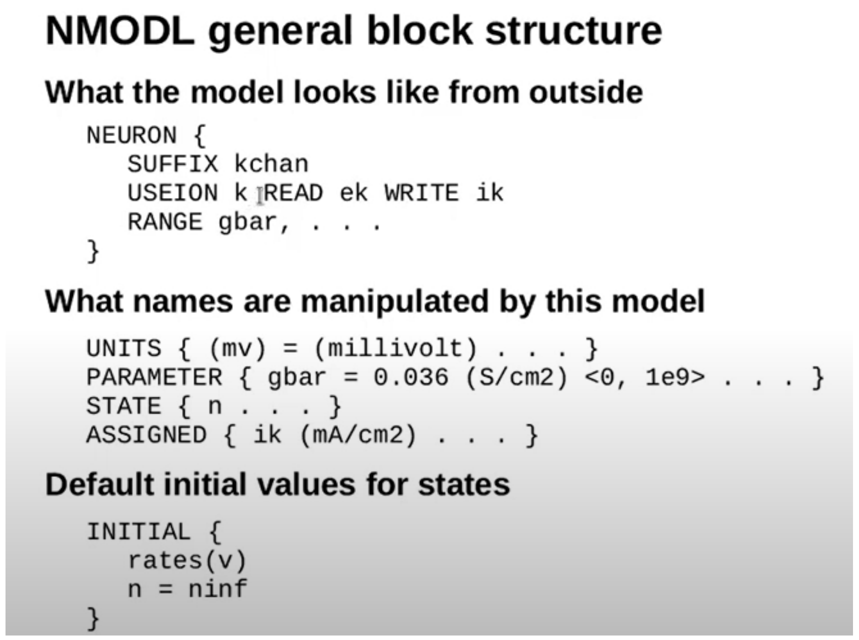
```
uto\RedesNeuronales\2024\Teoric

main, May  6 2024, 19:44:50) [M
' for more information.

AttributeError:
>>> instantiated
>>> h.topology(

][0-1)
l[0].ap[2](0-1)

0].axon(0-1)
```

- Ver efecto en el axón (el disparo no se inicia en el soma, aún cuando se inyecta ahí)

# Introducción a NEURON / NetPyNE

-> Construyendo un modelo con **extensión espacial**

- Agregando canales y biofísica: nmodl
  - Canales iónicos
  - Dinámica de acumulación (ligando / ion)
  - Difusión
  - Transporte
  - Reacciones gobernadas por ODEs, esquemas cinéticos
  - Mecanismos sinápticos, incluyendo plasticidad



**NMODL general block structure**

**What the model looks like from outside**

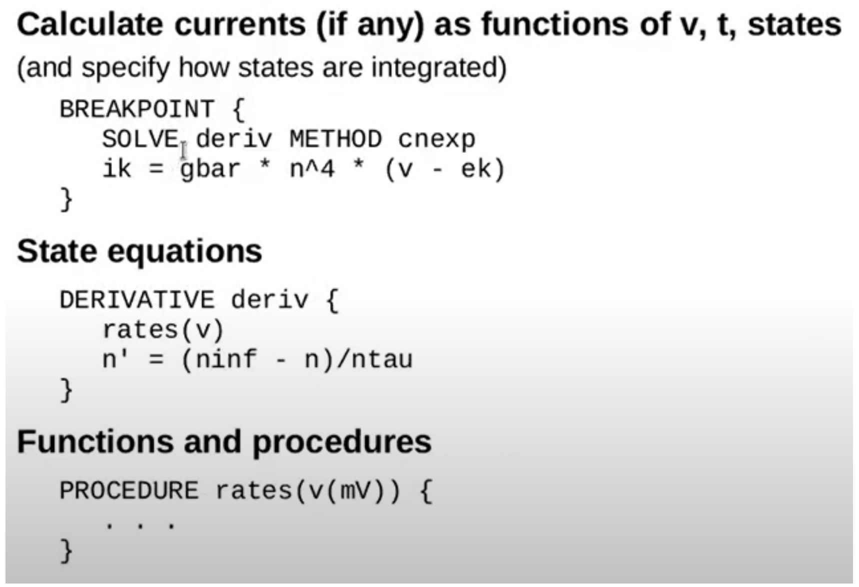
```
NEURON {
   SUFFIX kchan
   USEION k READ ek WRITE ik
   RANGE gbar, . . .
}
```

**What names are manipulated by this model**

```
UNITS { (mv) = (millivolt) . . . }
PARAMETER { gbar = 0.036 (S/cm2) <0, 1e9> . . . }
STATE { n . . . }
ASSIGNED { ik (mA/cm2) . . . }
```

**Default initial values for states**

```
INITIAL {
   rates(v)
   n = ninf
}
```



**Calculate currents (if any) as functions of v, t, states**

(and specify how states are integrated)

```
BREAKPOINT {
  SOLVE deriv METHOD cnexp
  ik = gbar * n^4 * (v - ek)
}
```

**State equations**

```
DERIVATIVE deriv {
  rates(v)
  n' = (ninf - n)/ntau
}
```

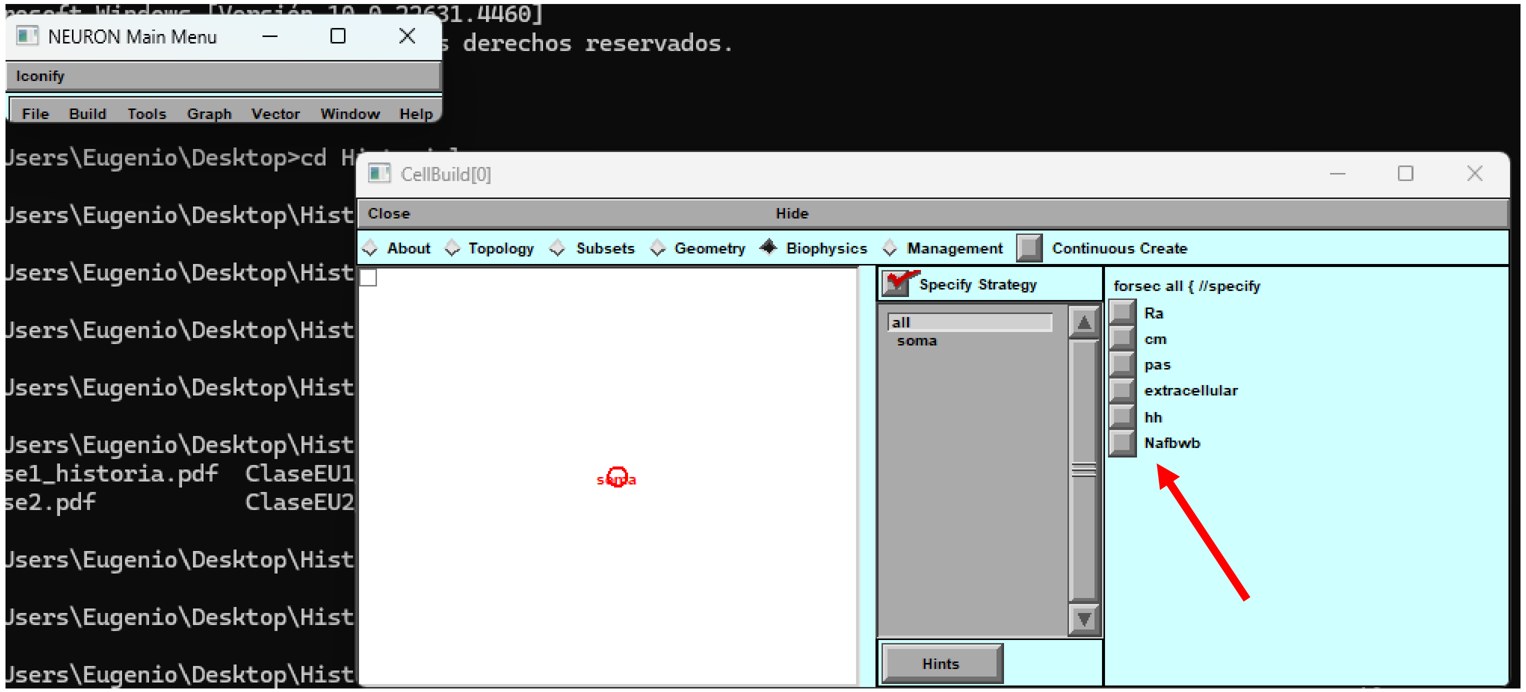
**Functions and procedures**

```
PROCEDURE rates(v(mV)) {
   . . .
}
```

# Introducción a NEURON / NetPyNE

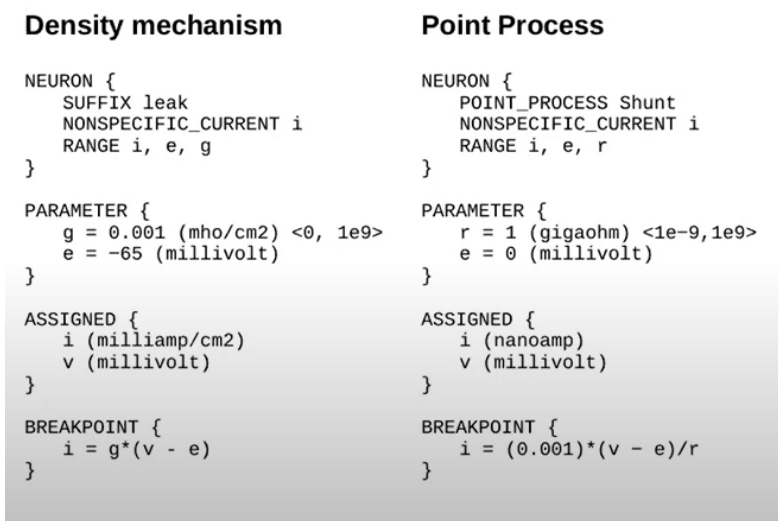
-> Construyendo un modelo con **extensión espacial**

- Una vez que está compilado (nrnivmodl), el mecanismo está presente para incorporarse en una sección (distribuído) o en una locación (point process)



```
rosoft Windows [Versión 10.0.22631.4460]
s derechos reservados.

Users\Eugenio\Desktop>cd Hi

Users\Eugenio\Desktop\Hist

Users\Eugenio\Desktop\Hist

Users\Eugenio\Desktop\Hist

Users\Eugenio\Desktop\Hist

Users\Eugenio\Desktop\Hist
se1_historia.pdf   ClaseEU1
se2.pdf            ClaseEU2

Users\Eugenio\Desktop\Hist

Users\Eugenio\Desktop\Hist

Users\Eugenio\Desktop\Hist
```

```
forsec all { //specify
Ra
cm
pas
extracellular
hh
Nafbwb
```

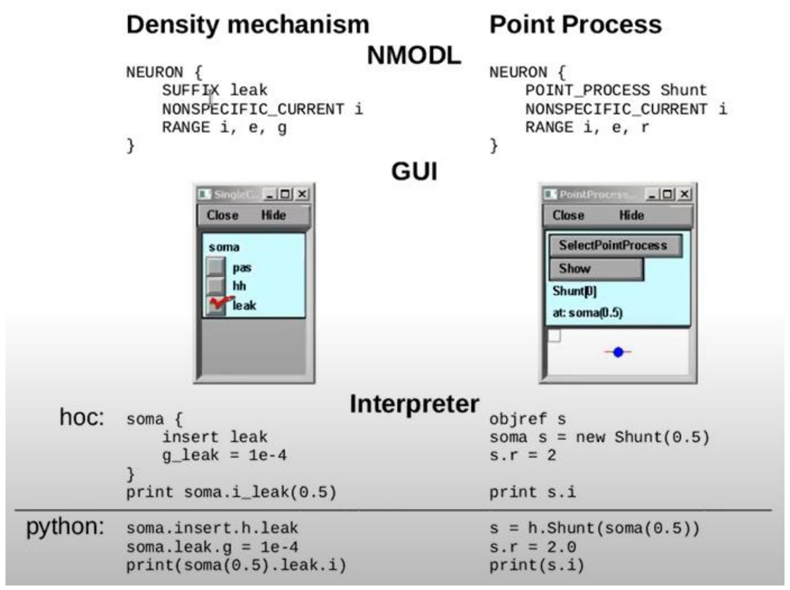


**Density mechanism**

```
NEURON {
   SUFFIX leak
   NONSPECIFIC_CURRENT i
   RANGE i, e, g
}

PARAMETER {
   g = 0.001 (mho/cm2) <0, 1e9>
   e = -65 (millivolt)
}

ASSIGNED {
   i (milliamp/cm2)
   v (millivolt)
}

BREAKPOINT {
   i = g*(v - e)
}
```

**Point Process**

```
NEURON {
   POINT_PROCESS Shunt
   NONSPECIFIC_CURRENT i
   RANGE i, e, r
}

PARAMETER {
   r = 1 (gigaohm) <1e-9,1e9>
   e = 0 (millivolt)
}

ASSIGNED {
   i (nanoamp)
   v (millivolt)
}

BREAKPOINT {
   i = (0.001)*(v - e)/r
}
```



Density mechanism — NMODL:

```
NEURON {
   SUFFIX leak
   NONSPECIFIC_CURRENT i
   RANGE i, e, g
}
```

Point Process — NMODL:

```
NEURON {
   POINT_PROCESS Shunt
   NONSPECIFIC_CURRENT i
   RANGE i, e, r
}
```

hoc:

```
soma {
   insert leak
   g_leak = 1e-4
}
print soma.i_leak(0.5)
```

```
objref s
soma s = new Shunt(0.5)
s.r = 2

print s.i
```

python:

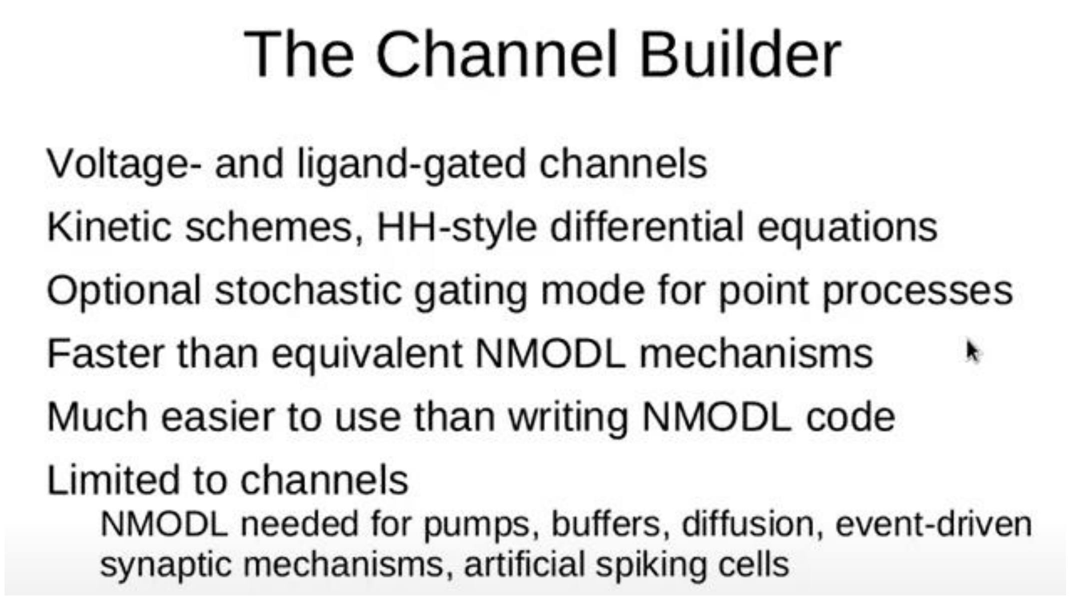
```
soma.insert.h.leak
soma.leak.g = 1e-4
print(soma(0.5).leak.i)
```

```
s = h.Shunt(soma(0.5))
s.r = 2.0
print(s.i)
```

## Introducción a NEURON / NetPyNE

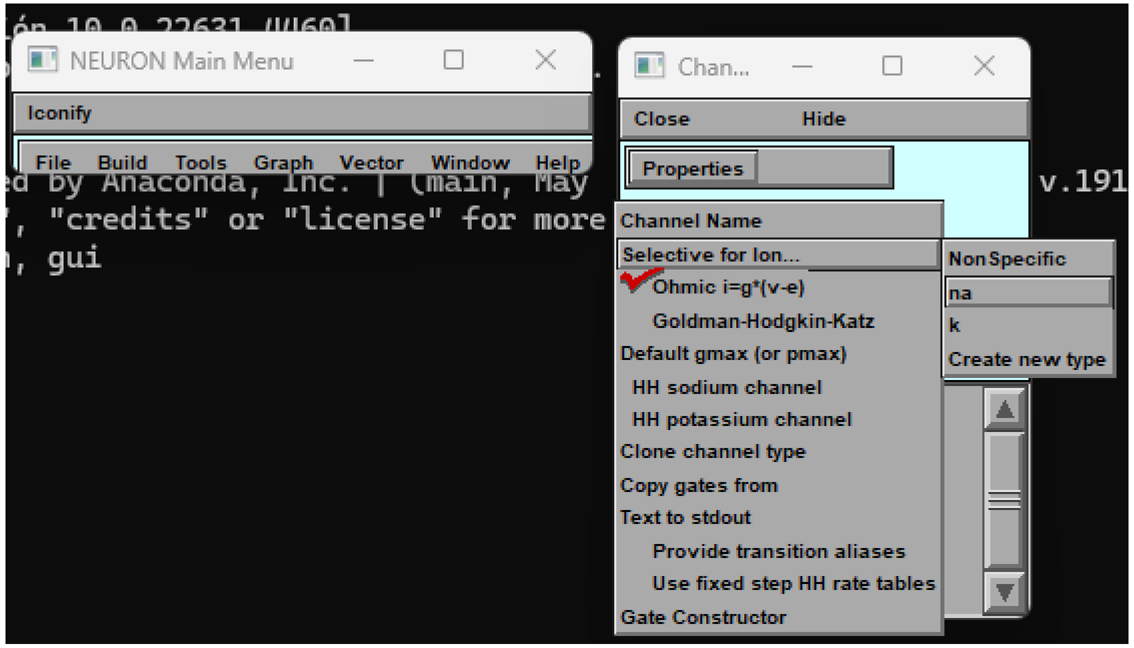
-> Construyendo un modelo con **extensión espacial**

- Channel Builder
  - Main manu -> Build -> Channel Builder -> Density
  - Properties -> Channel Name: myNa
  - Selective for ion …



### The Channel Builder

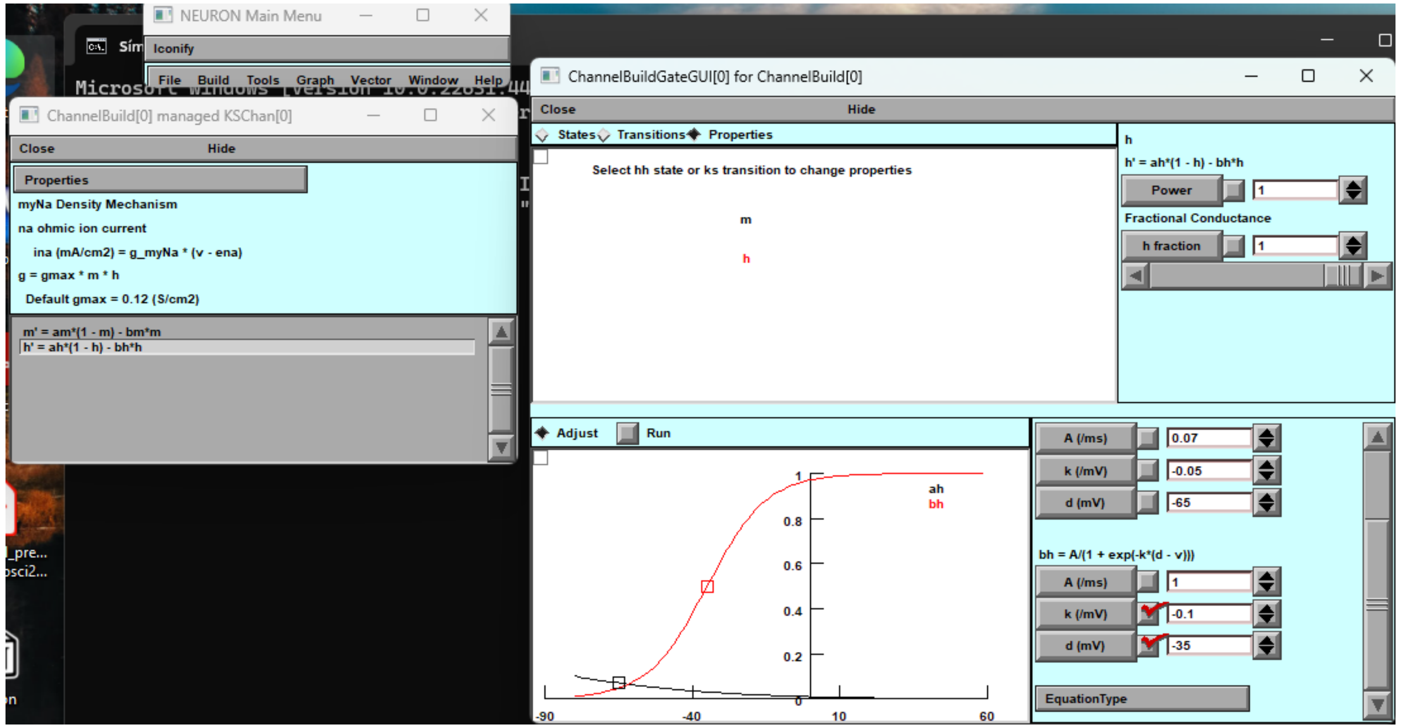
Voltage- and ligand-gated channels
Kinetic schemes, HH-style differential equations
Optional stochastic gating mode for point processes
Faster than equivalent NMODL mechanisms
Much easier to use than writing NMODL code
Limited to channels
NMODL needed for pumps, buffers, diffusion, event-driven synaptic mechanisms, artificial spiking cells



# Introducción a NEURON / NetPyNE

-> Construyendo un modelo con **extensión espacial**

- Channel Builder
  - Default gmax: Setear 0.12 S/cm2
  - Seleccionar debajo de todo “Select here to construct gates”



ChannelBuild[0] managed KSChan[0]

Properties

myNa Density Mechanism

na ohmic ion current

ina (mA/cm2) = g_myNa * (v - ena)

g = gmax * m * h

Default gmax = 0.12 (S/cm2)

m' = am*(1 - m) - bm*m

h' = ah*(1 - h) - bh*h

ChannelBuildGateGUI[0] for ChannelBuild[0]

Select hh state or ks transition to change properties

h

h' = ah*(1 - h) - bh*h

Power: 1

Fractional Conductance

h fraction: 1

A (/ms): 0.07

k (/mV): -0.05

d (mV): -65

bh = A/(1 + exp(-k*(d - v)))

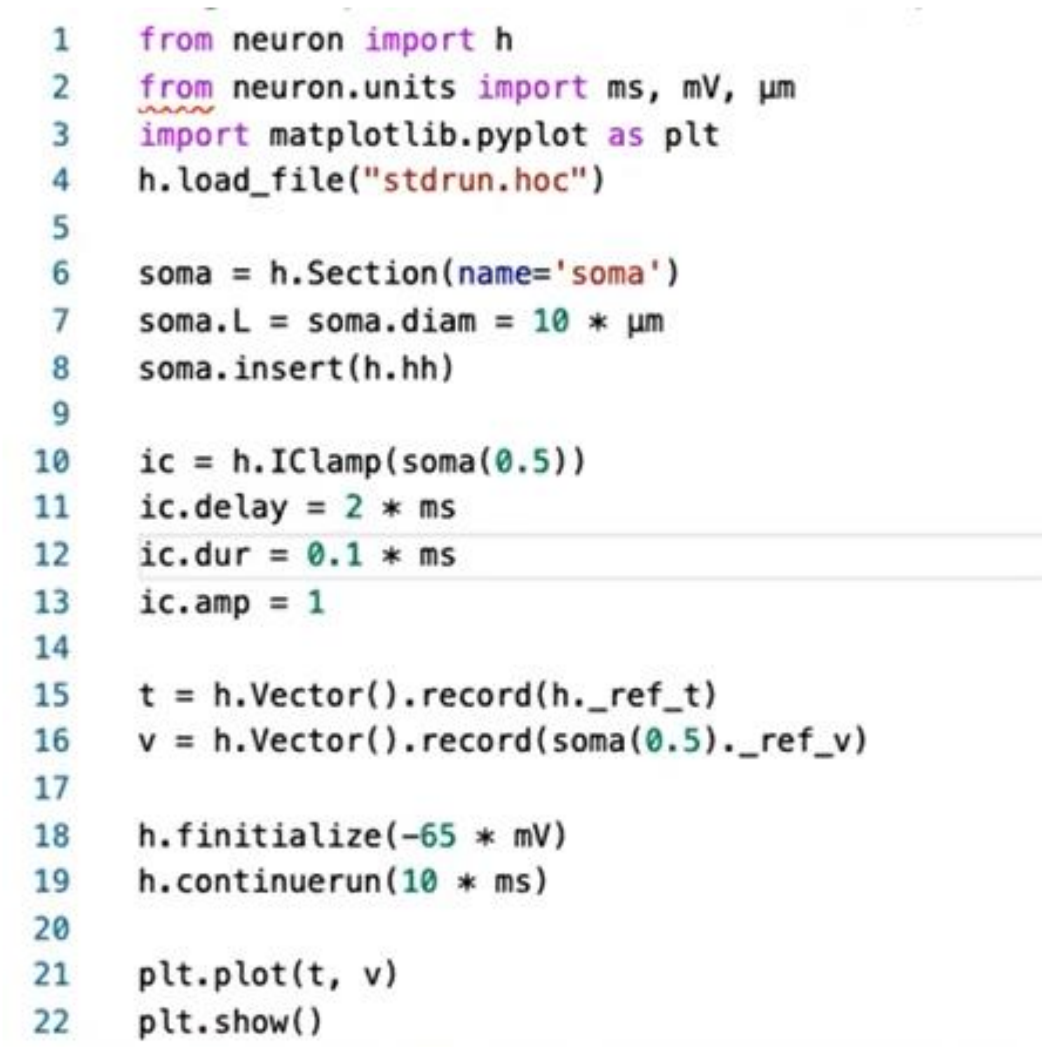
A (/ms): 1

k (/mV): -0.1

d (mV): -35

# Introducción a NEURON / NetPyNE

-> Modelando via python

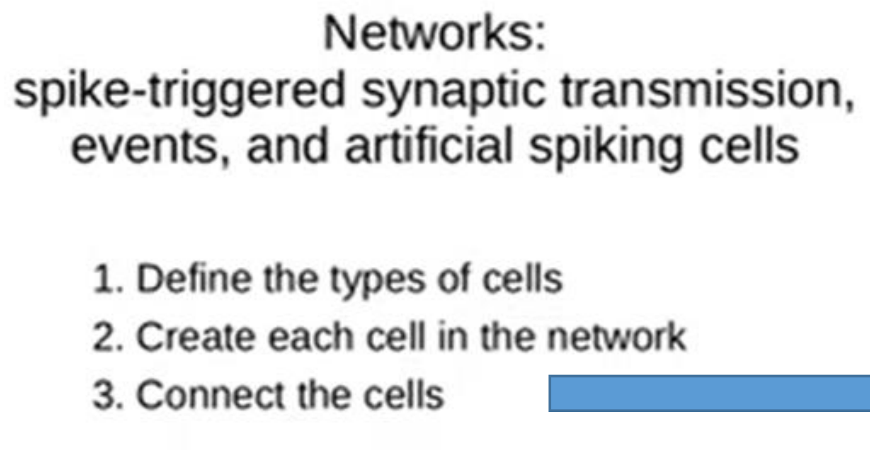


```python
from neuron import h
from neuron.units import ms, mV, µm
import matplotlib.pyplot as plt
h.load_file("stdrun.hoc")

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()
```

- Ball & Stick

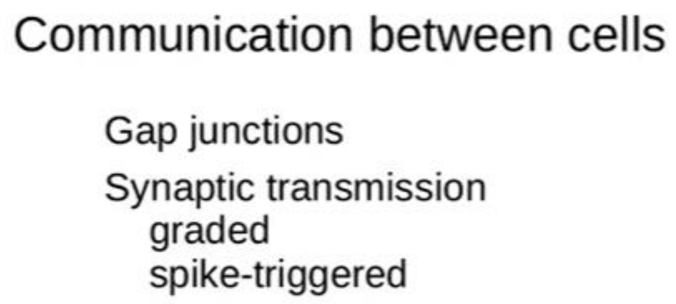
## Introducción a NEURON / NetPyNE

-> Construyendo modelos de redes



Networks:
spike-triggered synaptic transmission,
events, and artificial spiking cells

1. Define the types of cells
2. Create each cell in the network
3. Connect the cells



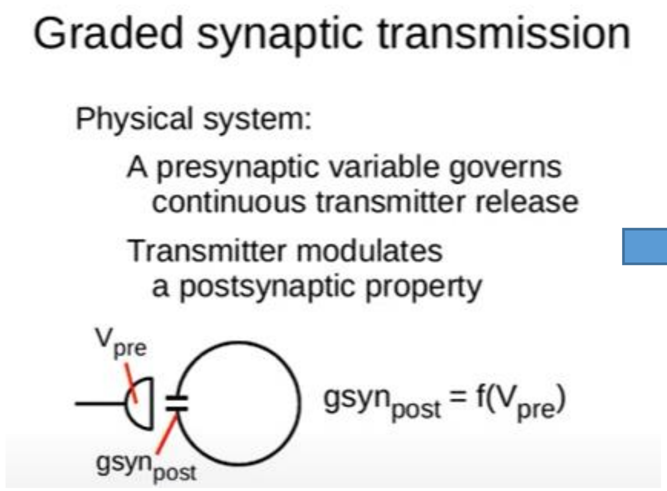
Communication between cells

Gap junctions

Synaptic transmission
graded
spike-triggered

# Introducción a NEURON / NetPyNE

-> Construyendo modelos de redes



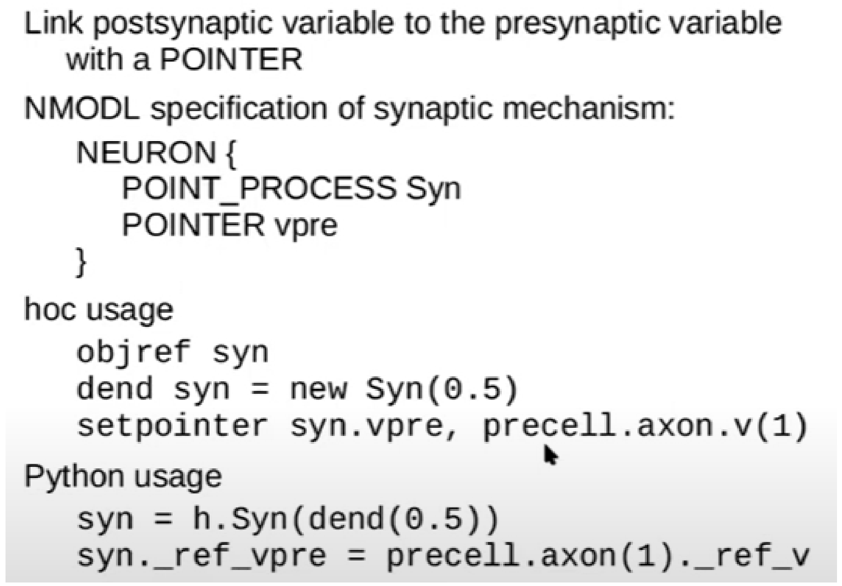
## Graded synaptic transmission

Physical system:

- A presynaptic variable governs continuous transmitter release
- Transmitter modulates a postsynaptic property

$V_{pre}$
$gsyn_{post}$

$gsyn_{post} = f(V_{pre})$



Link postsynaptic variable to the presynaptic variable with a POINTER

NMODL specification of synaptic mechanism:

```
NEURON {
    POINT_PROCESS Syn
    POINTER vpre
}
```

hoc usage

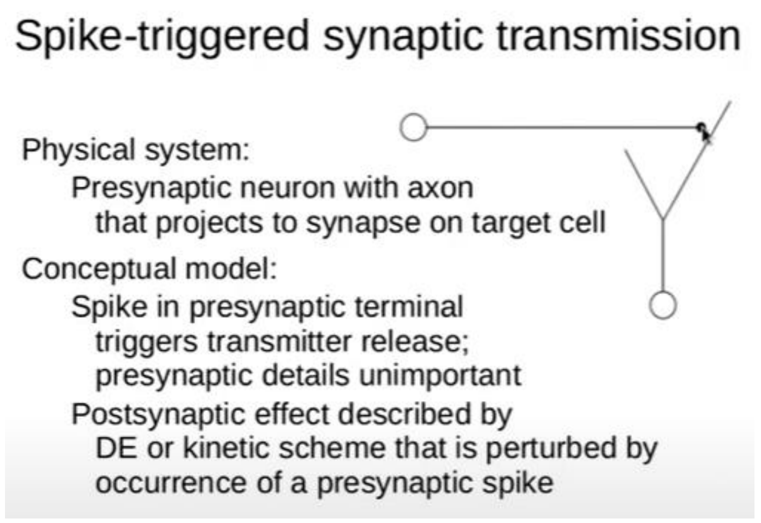
```
objref syn
dend syn = new Syn(0.5)
setpointer syn.vpre, precell.axon.v(1)
```

Python usage

```
syn = h.Syn(dend(0.5))
syn._ref_vpre = precell.axon(1)._ref_v
```

# Introducción a NEURON / NetPyNE

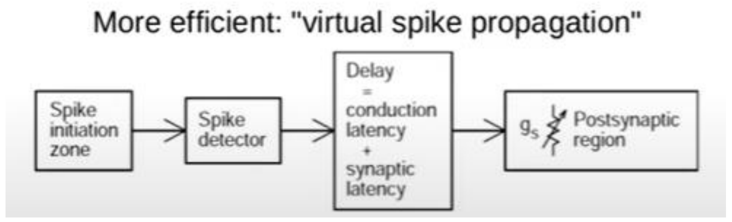
-> Construyendo modelos de redes

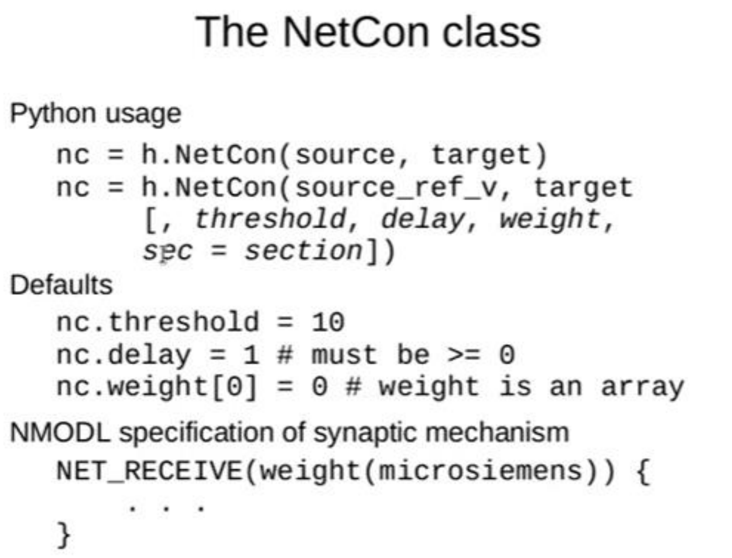


## Spike-triggered synaptic transmission

Physical system:
- Presynaptic neuron with axon that projects to synapse on target cell

Conceptual model:
- Spike in presynaptic terminal triggers transmitter release; presynaptic details unimportant
- Postsynaptic effect described by DE or kinetic scheme that is perturbed by occurrence of a presynaptic spike


More efficient: "virtual spike propagation"
Spike initiation zone
Spike detector
Delay = conduction latency + synaptic latency
$g_s$
Postsynaptic region



## The NetCon class

Python usage

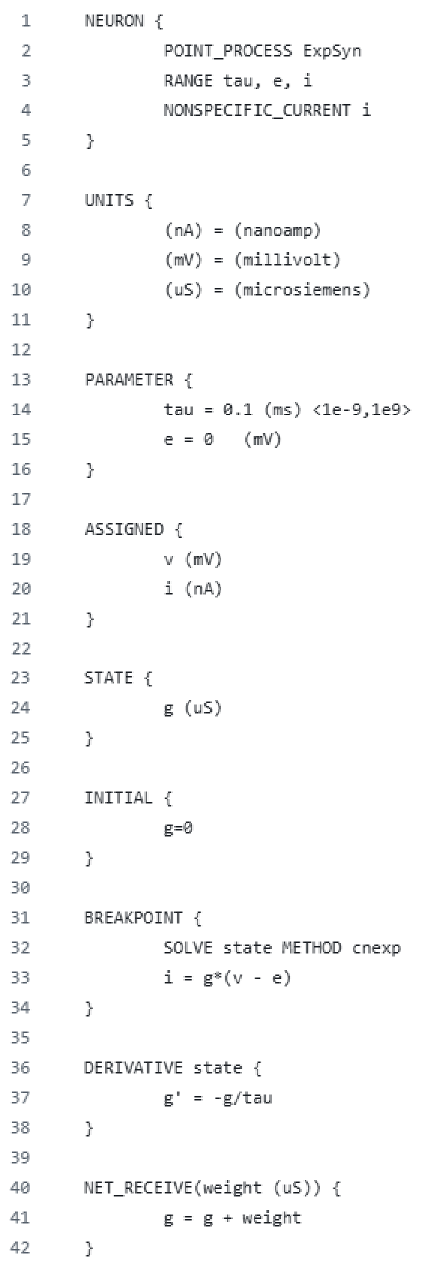
```
nc = h.NetCon(source, target)
nc = h.NetCon(source_ref_v, target
     [, threshold, delay, weight,
     sec = section])
```

Defaults

```
nc.threshold = 10
nc.delay = 1 # must be >= 0
nc.weight[0] = 0 # weight is an array
```

NMODL specification of synaptic mechanism

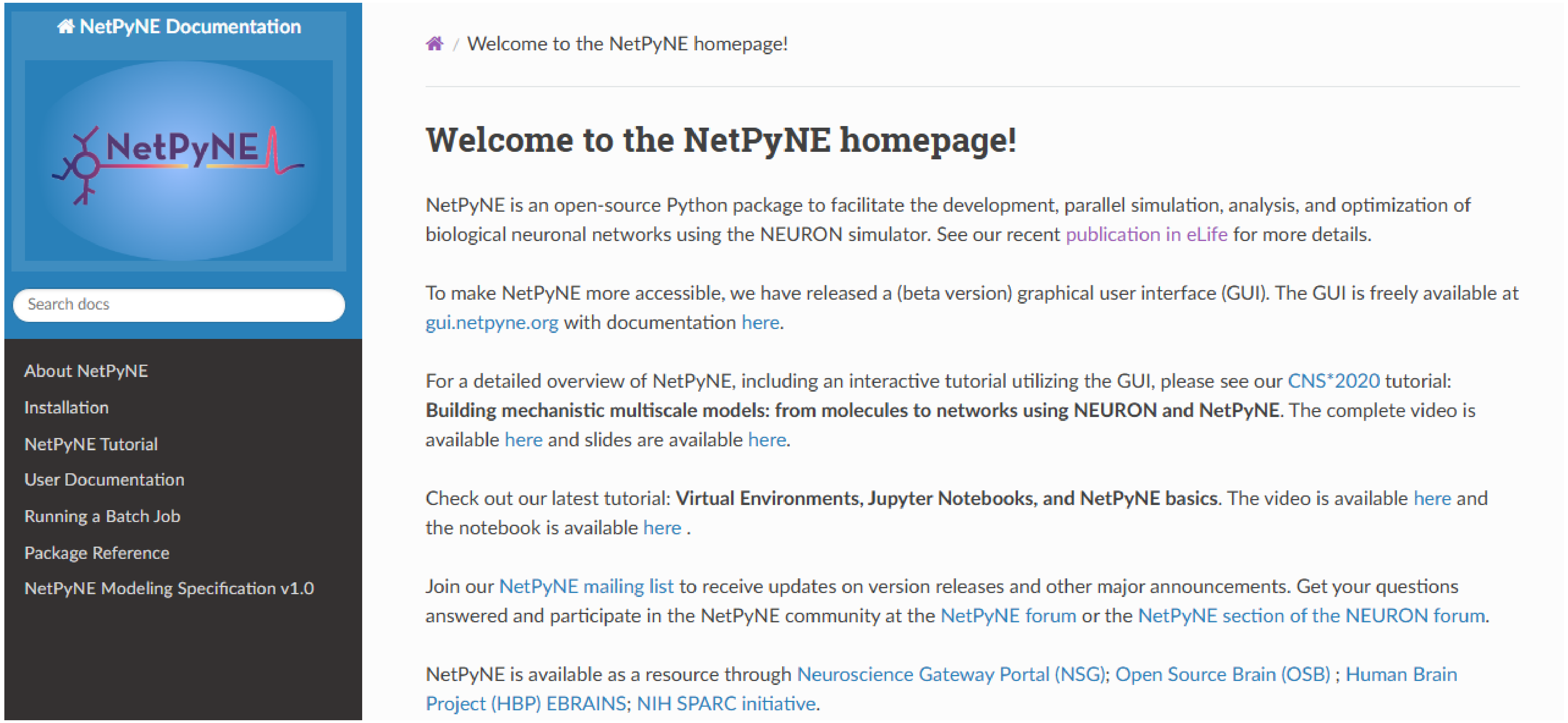
```
NET_RECEIVE(weight(microsiemens)) {
    . . .
}
```



```
NEURON {
        POINT_PROCESS ExpSyn
        RANGE tau, e, i
        NONSPECIFIC_CURRENT i
}

UNITS {
        (nA) = (nanoamp)
        (mV) = (millivolt)
        (uS) = (microsiemens)
}

PARAMETER {
        tau = 0.1 (ms) <1e-9,1e9>
        e = 0   (mV)
}

ASSIGNED {
        v (mV)
        i (nA)
}

STATE {
        g (uS)
}

INITIAL {
        g=0
}

BREAKPOINT {
        SOLVE state METHOD cnexp
        i = g*(v - e)
}

DERIVATIVE state {
        g' = -g/tau
}

NET_RECEIVE(weight (uS)) {
        g = g + weight
}
```

## Introducción a NEURON / NetPyNE

-> Overview de NetPyNE



/ Welcome to the NetPyNE homepage!

**Welcome to the NetPyNE homepage!**

NetPyNE is an open-source Python package to facilitate the development, parallel simulation, analysis, and optimization of biological neuronal networks using the NEURON simulator. See our recent publication in eLife for more details.

To make NetPyNE more accessible, we have released a (beta version) graphical user interface (GUI). The GUI is freely available at gui.netpyne.org with documentation here.

For a detailed overview of NetPyNE, including an interactive tutorial utilizing the GUI, please see our CNS*2020 tutorial: **Building mechanistic multiscale models: from molecules to networks using NEURON and NetPyNE**. The complete video is available here and slides are available here.

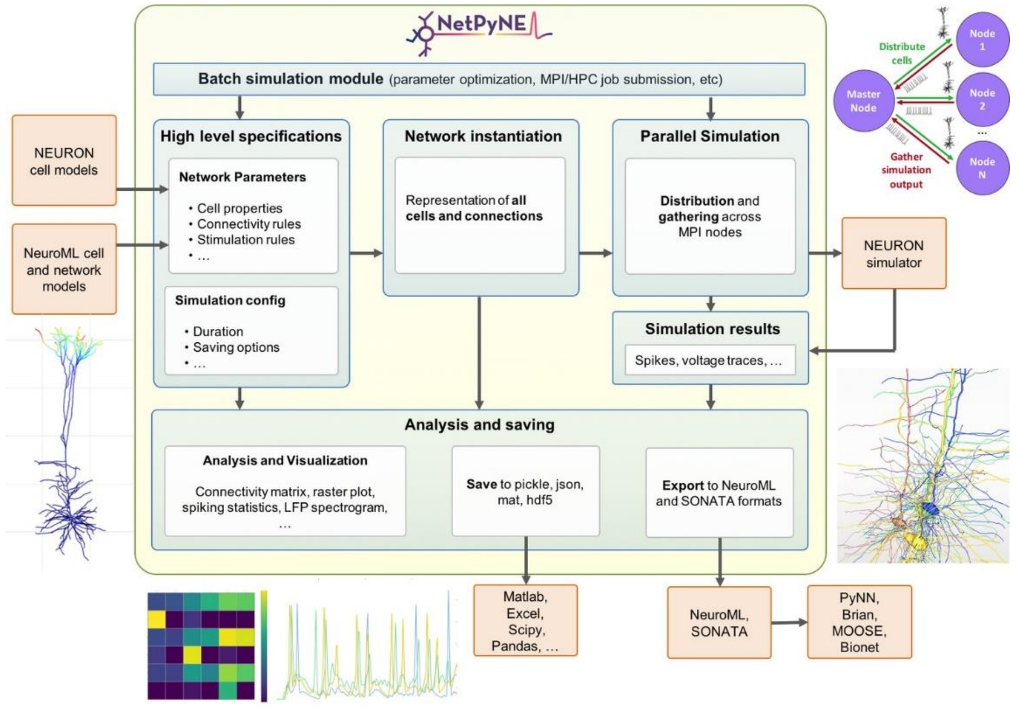
Check out our latest tutorial: **Virtual Environments, Jupyter Notebooks, and NetPyNE basics**. The video is available here and the notebook is available here .

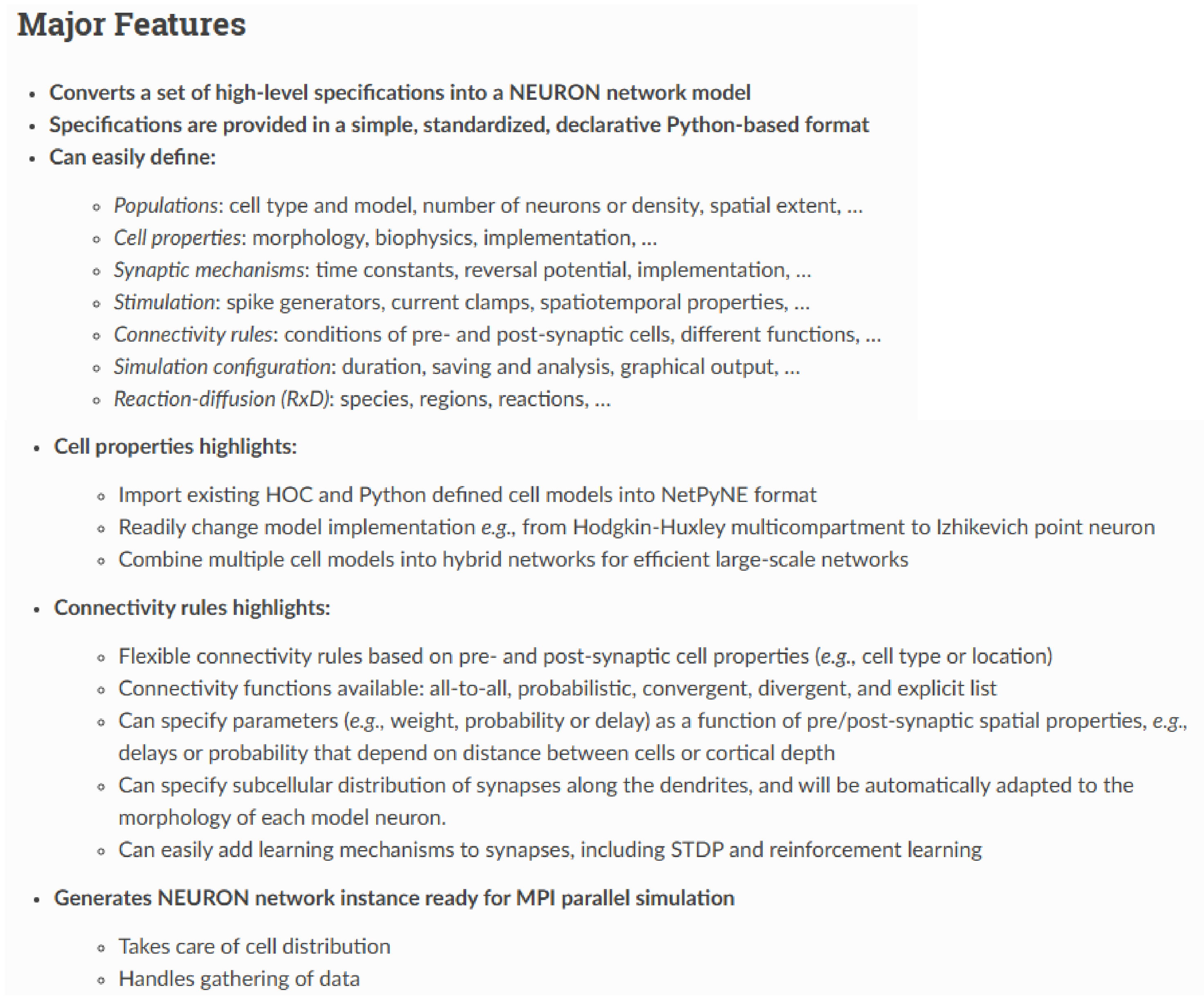
Join our NetPyNE mailing list to receive updates on version releases and other major announcements. Get your questions answered and participate in the NetPyNE community at the NetPyNE forum or the NetPyNE section of the NEURON forum.

NetPyNE is available as a resource through Neuroscience Gateway Portal (NSG); Open Source Brain (OSB) ; Human Brain Project (HBP) EBRAINS; NIH SPARC initiative.

# Introducción a NEURON / NetPyNE

-> Overview de NetPyNE


NetPyNE
Batch simulation module (parameter optimization, MPI/HPC job submission, etc)
NEURON cell models
NeuroML cell and network models
High level specifications
Network Parameters
• Cell properties
• Connectivity rules
• Stimulation rules
• ...
Simulation config
• Duration
• Saving options
• ...
Network instantiation
Representation of all cells and connections
Parallel Simulation
Distribution and gathering across MPI nodes
NEURON simulator
Simulation results
Spikes, voltage traces, ...
Distribute cells
Master Node
Node 1
Node 2
Node N
Gather simulation output
Analysis and saving
Analysis and Visualization
Connectivity matrix, raster plot, spiking statistics, LFP spectrogram, ...
Save to pickle, json, mat, hdf5
Export to NeuroML and SONATA formats
Matlab, Excel, Scipy, Pandas, ...
NeuroML, SONATA
PyNN, Brian, MOOSE, Bionet



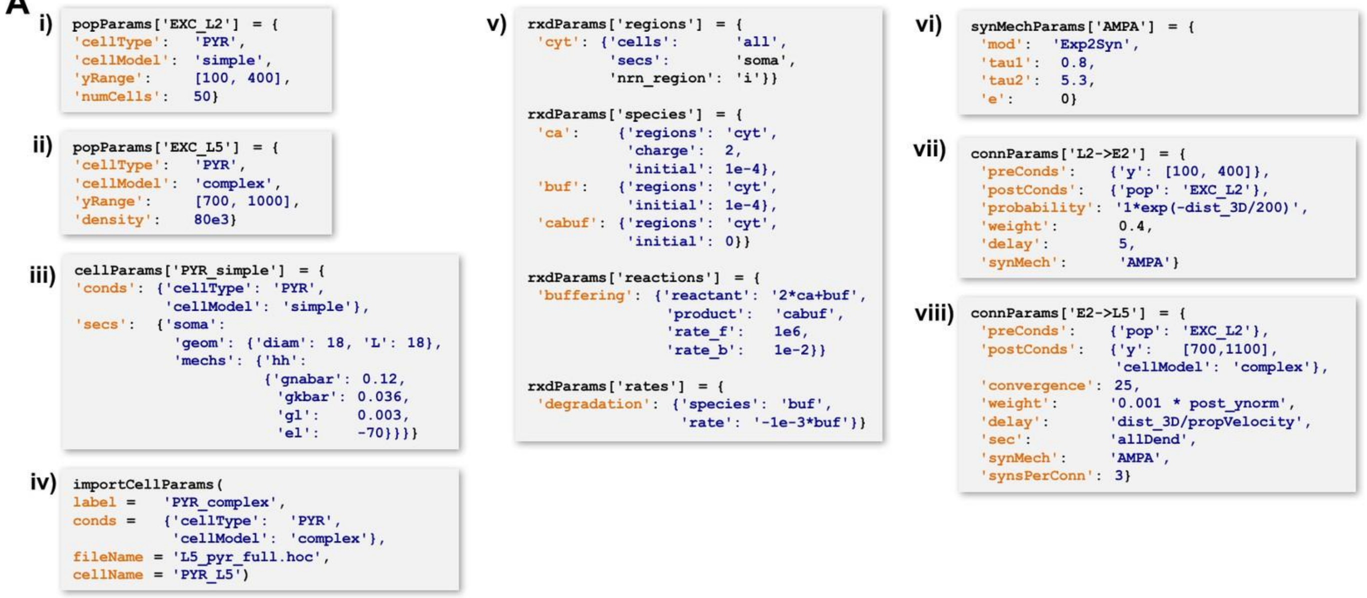
## Major Features

- **Converts a set of high-level specifications into a NEURON network model**
- **Specifications are provided in a simple, standardized, declarative Python-based format**
- **Can easily define:**
  - *Populations*: cell type and model, number of neurons or density, spatial extent, ...
  - *Cell properties*: morphology, biophysics, implementation, ...
  - *Synaptic mechanisms*: time constants, reversal potential, implementation, ...
  - *Stimulation*: spike generators, current clamps, spatiotemporal properties, ...
  - *Connectivity rules*: conditions of pre- and post-synaptic cells, different functions, ...
  - *Simulation configuration*: duration, saving and analysis, graphical output, ...
  - *Reaction-diffusion (RxD)*: species, regions, reactions, ...
- **Cell properties highlights:**
  - Import existing HOC and Python defined cell models into NetPyNE format
  - Readily change model implementation *e.g.*, from Hodgkin-Huxley multicompartment to Izhikevich point neuron
  - Combine multiple cell models into hybrid networks for efficient large-scale networks
- **Connectivity rules highlights:**
  - Flexible connectivity rules based on pre- and post-synaptic cell properties (*e.g.*, cell type or location)
  - Connectivity functions available: all-to-all, probabilistic, convergent, divergent, and explicit list
  - Can specify parameters (*e.g.*, weight, probability or delay) as a function of pre/post-synaptic spatial properties, *e.g.*, delays or probability that depend on distance between cells or cortical depth
  - Can specify subcellular distribution of synapses along the dendrites, and will be automatically adapted to the morphology of each model neuron.
  - Can easily add learning mechanisms to synapses, including STDP and reinforcement learning
- **Generates NEURON network instance ready for MPI parallel simulation**
  - Takes care of cell distribution
  - Handles gathering of data

# Introducción a NEURON / NetPyNE

-> Overview de NetPyNE

A



i)
```
popParams['EXC_L2'] = {
'cellType':   'PYR',
'cellModel':  'simple',
'yRange':     [100, 400],
'numCells':   50}
```

ii)
```
popParams['EXC_L5'] = {
'cellType':   'PYR',
'cellModel':  'complex',
'yRange':     [700, 1000],
'density':    80e3}
```

iii)
```
cellParams['PYR_simple'] = {
'conds': {'cellType': 'PYR',
          'cellModel': 'simple'},
'secs':  {'soma':
            'geom': {'diam': 18, 'L': 18},
            'mechs': {'hh':
                       {'gnabar': 0.12,
                        'gkbar': 0.036,
                        'gl':    0.003,
                        'el':    -70}}}}
```

iv)
```
importCellParams(
label =    'PYR_complex',
conds =    {'cellType':  'PYR',
            'cellModel': 'complex'},
fileName = 'L5_pyr_full.hoc',
cellName = 'PYR_L5')
```

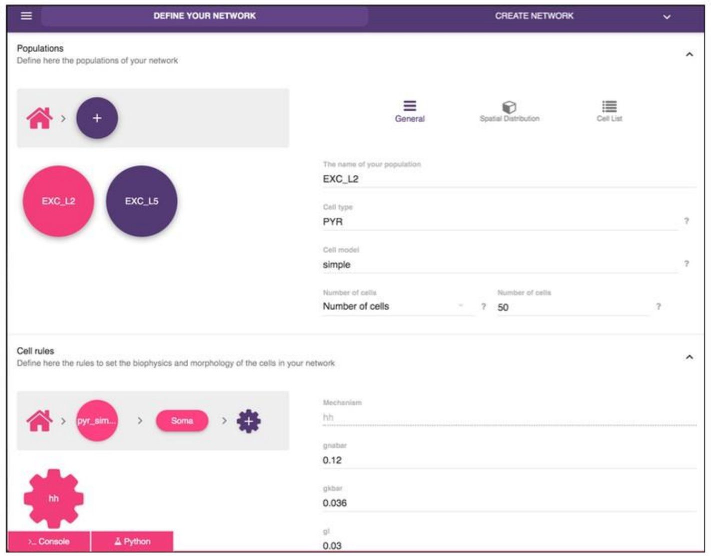
v)
```
rxdParams['regions'] = {
 'cyt': {'cells':       'all',
         'secs':        'soma',
         'nrn_region':  'i'}}

rxdParams['species'] = {
 'ca':    {'regions': 'cyt',
           'charge':  2,
           'initial': 1e-4},
 'buf':   {'regions': 'cyt',
           'initial': 1e-4},
 'cabuf': {'regions': 'cyt',
           'initial': 0}}

rxdParams['reactions'] = {
 'buffering': {'reactant': '2*ca+buf',
               'product':  'cabuf',
               'rate_f':   1e6,
               'rate_b':   1e-2}}

rxdParams['rates'] = {
 'degradation': {'species': 'buf',
                 'rate': '-1e-3*buf'}}
```

vi)
```
synMechParams['AMPA'] = {
 'mod':   'Exp2Syn',
 'tau1':  0.8,
 'tau2':  5.3,
 'e':     0}
```

vii)
```
connParams['L2->E2'] = {
 'preConds':    {'y': [100, 400]},
 'postConds':   {'pop': 'EXC_L2'},
 'probability': '1*exp(-dist_3D/200)',
 'weight':       0.4,
 'delay':        5,
 'synMech':      'AMPA'}
```

viii)
```
connParams['E2->L5'] = {
 'preConds':    {'pop': 'EXC_L2'},
 'postConds':   {'y':   [700,1100],
                 'cellModel': 'complex'},
 'convergence': 25,
 'weight':      '0.001 * post_ynorm',
 'delay':       'dist_3D/propVelocity',
 'sec':         'allDend',
 'synMech':     'AMPA',
 'synsPerConn': 3}
```

B



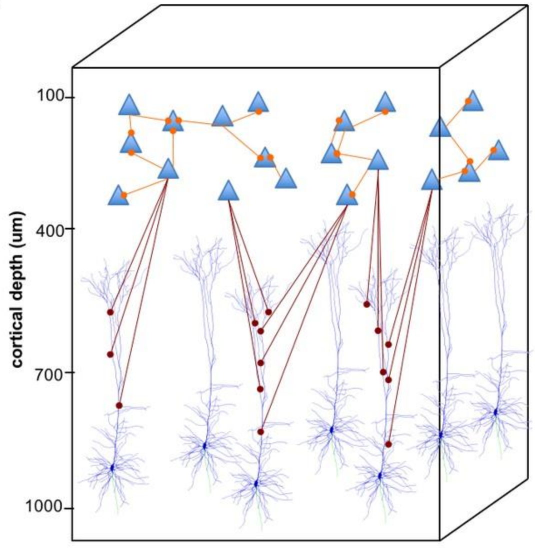
Populations
Define here the populations of your network

The name of your population: EXC_L2
Cell type: PYR
Cell model: simple
Number of cells: Number of cells
Number of cells: 50

Cell rules
Define here the rules to set the biophysics and morphology of the cells in your network

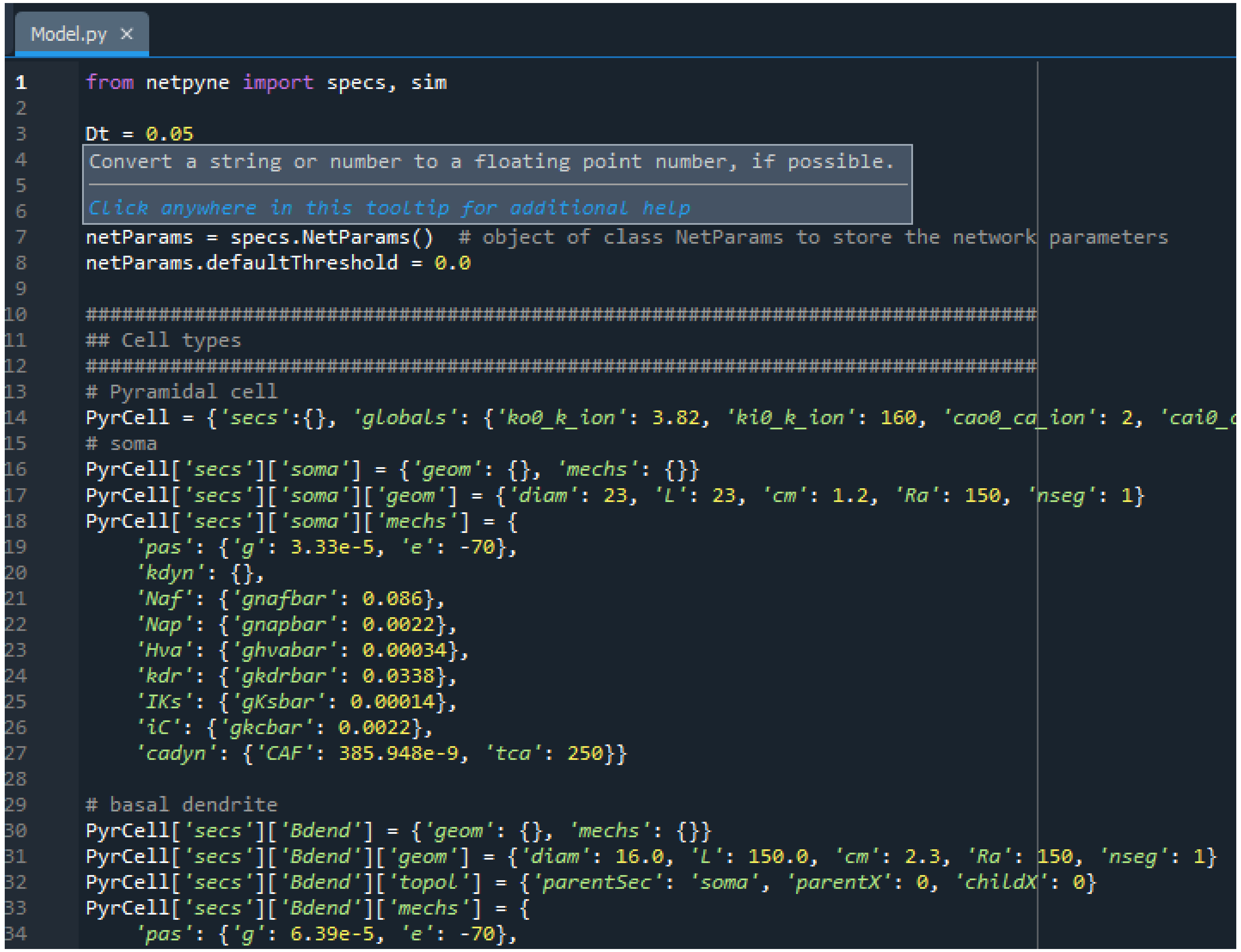
Mechanism: hh
gnabar: 0.12
gkbar: 0.036
gl: 0.03

C



# Introducción a NEURON / NetPyNE

-> Overview de NetPyNE



Model.py ×

```
from netpyne import specs, sim

Dt = 0.05
Convert a string or number to a floating point number, if possible.

Click anywhere in this tooltip for additional help
netParams = specs.NetParams()  # object of class NetParams to store the network parameters
netParams.defaultThreshold = 0.0

###############################################################################
## Cell types
###############################################################################
# Pyramidal cell
PyrCell = {'secs':{}, 'globals': {'ko0_k_ion': 3.82, 'ki0_k_ion': 160, 'cao0_ca_ion': 2, 'cai0_c
# soma
PyrCell['secs']['soma'] = {'geom': {}, 'mechs': {}}
PyrCell['secs']['soma']['geom'] = {'diam': 23, 'L': 23, 'cm': 1.2, 'Ra': 150, 'nseg': 1}
PyrCell['secs']['soma']['mechs'] = {
    'pas': {'g': 3.33e-5, 'e': -70},
    'kdyn': {},
    'Naf': {'gnafbar': 0.086},
    'Nap': {'gnapbar': 0.0022},
    'Hva': {'ghvabar': 0.00034},
    'kdr': {'gkdrbar': 0.0338},
    'IKs': {'gKsbar': 0.00014},
    'iC': {'gkcbar': 0.0022},
    'cadyn': {'CAF': 385.948e-9, 'tca': 250}}

# basal dendrite
PyrCell['secs']['Bdend'] = {'geom': {}, 'mechs': {}}
PyrCell['secs']['Bdend']['geom'] = {'diam': 16.0, 'L': 150.0, 'cm': 2.3, 'Ra': 150, 'nseg': 1}
PyrCell['secs']['Bdend']['topol'] = {'parentSec': 'soma', 'parentX': 0, 'childX': 0}
PyrCell['secs']['Bdend']['mechs'] = {
    'pas': {'g': 6.39e-5, 'e': -70},
```